FUTURE VISION BIE

One Stop for All Study Materials

& Lab Programs

Future Vision

By K B Hemanth Raj

Scan the QR Code to Visit the Web Page

Or

Visit : https://hemanthrajhemu.github.io

Gain Access to All Study Materials according to VTU,
CSE – Computer Science Engineering,
ISE – Information Science Engineering,
ECE - Electronics and Communication Engineering
& MORE...

Join Telegram to get Instant Updates: https://bit.ly/VTU_TELEGRAM

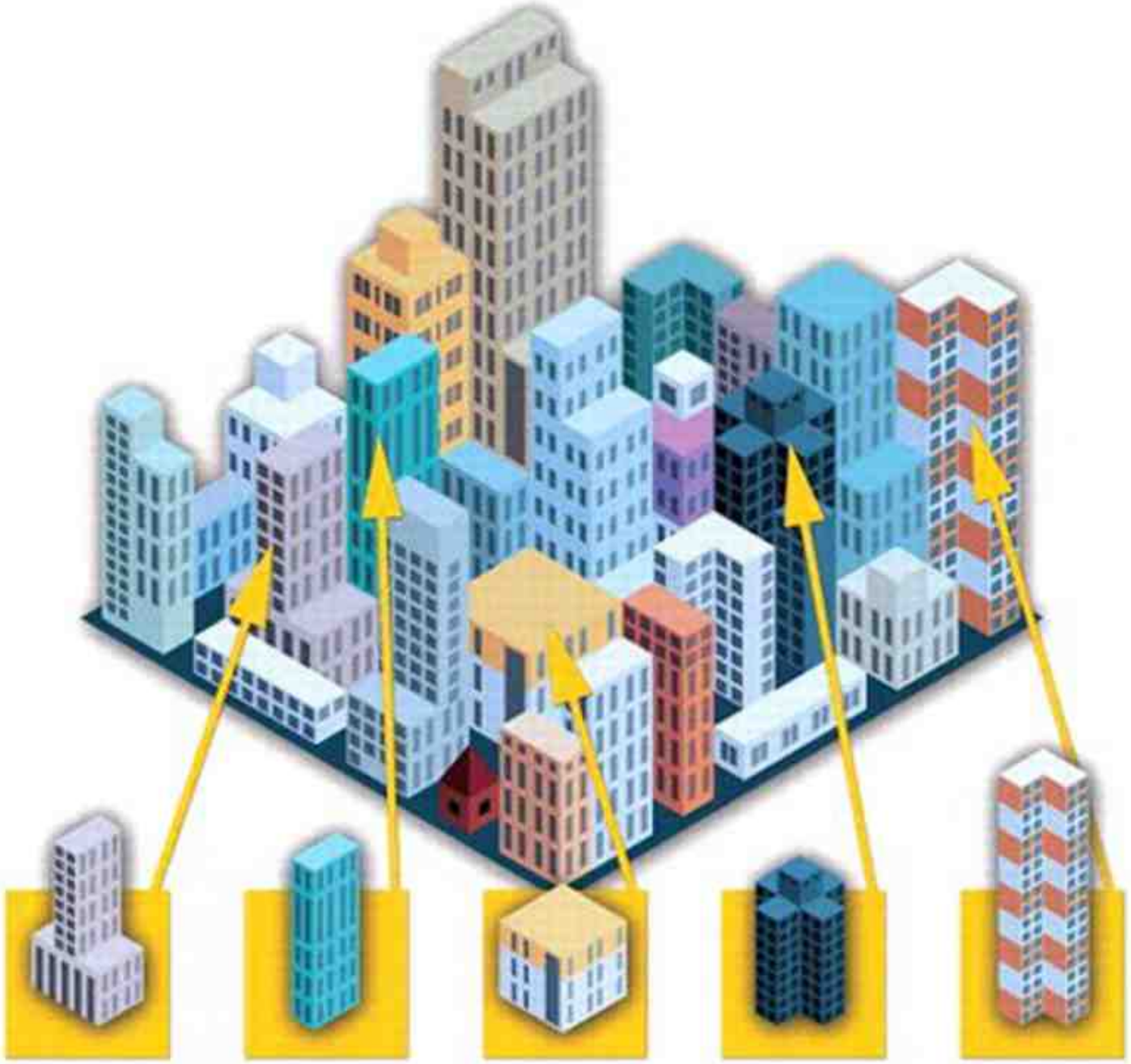
Contact: MAIL: futurevisionbie@gmail.com

INSTAGRAM: www.instagram.com/hemanthraj_hemu/

INSTAGRAM: www.instagram.com/futurevisionbie/

WHATSAPP SHARE: https://bit.ly/FVBIESHARE

FUNDAMENTALS of WEB DEVELOPMENT



RANDY CONNOLLY • RICARDO HOAR

6 JavaScript: Client-Side Scripting

CHAPTER OBJECTIVES

In this chapter you will learn . . .

- About the role of client-side scripting in web development
- How to create fail-safe design that will work even if JavaScript is not enabled
- The important syntactic elements of JavaScript
- About built-in JavaScript objects
- How to prevalidate forms using JavaScript

This chapter introduces the JavaScript (JS) client-side scripting language. Using your knowledge of CSS selectors, JavaScript can programmatically access and alter the HTML hierarchy you define in your markup. With JavaScript we can animate, move, transition, hide, and show parts of the page rather than refresh an entire page from the server. We can also do prevalidation and other logic on the client machine, reducing the number of requests to the server. This power is what makes JavaScript and JavaScript-based frameworks like jQuery crucial participants in modern web development. This chapter will introduce client programming and concepts and JavaScript syntax, including functions and classes, and then describe how JavaScript is best woven into an application together with HTML and CSS.

NOTE

It should be noted that JavaScript is not an ideal first programming language for students. Its complicated and difficult syntax mechanisms and the challenges of the client-server model make it a poor choice to learn programming. For that reason we expect the reader of this chapter to already have some familiarity with another programming language before learning about JavaScript.

6.1 What Is JavaScript and What Can It Do?

Larry Ullman, in his *Modern JavaScript: Develop and Design* (Peachpit Press, 2012), has an especially succinct definition of JavaScript: it is an object-oriented, dynamically typed, scripting language. In the context of this book, we can add as well, that it is primarily a client-side scripting language. (Though there are server-side implementations of JavaScript, in this book we are only concerned with the more common client-side version.)

Although it contains the word *Java*, JavaScript and Java are vastly different programming languages with different uses. Java is a full-fledged compiled, object-oriented language, popular for its ability to run on any platform with a Java Virtual Machine installed. Conversely, JavaScript is one of the world's most popular languages, with fewer of the object-oriented features of Java, and runs directly inside the browser, without the need for the JVM. Although there are some syntactic similarities, the two languages are not interchangeable and should not be confused with one another.

JavaScript is object oriented in that almost everything in the language is an object. For instance, variables are objects in that they have constructors, properties, and methods (more about these terms in Section 6.4.1). Unlike more familiar object-oriented languages such as Java, C#, and Visual Basic, functions in JavaScript are also objects. As you will see later in Chapter 15, the objects in JavaScript are prototype-based rather than class-based, which means that while JavaScript shares some syntactic features of Java or C#, it is also quite different from those languages. Given that JavaScript approaches objects far differently than other languages, and does not have a formal class mechanism nor inheritance syntax, we might say that it is a *strange* object-oriented language.

JavaScript is dynamically typed (also called weakly typed) in that variables can be easily (or implicitly) converted from one data type to another. In a programming language such as Java, variables are statically typed, in that the data type of a variable is defined by the programmer (e.g., `int abc`) and enforced by the compiler. With JavaScript, the type of data a variable can hold is assigned at runtime and can change during run time as well.

The final term in the above definition of JavaScript is that it is a client-side scripting language, and due to the importance of this aspect, it will be covered in a bit more detail below.

6.1.1 Client-Side Scripting

The idea of client-side scripting is an important one in web development. It refers to the client machine (i.e., the browser) running code locally rather than relying on the server to execute code and return the result. There are many client-side languages that have come into use over the past decade including Flash, VBScript, Java, and JavaScript. Some of these technologies only work in certain browsers, while others require plug-ins to function. We will focus on JavaScript due to its browser interoperability (that is, its ability to work/operate on most browsers). Figure 6.1 illustrates how a client machine downloads and executes JavaScript code.

There are many advantages of client-side scripting:

- Processing can be offloaded from the server to client machines, thereby reducing the load on the server.
- The browser can respond more rapidly to user events than a request to a remote server ever could, which improves the user experience.

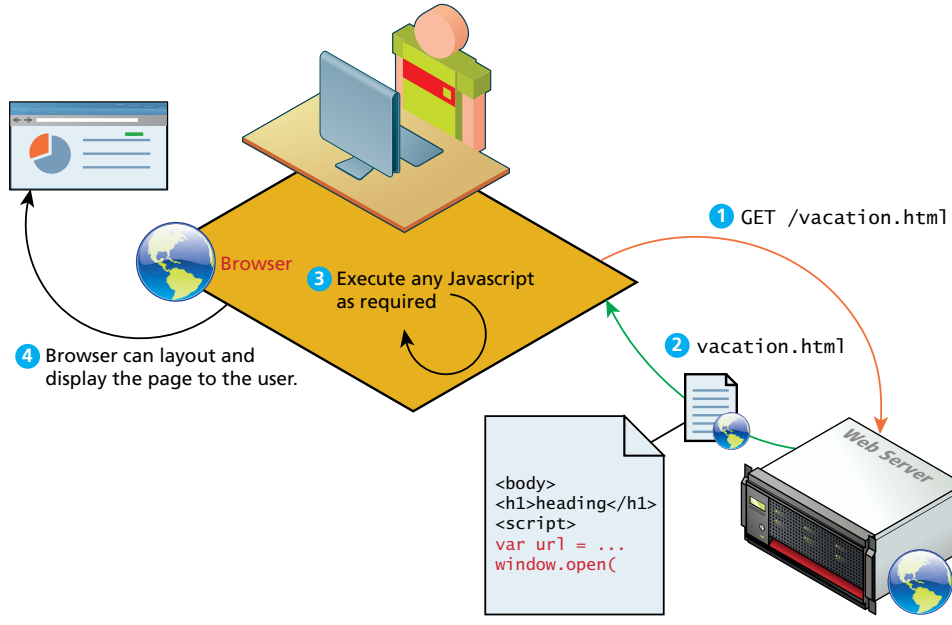


FIGURE 6.1 Downloading and executing a client-side JavaScript script

- JavaScript can interact with the downloaded HTML in a way that the server cannot, creating a user experience more like desktop software than simple HTML ever could.

The disadvantages of client-side scripting are mostly related to how programmers use JavaScript in their applications. Some of these include:

- There is no guarantee that the client has JavaScript enabled, meaning any required functionality must be housed on the server, despite the possibility that it could be offloaded.
- The idiosyncrasies between various browsers and operating systems make it difficult to test for all potential client configurations. What works in one browser, may generate an error in another.
- JavaScript-heavy web applications can be complicated to debug and maintain. JavaScript has often been used through inline HTML hooks that are embedded into the HTML of a web page. Although this technique has been used for years, it has the distinct disadvantage of blending HTML and JavaScript together, which decreases code readability, and increases the difficulty of web development.

Despite these limitations, the ability to enhance the visual appearance of a web application while potentially reducing the demands on the server make client-side scripting something that is a required competency for the web developer. An understanding of the concepts will help you avoid JavaScript's pitfalls and allow you to create compelling web applications.

We should mention that JavaScript is not the only type of client-side scripting. There are two other noteworthy client-side approaches to web programming.

Perhaps the most familiar of these alternatives is **Adobe Flash**, which is a vector-based drawing and animation program, a video file format, and a software platform that has its own JavaScript-like programming language called **ActionScript**. Flash is often used for animated advertisements and online games, and can also be used to construct web interfaces.

It is worth understanding how Flash works in the browser. Flash objects (not videos) are in a format called SWF (Shockwave Flash) and are included within an HTML document via the `<object>` tag. The SWF file is then downloaded by the browser and then the browser delegates control to a plug-in to execute the Flash file, as shown in Figure 6.2. A **browser plug-in** is a software add-on that extends the functionality and capabilities of the browser by allowing it to view and process different types of web content.

It should be noted that a browser plug-in is different than a **browser extension**—these also extend the functionality of a browser but are not used to process downloaded content. For instance, FireBug in the Firefox browser provides a wide range of tools that help the developer understand what's in a page; it doesn't really alter how the browser displays a page.

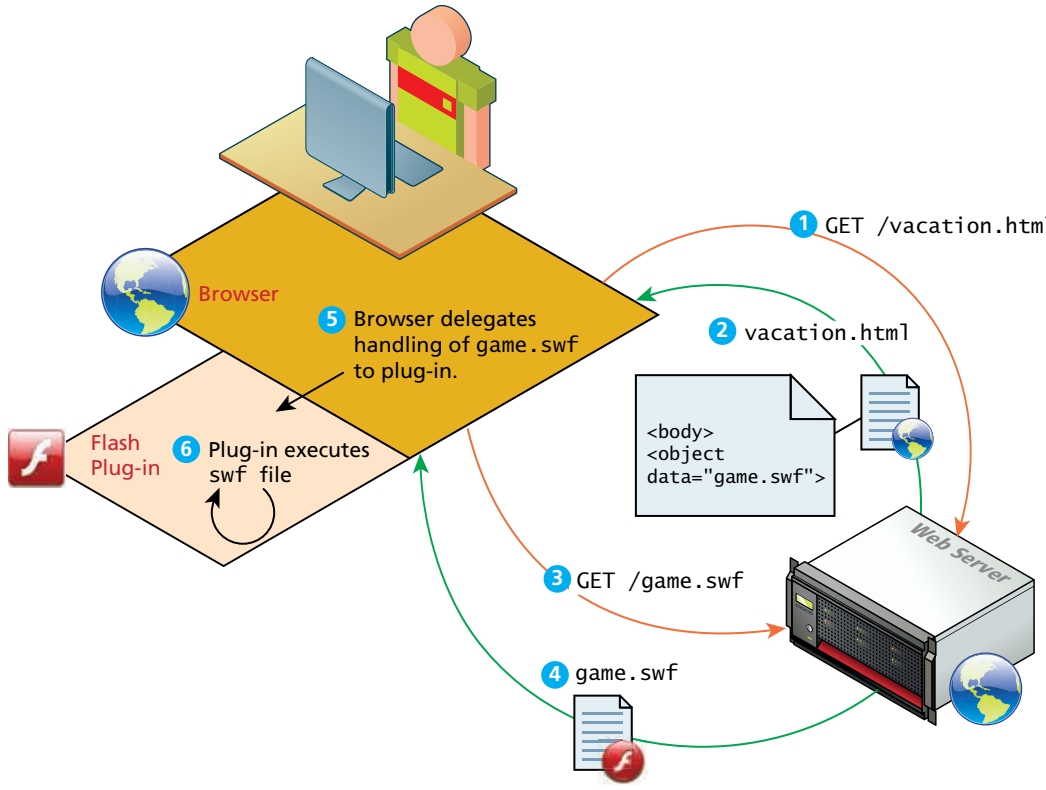


FIGURE 6.2 Adobe Flash

The second (and oldest) of these alternatives to JavaScript is Java applets. An applet is a term that refers to a small application that performs a relatively small task. Java applets are written using the Java programming language and are separate objects that are included within an HTML document via the `<applet>` tag, downloaded, and then passed on to a Java plug-in. This plug-in then passes on the execution of the applet outside the browser to the Java Runtime Environment (JRE) that is installed on the client's machine. Figure 6.3 illustrates how Java applets work in the web environment.

Both Flash plug-ins and Java applets are losing support by major players for a number of reasons. First, Java applets require the JVM be installed and up to date, which some players are not allowing for security reasons (Apple's iOS powering iPhones and iPads supports neither Flash nor Java applets). Second, Flash and Java applets also require frequent updates, which can annoy the user and present security risks. With the universal adoption of JavaScript and HTML5, JavaScript remains the most dynamic and important client-side scripting language for the modern web developer.

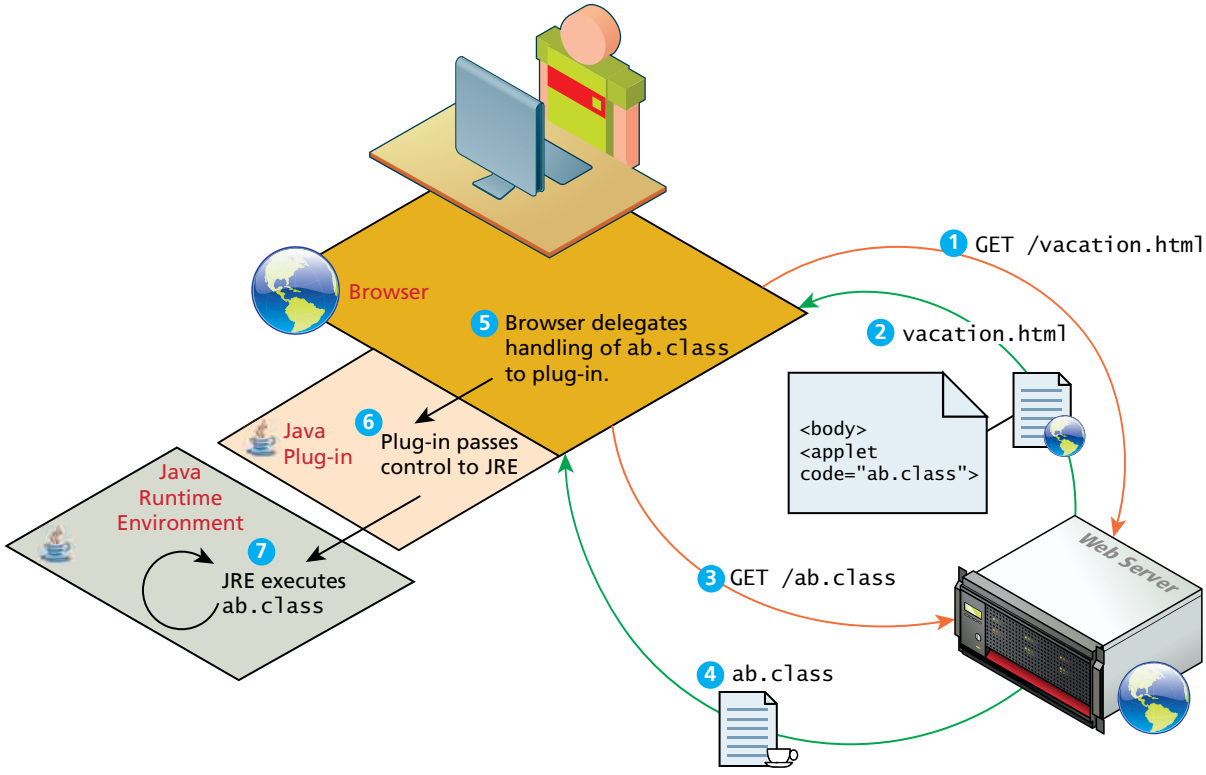


FIGURE 6.3 Java applets

6.1.2 JavaScript's History and Uses

JavaScript was introduced by Netscape in their Navigator browser back in 1996. It originally was called LiveScript, but was renamed partly because one of its original purposes was to provide a measure of control within the browser over Java applets. JavaScript is in fact an implementation of a standardized scripting language called ECMAScript.

Internet Explorer (IE) at first did not support JavaScript, but instead had its own browser-based scripting language (VBScript). While IE now does support JavaScript, Microsoft sometimes refers to it as JScript, primarily for trademark reasons (Oracle currently owns the trademark for JavaScript). The current version for JavaScript at the time of writing is 1.8.5.

One of this book's authors first started teaching web development in 1998. At that time, JavaScript was only slightly useful, and quite often, very annoying to many users. At that time, JavaScript had only a few common uses: graphic roll-overs (that is, swapping one image for another when the user hovered the mouse over an

image), pop-up alert messages, scrolling text in the status bar, opening new browser windows, and prevalidating user data in online forms.

It wasn't until the middle of the 2000s with the emergence of so-called AJAX sites that JavaScript became a much more important part of web development. **AJAX** is both an acronym as well as a general term. As an acronym it means Asynchronous JavaScript and XML, which was accurate for some time; but since XML is no longer always the data format for data transport in AJAX sites, the acronym meaning is becoming less and less accurate. As a general term, AJAX refers to a style of website development that makes use of JavaScript to create more responsive user experiences.

The most important way that this responsiveness is created is via asynchronous data requests via JavaScript and the **XMLHttpRequest** object. This addition to JavaScript was introduced by Microsoft as an ActiveX control (the IE version of plug-ins) in 1999, but it wasn't until sophisticated websites by Google (such as Gmail and Maps) and Flickr demonstrated what was possible using these techniques that the term AJAX became popular.

The most important feature of AJAX sites is the asynchronous data requests. You will eventually learn how to program these asynchronous data requests in Chapter 15. For now, however, we should say a few words about how asynchronous requests are different from the normal HTTP request-response loop.

You might want to remind yourself about how the "normal" HTTP request-response loop looks. Figure 6.4 illustrates the processing flow for a page that requires updates based on user input using the normal synchronous non-AJAX page request-response loop.

As you can see in Figure 6.4, such interaction requires multiple requests to the server, which not only slows the user experience, it puts the server under extra load, especially if, as the case in Figure 6.4, each request is invoking a server-side script.

With ever-increasing access to processing power and bandwidth, sometimes it can be really hard to tell just how much impact these requests to the server have, but it's important to remember that more trips to the server do add up, and on a large scale this can result in performance issues.

But as can be seen in Figure 6.5, when these multiple requests are being made across the Internet to a busy server, then the time costs of the normal HTTP request-response loop will be more visually noticeable to the user.

AJAX provides web authors with a way to avoid the visual and temporal deficiencies of normal HTTP interactions. With AJAX web pages, it is possible to update sections of a page by making special requests of the server in the background, creating the illusion of continuity. Figure 6.6 illustrates how the interaction shown in Figure 6.4 would differ in an AJAX-enhanced web page.

This type of AJAX development can be difficult but thankfully, the other key development in the history of JavaScript has made AJAX programming significantly

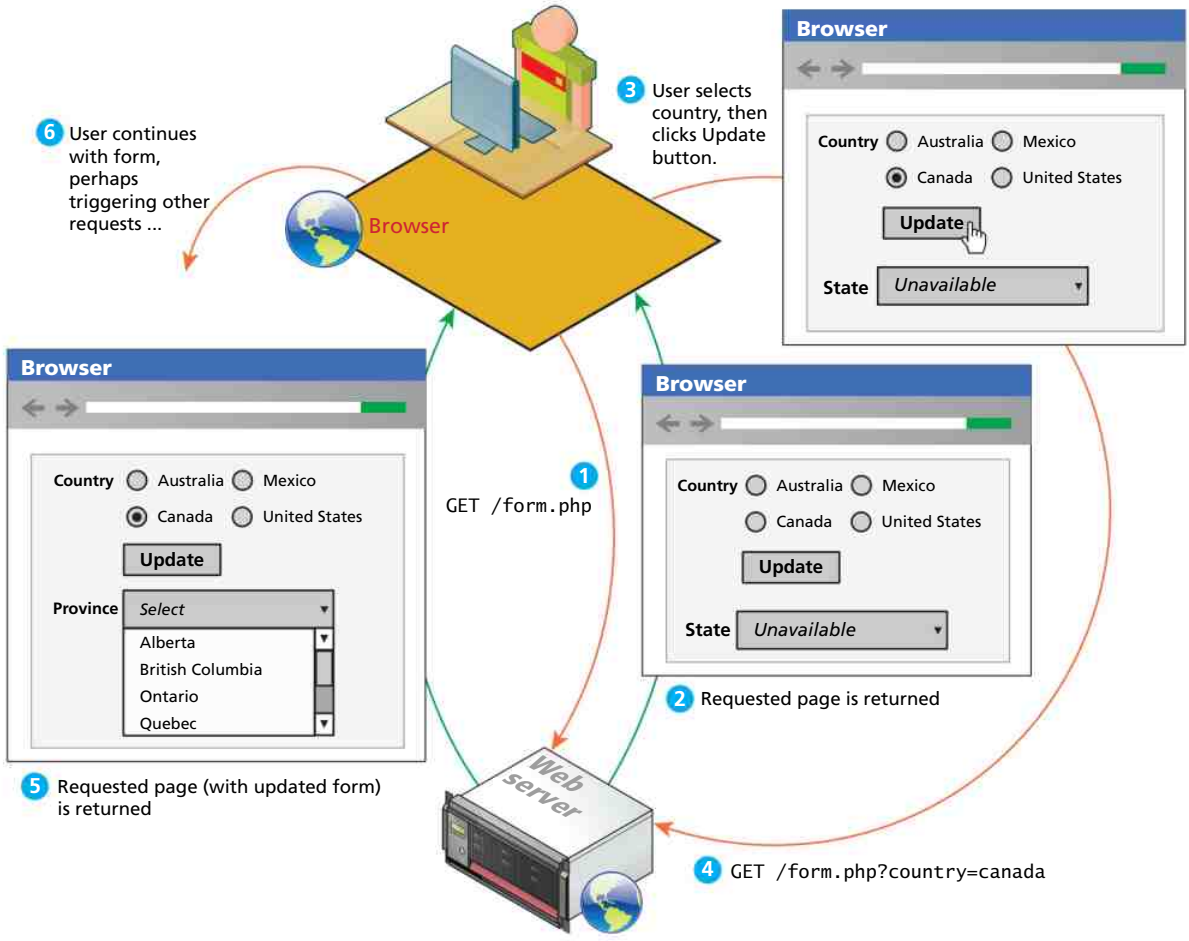


FIGURE 6.4 Normal HTTP request-response loop

less tricky. This development has been the creation of JavaScript frameworks, such as jQuery, Prototype, ASP.NET AJAX, and MooTools. These JavaScript frameworks reduce the amount of JavaScript code required to perform typical AJAX tasks. Some of these extend the JavaScript language; others provide functions and objects to simplify the creation of complex user interfaces. jQuery, in particular, has an extremely large user base, used on over half of the top 100,000 websites. Figure 6.7 illustrates some sample jQuery plug-ins, which are a way for developers to extend the functionality of jQuery. There are thousands of jQuery plug-ins available, which handle everything from additional user interface functionality to data handling.

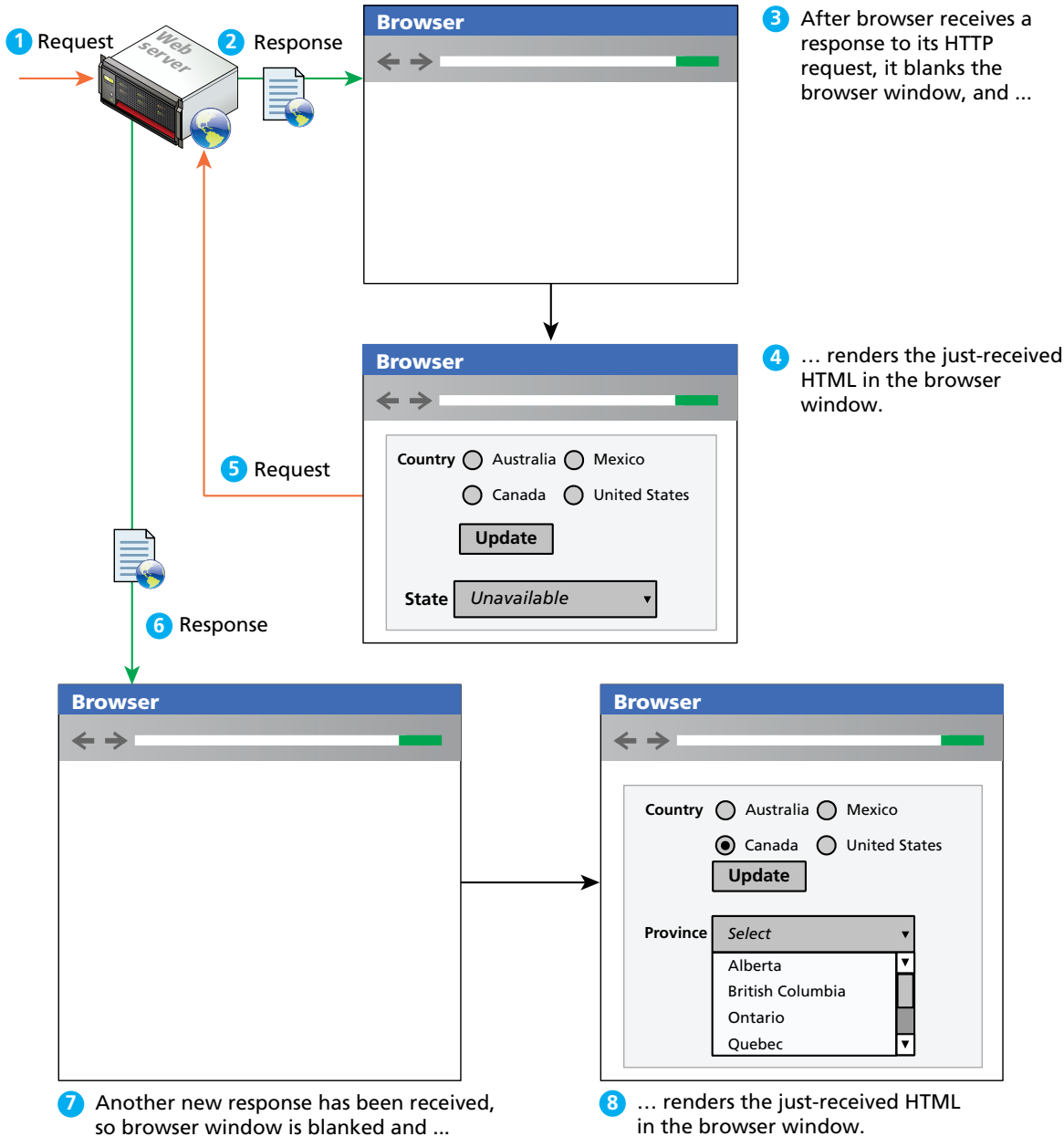


FIGURE 6.5 Normal HTTP request-response loop, take two

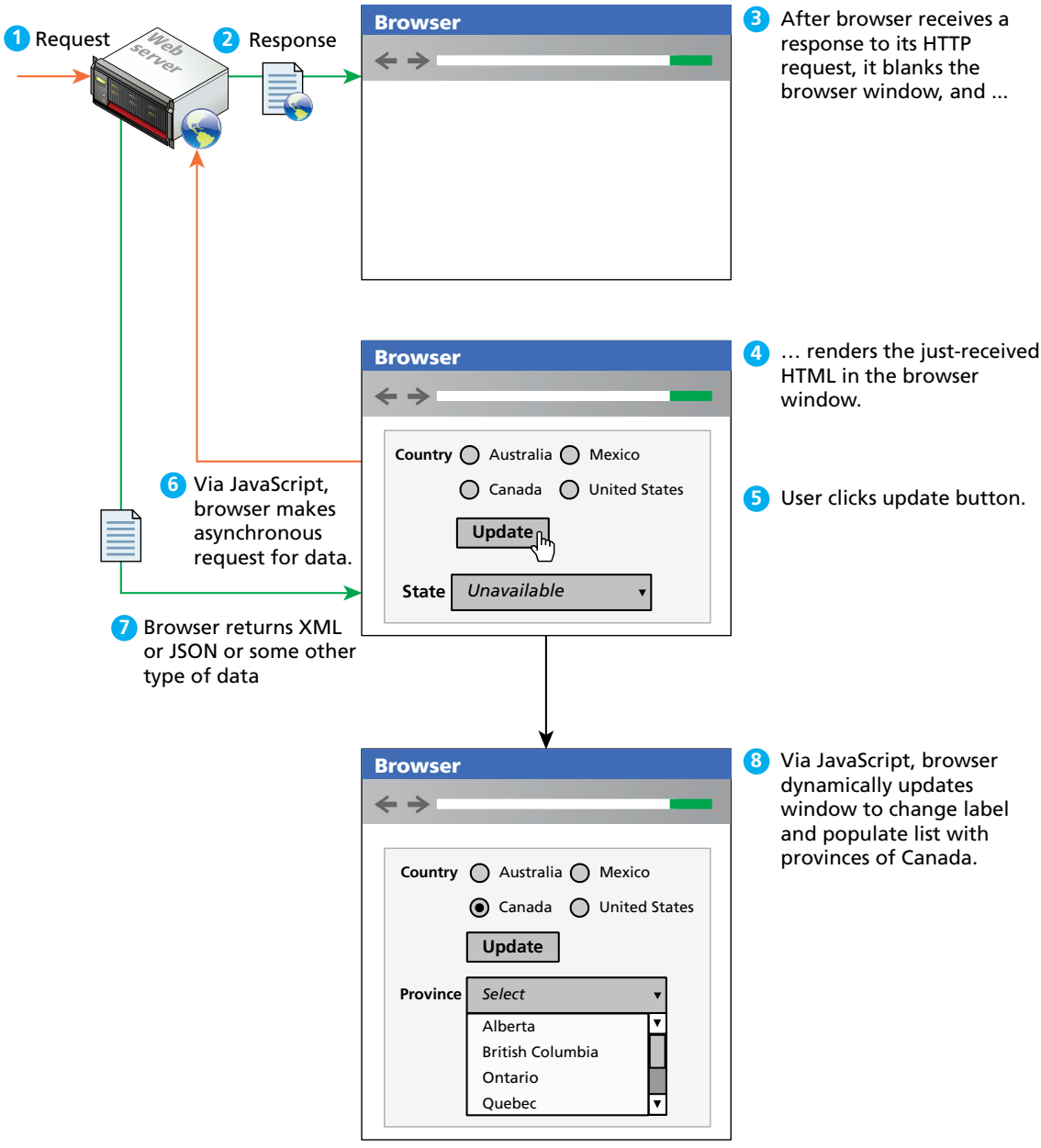


FIGURE 6.6 Asynchronous data requests

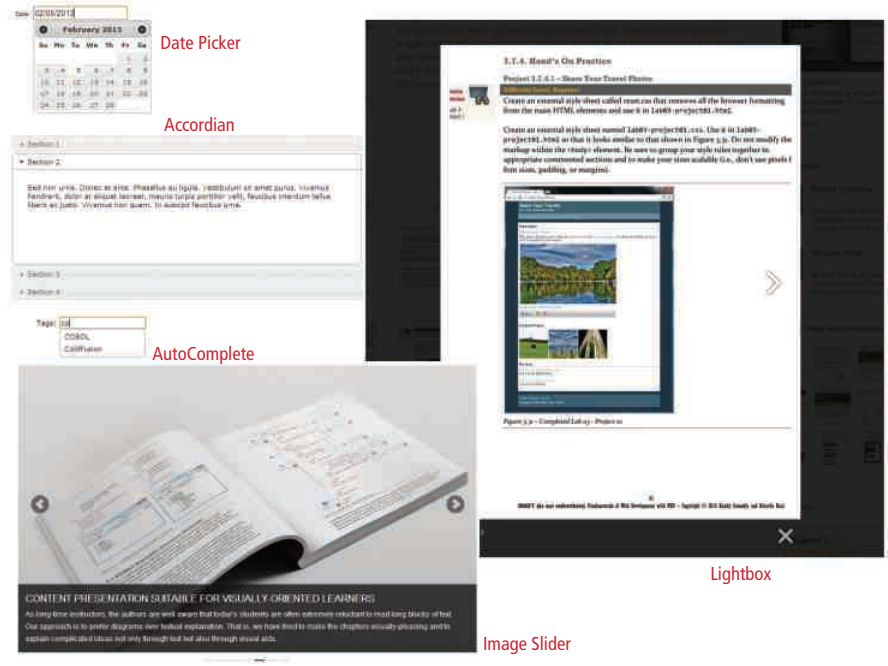


FIGURE 6.7 Example jQuery plug-ins

More recently, sophisticated MVC JavaScript frameworks such as AngularJS, Backbone, and Knockout have gained a lot of interest from developers wanting to move more data processing and handling from server-side scripts to HTML pages using a software engineering best practice, namely the separation of the model (data representation) from the view (presentation of data) design pattern. You will learn more about jQuery and this pattern in Chapter 15.

6.2 JavaScript Design Principles

As mentioned earlier, JavaScript does have a bad reputation for being a difficult language to use. Although frameworks and developer tools can help, there is some truth to this reputation.

It should be said, however, that this reputation is based not so much on the language itself but in how developers have tended to use it. JavaScript has often been used through inline HTML hooks—that is, embedded into the HTML of a web page. Although this technique has been used for years, it has the distinct disadvantage of blending HTML and JavaScript together, which decreases code readability, and increases the difficulty of web development.

This chapter briefly covers this original method before transitioning to a modern, software design–focused approach. Before getting to this current best practice, however, we should articulate these JavaScript design principles. These principles increase the quality and reusability of the code while making it easier to understand, and hence more maintainable.

6.2.1 Layers

When designing software to solve a problem, it is often helpful to abstract the solution a little bit to help build a cognitive model in your mind that you can then implement. Perhaps the most common way of articulating such a cognitive model is via the term **layer**. In object-oriented programming, a software layer is a way of conceptually grouping programming classes that have similar functionality and dependencies. Common software design layer names include:

- **Presentation layer.** Classes focused on the user interface.
- **Business layer.** Classes that model real-world entities, such as customers, products, and sales.
- **Data layer.** Classes that handle the interaction with the data sources.

You will learn more about these types of software layers in Chapter 14 on Web Application Design. We can say here simply that layers are a time-tested way to improve the quality and maintainability of software projects.

To help us conceptualize good design, we will consider JavaScript layers that exist both above and below pure HTML pages. These layers have different capabilities and responsibilities, but are always considered optional, except in some special circumstances like online games.

Although each layer can perform many tasks, it is helpful to visualize and understand the types of conceptual layers that are common. Figure 6.8 illustrates the idea of JavaScript layers.

Presentation Layer

This type of programming focuses on the display of information. JavaScript can alter the HTML of a page, which results in a change, visible to the user. These presentation layer applications include common things like creating, hiding, and showing divs, using tabs to show multiple views, or having arrows to page through result sets. This layer is most closely related to the user experience and the most visible to the end user.

Validation Layer

JavaScript can be also used to validate logical aspects of the user's experience. This could include, for example, validating a form to make sure the email entered is valid before sending it along. It is often used in conjunction with the presentation layer

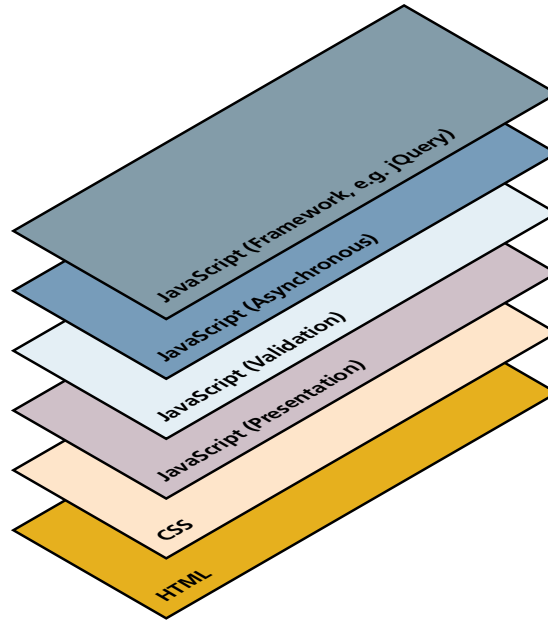


FIGURE 6.8 JavaScript layers

to create a coherent user experience, where a message to the presentation layer highlights bad fields. Both layers exist on the client machine, although the intention is to prevalidate forms before making transmissions back to the server.

Asynchronous Layers

Normally, JavaScript operates in a synchronous manner where a request sent to the server requires a response before the next lines of code can be executed. During the wait between request and response the browser sits in a loading state and only updates upon receiving the response. In contrast, an asynchronous layer can route requests to the server in the background. In this model, as certain events are triggered, the JavaScript sends the HTTP requests to the server, but while waiting for the response, the rest of the application functions normally, and the browser isn't in a loading state. When the response arrives JavaScript will (perhaps) update a portion of the page. Asynchronous layers are considered advanced versions of the presentation and validation layers above.

Typically developers work on a single file or application, weaving aspects of logic and presentation together. Although this is a common practice, separating the presentation and logic in your code is a powerful technique worth keeping in mind as you code. Having separate presentation and logic functions/classes will help you achieve more reusable code, which also happens to be easier to understand and maintain as illustrated in Figure 6.20.

HANDS-ON EXERCISES

LAB 6 EXERCISE
Enabling/Disabling JavaScript

6.2.2 Users without JavaScript

Too often website designers believe (erroneously) that users without JavaScript are somehow relics of a forgotten age, using decades-old computers in a bomb shelter somewhere philosophically opposed to updating their OS and browsers and therefore not worth worrying about. Nothing could be more of a straw man argument. Users have a myriad of reasons for not using JavaScript, and that includes some of the most important clients, like search engines. A client may not have JavaScript because they are a web crawler, have a browser plug-in, are using a text browser, or are visually impaired.

- **Web crawler.** A web crawler is a client running on behalf of a search engine to download your site, so that it can eventually be featured in their search results. These automated software agents do not interpret JavaScript, since it is costly, and the crawler cannot see the enhanced look anyway.
- **Browser plug-in.** A browser plug-in is a piece of software that works within the browser, that might interfere with JavaScript. There are many uses of JavaScript that are not desirable to the end user. Many malicious sites use JavaScript to compromise a user's computer, and many ad networks deploy advertisements using JavaScript. This motivates some users to install plug-ins that stop JavaScript execution. An ad-blocking plug-in, for example, may filter JavaScript scripts that include the word *ad*, so a script named advanced.js would be blocked inadvertently.
- **Text-based client.** Some clients are using a text-based browser. Text-based browsers are widely deployed on web servers, which are often accessed using a command-line interface. A website administrator might want to see what an HTTP `GET` request to another server is returning for testing or support purposes. Such software includes lynx as shown in Figure 6.9.
- **Visually disabled client.** A visually disabled client will use special web browsing software to read the contents of a web page out loud to them. These specialized browsers do not interpret JavaScript, and some JavaScript on sites is not accessible to these users. Designing for these users requires some extra considerations, with lack of JavaScript being only one of them. Open-source browsers like WebIE would display the same site as shown in Figure 6.10.

The <NoScript> Tag

Now that we know there are many sets of users that may have JavaScript disabled, we may want to make use of a simple mechanism to show them special HTML content that will not be seen by those with JavaScript. That mechanism is the HTML tag `<noscript>`. Any text between the opening and closing tags will only be displayed to users without the ability to load JavaScript. It is often used to prompt users to enable JavaScript, but can also be used to show additional text to search engines.

HANDS-ON EXERCISES

LAB 6 EXERCISE
Using NoScript

FIGURE 6.9 Surfing the web with Lynx

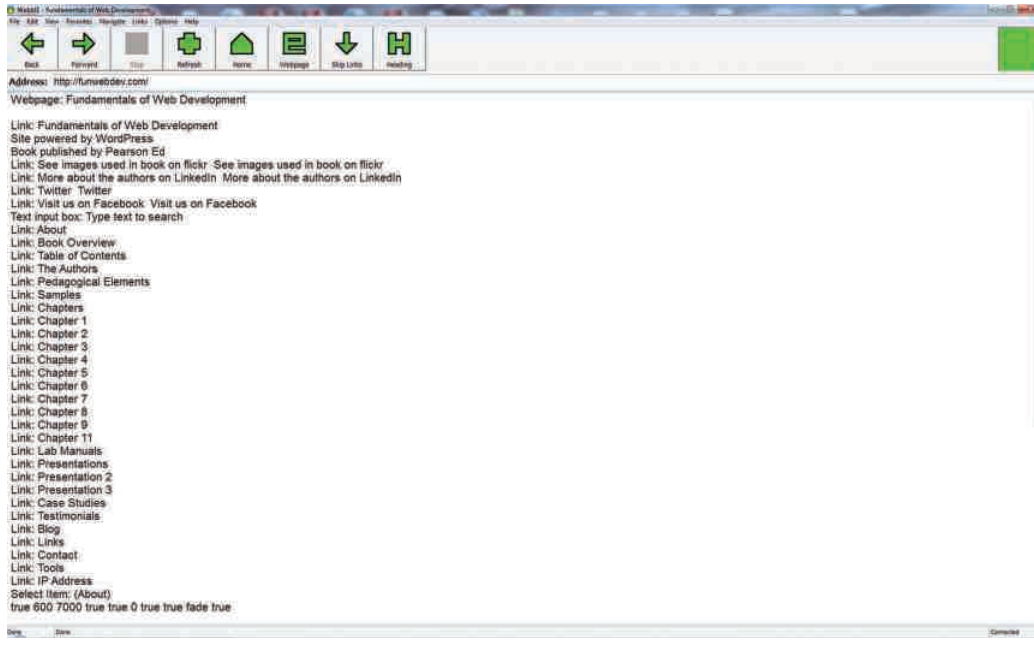


FIGURE 6.10 Screenshot of WebIE, browser for the visually impaired

Increasingly, websites that focus on JavaScript or Flash only risk missing out on an important element to help get them noticed: search engine optimization (SEO). Moreover, older or mobile browsers may not have a complete JavaScript implementation. Requiring JavaScript (or Flash) for the basic operation of your site will cause problems eventually and should be avoided. In this spirit, we should create websites with all the basic functionality enabled using regular HTML. For those (majority) of users with JavaScript enabled we can then enhance the basic layout using JavaScript to: embellish the look of certain elements, animate certain user interactions, prevalidate forms, and generally replace static HTML elements with more visually and logically enhanced elements from JavaScript. Some examples of this principle would be to replace submit buttons with animated images, or adding dropdown menus to an otherwise static menu structure.

This approach of adding functional replacements for those without JavaScript is also referred to as **fail-safe design**, which is a phrase with a meaning beyond web development. It means that when a plan (such as displaying a fancy JavaScript pop-up calendar widget) fails (because for instance JavaScript is not enabled), then the system's design will still work (for instance, by allowing the user to simply type in a date inside a text box).

NOTE

The Google search crawlers have started to interpret some asynchronous JavaScript portions of websites, but only by request, and only related to certain asynchronous aspects of JavaScript.[1] Nonetheless, fail-safe design is still the best way to design your site, and ensure it works for everyone, including search crawlers.

SECURITY

While the above examples describe benign users with special needs, avoiding JavaScript is also a technique used by malicious and curious clients, intent on circumventing any JavaScript locks you have in place. You must remember that at the end of the day only HTTP requests are sent to the server, and nothing you expect to be done by JavaScript is guaranteed, since you do not have control over the client's computer.

6.2.3 Graceful Degradation and Progressive Enhancement

The principle of fail-safe design can still apply even to browsers that have enabled JavaScript. Over the years, browser support for different JavaScript objects has varied. Something that works in the current version of Chrome might not work in IE version 8; something that works in a desktop browser might not work in a mobile browser. In such cases, what strategy should we take as web application developers?

The principle of **graceful degradation** is one possible strategy. With this strategy you develop your site for the abilities of current browsers. For those users who are not using current browsers, you might provide an alternate site or pages for those using older browsers that lack the JavaScript (or CSS or HTML5) used on the main site. The idea here is that the site is "degraded" (i.e., loses capability) "gracefully" (i.e., without pop-up JavaScript error codes or without condescending messages telling users to upgrade their browsers). Figure 6.11 illustrates the idea of graceful degradation.

The alternate strategy is **progressive enhancement**, which takes the opposite approach to the problem. In this case, the developer creates the site using CSS, JavaScript, and HTML features that are supported by all browsers of a certain age or newer. (Eventually, one does have to stop supporting ancient browsers; many developers have, for instance, stopped supporting IE 6.) To that baseline site, the developers can now "progressively" (i.e., for each browser) "enhance" (i.e., add functionality) to their site based on the capabilities of the users' browsers. For instance, users using the current version of Opera and Chrome might see the fancy HTML5 color input form elements (since both support it at present), users using current versions of other browsers might see a jQuery plug-in that has similar functionality, while users of IE 7 might just see a simple text box. Figure 6.12 illustrates the idea of progressive enhancement.

6.3 Where Does JavaScript Go?

JavaScript can be linked to an HTML page in a number of ways. Just as CSS styles can be **inline**, **embedded**, or **external**, JavaScript can be included in a number of ways. Just as with CSS these can be combined, but external is the preferred method for cleanliness and ease of maintenance.

Running JavaScript scripts in your browser requires downloading the JavaScript code to the browser and then running it. Pages with lots of scripts could potentially run slowly, resulting in a degraded experience while users wait for the page to load. Different browsers manage the downloading and loading of scripts in different ways, which are important things to realize when you decide how to link your scripts.

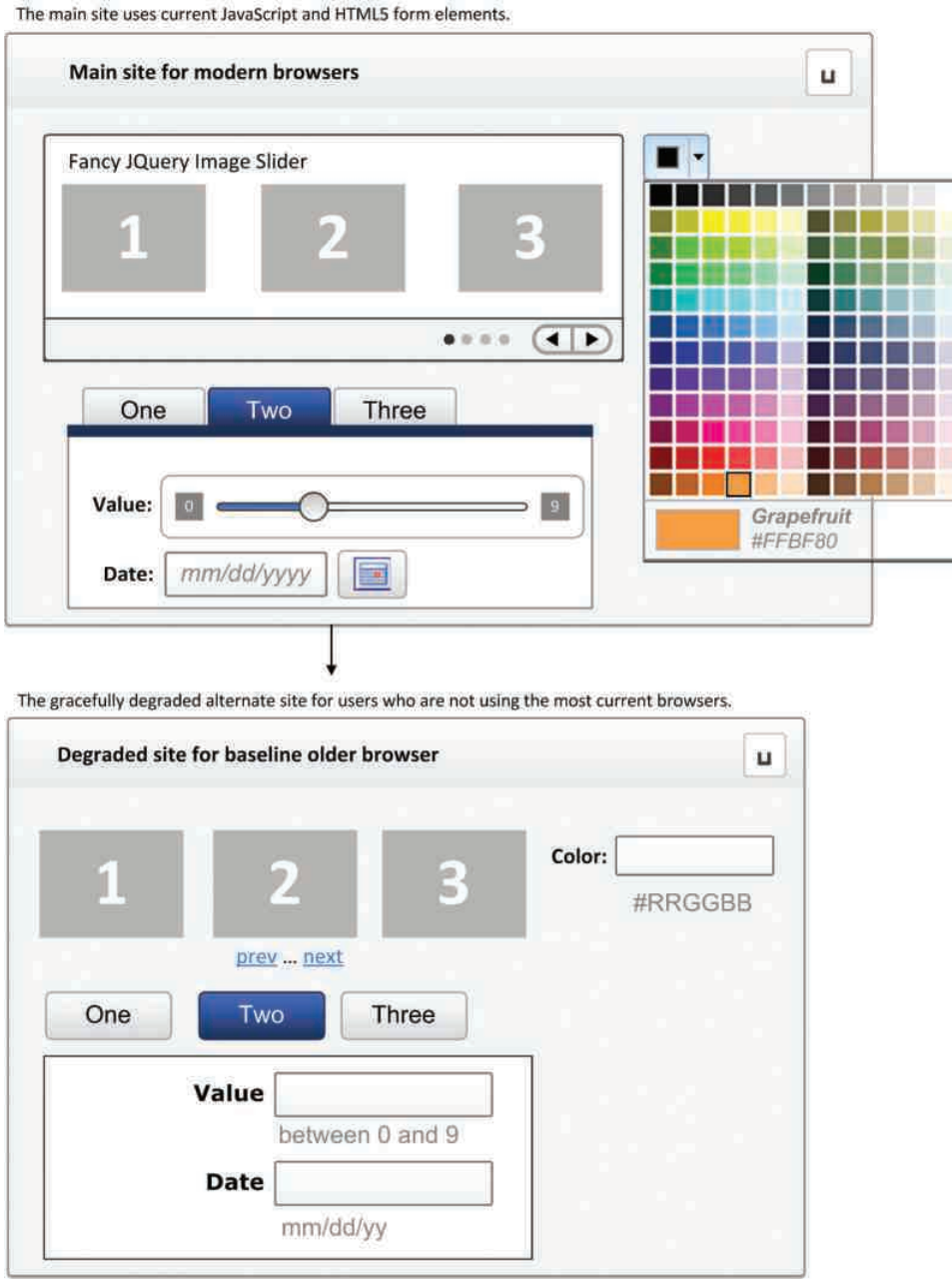


FIGURE 6.11 Example of graceful degradation

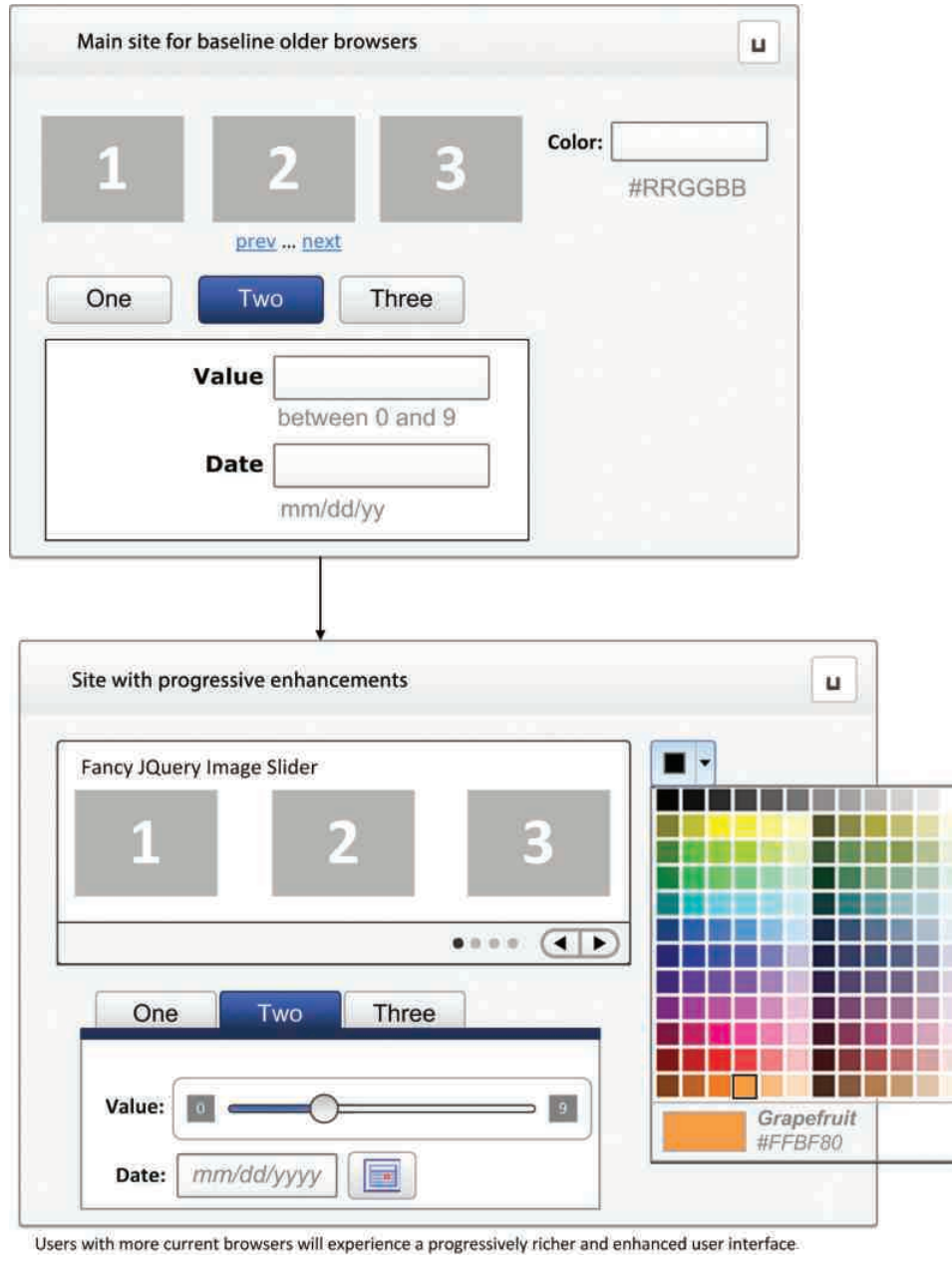


FIGURE 6.12 Site with Progressive Enhancements

6.3.1 Inline JavaScript

Inline JavaScript refers to the practice of including JavaScript code directly within certain HTML attributes, such as that shown in Listing 6.1.

```
<a href="JavaScript:OpenWindow();"more info</a>
<input type="button" onclick="alert('Are you sure?');" />
```

LISTING 6.1 Inline JavaScript example

You may recall that in Chapter 3 on CSS you were warned that inline CSS is in general a bad practice and should be avoided. The same is true with JavaScript. In fact, inline JavaScript is much worse than inline CSS. Inline JavaScript is a real maintenance nightmare, requiring maintainers to scan through almost every line of HTML looking for your inline JavaScript.

6.3.2 Embedded JavaScript

Embedded JavaScript refers to the practice of placing JavaScript code within a <script> element as shown in Listing 6.2. Like its equivalent in CSS, embedded JavaScript is okay for quick testing and for learning scenarios, but is frowned upon for normal real-world pages. Like with inline JavaScript, embedded scripts can be difficult to maintain.

HANDS-ON EXERCISES

LAB 6 EXERCISE
Embedded JavaScript

```
<script type="text/javascript">
/* A JavaScript Comment */
alert ("Hello World!");
</script>
```

LISTING 6.2 Embedded JavaScript example

PRO TIP

Some high-traffic sites prefer using embedded styles and JavaScript scripts to reduce the number of GET requests they must respond to from each client. Sites like the main page for Google Search embed styles and JavaScript in the HTML to speed up performance by reducing the need for extra HTTP requests. In these cases performance improves because the size of the embedded styles and JavaScript is quite modest.

For most sites and pages, external JavaScript (and CSS) will in fact provide the best performance because for frequently visited sites, the external files will more than likely be cached locally by the user's browser if those external files are referenced by multiple pages in the site.

Thus, if users for a site tend to view multiple pages on that site with each visit, **and** many of the site's pages re-use the same scripts and style sheets, then the site will likely benefit from cached external files.

6.3.3 External JavaScript

HANDS-ON EXERCISES

LAB 6 EXERCISE
External JavaScript

Since writing code is a different competency than designing HTML and CSS, it is often advantageous to separate the two into different files. JavaScript supports this separation by allowing links to an external file that contains the JavaScript, as shown in Listing 6.3.

This is the recommended way of including JavaScript scripts in your HTML pages.

By convention, JavaScript external files have the extension `.js`. Modern websites often have links to several, maybe even dozens, of external JavaScript files (also called libraries). These external files typically contain function definitions, data definitions, and other blocks of JavaScript code.

```
<head>
  <script type="text/JavaScript" src="greeting.js">
  </script>
</head>
```

LISTING 6.3 External JavaScript example

In Listing 6.3, the link to the external JavaScript file is placed within the `<head>` element, just as was the case with links to external CSS files. While this is convention, it is in fact possible to place these links anywhere within the `<body>` element. We certainly recommend placing them either in the `<head>` element or the very bottom of the `<body>` element.

The argument for placing external scripts at the bottom of the `<body>` has to do with performance. A JavaScript file has to be loaded completely before the browser can begin any other downloads (including images). For sites with multiple external JavaScript files, this can cause a noticeable delay in initial page rendering. Similarly, if your page is loading a third-party JavaScript library from an external site, and that site becomes unavailable or especially slow, then your pages will be rendered especially slow.

Nonetheless, it is not uncommon for JavaScript to insert markup into the page before loading, and in such a case the JavaScript must be within the `<head>`. In this book we will place our links to external JavaScript files within the `<head>` in the name of simplicity, but in a real-world scenario, we would likely try moving them to the end of the document for the above-mentioned performance reasons.

6.3.4 Advanced Inclusion of JavaScript

Imagine for a moment a user with a browser that has JavaScript disabled. When downloading a page, if the JavaScript scripts are embedded in the page, they must download those scripts in their entirety, despite being unable to process them. A subtler version of that scenario is a user with JavaScript enabled, who has a slow

computer, or Internet connection. Making them wait for every script to download may have a net negative impact on the user experience if the page must download and interpret all JavaScript before proceeding with rendering the page. It is possible to include JavaScript in such a way that minimizes these problems. (Due to their advanced nature the details are described in the lab manual.)

One approach is to load one or more scripts (or style sheets) into an `<iframe>` on the same domain. In such an advanced scenario, the main JavaScript code in the page can utilize functions in the `<iframe>` using the DOM hierarchy to reference the frame.

Another approach is to load a JavaScript file from within another JavaScript file. In such a scenario a simple JavaScript script is downloaded, with the only objective of downloading a larger script later, upon demand, or perhaps after the page has finished loading. We will see how social networks use this technique extensively in the last chapter.

6.4 Syntax

Since it's a lightweight scripting language, JavaScript has some features (such as dynamic typing) that are especially helpful to the novice programmer. However, a novice programmer faces challenges when he or she tries to use JavaScript in the same way as a full object-oriented language such as Java, as JavaScript's object features (such as prototypes and inline functions) are quite unlike those of more familiar languages.

We will briefly cover the fundamental syntax for the most common programming constructs including variables, assignment, conditionals, loops, and arrays before moving on to advanced topics such as events and classes.

JavaScript's reputation for being quirky not only stems from its strange way of implementing object-oriented principles, but also from some odd syntactic *gotchas* that every JavaScript developer will eventually encounter, some of which include:

- Everything is type sensitive, including function, class, and variable names.
- The scope of variables in blocks is not supported. This means variables declared inside a loop may be accessible outside of the loop, counter to what one would expect.
- There is a `===` operator, which tests not only for equality but type equivalence.
- Null and undefined are two distinctly different states for a variable.
- Semicolons are not required, but are permitted (and encouraged).
- There is no integer type, only number, which means floating-point rounding errors are prevalent even with values intended to be integers.

```
var x;      <-- a variable x is defined
var y = 0;  <-- y is defined and initialized to 0
y = 4;      <-- y is assigned the value of 4
```

FIGURE 6.13 Variable declaration and assignment

6.4.1 Variables

Variables in JavaScript are dynamically typed, meaning a variable can be an integer, and then later a string, then later an object, if so desired. This simplifies variable declarations, so that we do not require the familiar type fields like *int*, *char*, and *String*. Instead, to declare a variable `x`, we use the `var` keyword, the name, and a semicolon as shown in Figure 6.13. If we specify no value, then (being typeless) the default value is `undefined`.

Assignment can happen at declaration-time by appending the value to the declaration, or at run time with a simple right-to-left assignment as illustrated in Figure 6.13. This syntax should be familiar to those who have programmed in languages like C and Java.

In addition, the conditional assignment operator, shown in Figure 6.14, can also be used to assign based on condition, although its use is sometimes discouraged.

NOTE

There are two styles of comment in JavaScript, the end-of-line comment, which starts with two slashes //, and the block comment, which begins with /* and ends with */.

6.4.2 Comparison Operators

The core of any programming language is the ability to distill things down to Boolean statements where something is either true or false. JavaScript is no exception and comes equipped with a number of operators to compare two values, listed in Table 6.1.

```
/* x conditional assignment */
x = (y==4) ? "y is 4" : "y is not 4";
    Condition  Value      Value
               if true    if false
```

FIGURE 6.14 The conditional assignment operator

Operator	Description	Matches (x=9)
==	Equals	(x==9) is true (x=="9") is true
===	Exactly equals, including type	(x==="9") is false (x===9) is true
< , >	Less than, greater than	(x<5) is false
<= , >=	Less than or equal, greater than or equal	(x<=9) is true
!=	Not equal	(4!=x) is true
!==	Not equal in either value or type	(x!=="9") is true (x!==9) is false

TABLE 6.1 Comparison Operators

These operators will be familiar to those of you who have programmed in PHP, C, or Java. These comparison operators are used in conditional, loop, and assignment statements.

6.4.3 Logical Operators

Comparison operators are useful, but without being able to combine several together, their usefulness would be severely limited. Therefore, like most languages JavaScript includes Boolean operators, which allow us to build complicated expressions. The Boolean operators `and`, `or`, and `not` and their truth tables are listed in Table 6.2. Syntactically they are represented with `&&` (and), `||` (or), and `!` (not).

6.4.4 Conditionals

JavaScript's syntax is almost identical to that of PHP, Java, or C when it comes to conditional structures such as `if` and `if else` statements. In this syntax the condition to test is contained within `( )` brackets with the body contained in `{ }` blocks.

A	B	A && B
T	T	T
T	F	F
F	T	F
F	F	F

AND Truth Table

A	B	A || B
T	T	T
T	F	T
F	T	T
F	F	F

OR Truth Table

A	! A
T	F
F	T

NOT Truth Table

TABLE 6.2 AND, OR, and NOT Truth Tables

Optional `else if` statements can follow, with an `else` ending the branch. Listing 6.4 uses a conditional to set a greeting variable, depending on the hour of the day.

```
var hourOfDay;   // var to hold hour of day, set it later...
var greeting;   // var to hold the greeting message.
if (hourOfDay > 4 && hourOfDay < 12){
   // if statement with condition
   greeting =  "Good Morning";
}
else if (hourOfDay >= 12 && hourOfDay < 20){
   // optional else if
   greeting =  "Good Afternoon";
}
else{ // optional else branch
   greeting = "Good Evening";
}
```

LISTING 6.4 Conditional statement setting a variable based on the hour of the day

6.4.5 Loops

Like conditionals, loops use the `( )` and `{ }` blocks to define the condition and the body of the loop.

While Loops

The most basic loop is the `while` loop, which loops until the condition is not met. Loops normally initialize a loop control variable before the loop, use it in the condition, and modify it within the loop. One must be sure that the variables that make up the condition are updated inside the loop (or elsewhere) to avoid an infinite loop!

```
var i=0;
while(i < 10){
 //do something with i
 i++;
}
```

For Loops

A for loop combines the common components of a loop: initialization, condition, and post-loop operation into one statement. This statement begins with the `for` keyword and has the components placed between `( )` brackets, semicolon (`;`) separated as shown in Figure 6.15.

```
for (var i = 0; i < 10; i++){
  //do something with i
}
```

FIGURE 6.15 For loop

NOTE

Infinite loops can happen if we are not careful, and since the scripts are executing on the client computer, it can appear to the user that the browser is "locked" while endlessly caught in a loop processing. Some browsers will even try to terminate scripts that execute for too long a time to mitigate this unpleasantness.

6.4.6 Functions

Functions are the building block for modular code in JavaScript, and are even used to build **pseudo-classes**, which you will learn about later. They are defined by using the reserved word `function` and then the function name and (optional) parameters. Since JavaScript is dynamically typed, functions do not require a return type, nor do the parameters require type. Therefore a function to raise x to the yth power might be defined as:

```
function power(x,y){
  var pow=1;
  for (var i=0;i<y;i++){
     pow = pow*x;
  }
  return pow;
}
```

And called as

```
power(2,10);
```

With new programmers there is often confusion between defining a function and calling the function. Remember that when actually using the keyword `function`, we are defining what the function does. Later, we can use or call that function by using its given name *without* the function keyword.

Later in this chapter you will see the advanced use of functions to build classes.

Alert

The `alert()` function makes the browser show a pop-up to the user, with whatever is passed being the message displayed. The following JavaScript code displays a simple hello world message in a pop-up:

HANDS-ON EXERCISES

LAB 6 EXERCISE
Simple Script

```
alert ( "Good Morning" );
```

The pop-up may appear different to each user depending on their browser configuration. What is universal is that the pop-up obscures the underlying web page, and no actions can be done until the pop-up is dismissed.

Alerts are not used in production code, but are a useful tool for debugging and illustration purposes. Alerts are used throughout the chapter to provide example output, and in practice are often used for debugging or as placeholders for the eventual code, which might log to a file, transmit a message, or update an interface.

PRO TIP

Using alerts can get tedious fast. You have to click OK, and if you use it in a loop you may spend more time clicking OK than doing meaningful work. When using debugger tools in your browser you can normally write output to a log with:

```
Console.log("Put Messages Here");
```

And then use the debugger to access those logs. Any logging will be unseen by the user.

6.4.7 Errors Using Try and Catch

When the browser's JavaScript engine encounters an error, it will *throw* an exception. These exceptions interrupt the regular, sequential execution of the program and can stop the JavaScript engine altogether. However, you can optionally catch these errors preventing disruption of the program using the try–catch block as shown in Listing 6.5.

```
try {
  nonexistantfunction("hello");
}
catch(err) {
  alert("An exception was caught:" + err);
}
```

LISTING 6.5 Try-catch statement

Throwing Your Own Exceptions

Although `try-catch` can be used exclusively to catch built-in JavaScript errors, it can also be used by your programs, to throw your own messages. The `throw` keyword stops normal sequential execution, just like the built-in exceptions as shown in Listing 6.6.

The general consensus in software development is that `try-catch` and `throw` statements should be used for *abnormal* or *exceptional* cases in your program. They

should not be used as a normal way of controlling flow, although no formal mechanism exists to enforce that idea. We will generally avoid `try-catch` statements in our code unless illustrative of some particular point. Listing 6.6 demonstrates the throwing of a user-defined exception as a string. In reality any object can be thrown, although in practice a string usually suffices.

```
try {
   var x = -1;
   if (x<0)
      throw "smallerthan0Error";
}
catch(err){
   alert (err + "was thrown");
}
```

LISTING 6.6 Throwing a user-defined exception

It should be noted that throwing an exception disrupts the sequential execution of a program. That is, when the exception is thrown all subsequent code is not executed until the `catch` statement is reached. This reinforces why `try-catch` is for exceptional cases.

6.5 JavaScript Objects

HANDS-ON EXERCISES

LAB 6 EXERCISE
JavaScript Objects

JavaScript is not a full-fledged object-oriented programming language. It does not have classes per se, and it does not support many of the patterns you'd expect from an object-oriented language like inheritance and polymorphism in a straightforward way.

The language does, however, support objects. User-defined objects are declared in a slightly odd way to developers familiar with languages like C++ or Java, so the syntax to build pseudo-classes can be challenging. Nonetheless the advantages of encapsulating data and methods into objects outweigh the syntactic hurdle you will have to overcome.

Objects can have constructors, properties, and methods associated with them, and are used very much like objects in other object-oriented languages. There are objects that are included in the JavaScript language; you can also define your own kind of objects.

6.5.1 Constructors

Normally to create a new object we use the `new` keyword, the class name, and `( )` brackets with *n* optional parameters inside, comma delimited as follows:

```
var someObject = new ObjectName(parameter 1,param 2,..., parameter n);
```

For some classes, shortcut constructors are defined, which can be confusing if we are not aware of them. For example, a `String` object can be defined with the shortcut

```
var greeting = "Good Morning";
```

Instead of the formal definition

```
var greeting = new String("Good Morning");
```

Arrays are another class with a shortcut constructor, described later in this section.

6.5.2 Properties

Each object might have properties that can be accessed, depending on its definition. When a property exists, it can be accessed using **dot notation** where a dot between the instance name and the property references that property.

```
alert(someObject.property); //show someObject.property to the user
```

NOTE

One should recall that in object-oriented programming each object maintains its own properties so `A.name != B.name`. This allows the programmer to manage complex related data in intuitive objects, rather than the alternative of arrays or other data structures.

Methods

Objects can also have methods, which are functions associated with an instance of an object. These methods are called using the same dot notation as for properties, but instead of accessing a variable, we are calling a method.

```
someObject.doSomething();
```

Methods may produce different output depending on the object they are associated with because they can utilize the internal properties of the object.

6.5.3 Objects Included in JavaScript

A number of useful objects are included with JavaScript. These include `Array`, `Boolean`, `Date`, `Math`, `String`, and others. In addition to these, JavaScript can also

access Document Object Model (DOM) objects that correspond to the content of a page's HTML. These DOM objects let JavaScript code access and modify HTML and CSS properties of a page dynamically.

Arrays

Arrays are one of the most used data structures, and they have been included in JavaScript as well. In practice, this class is defined to behave more like a linked list in that it can be resized dynamically, but the implementation is browser specific, meaning the efficiency of insert and delete operations is unknown.

Arrays will be the first objects we will examine. Objects can be created using the `new` syntax and calling the object constructor. The following code creates a new, empty array named `greetings`:

```
var greetings = new Array();
```

To initialize the array with values, the variable declaration would look like the following:

```
var greetings = new Array("Good Morning", "Good Afternoon");
```

or, using the square bracket notation:

```
var greetings = ["Good Morning", "Good Afternoon"];
```

While you should be careful to employ consistency in your own array declarations, it's important to familiarize yourself with notation that may be used by others. Teams should agree on some standards in this area.

Accessing and Traversing an Array

To access an element in the array you use the familiar square bracket notation from Java and C-style languages, with the index you wish to access inside the brackets.

```
alert ( greetings[0] );
```

One of the most common actions on an array is to traverse through the items sequentially. The following `for` loop quickly loops through an array, accessing the ith element each time using the `Array` object's `length` property to determine the maximum valid index. It will alert "Good Morning" and "Good Afternoon" to the user.

```
for (var i = 0; i < greetings.length; i++){
     alert(greetings[i]);

}
```

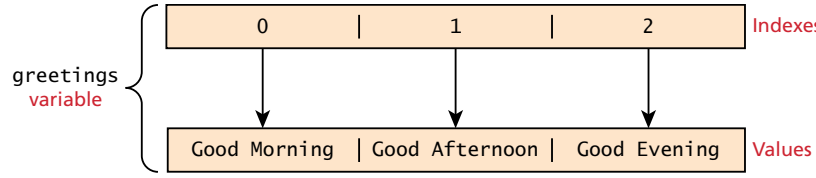


FIGURE 6.16 JavaScript array with indexes and values illustrated

Modifying an Array

To add an item to an existing array, you can use the push method.

```
greetings.push("Good Evening");
```

Figure 6.16 illustrates an array with indexes and the corresponding values.

The pop method can be used to remove an item from the back of an array. Additional methods that modify arrays include `concat()`, `slice()`, `join()`, `reverse()`, `shift()`, and `sort()`. A full accounting of all these methods is beyond the scope of a single chapter, but as you begin to use arrays you should explore them.

HANDS-ON EXERCISES

LAB 6 EXERCISE
JavaScript Arrays

Math

The Math class allows one to access common mathematic functions and common values quickly in one place. This static class contains methods such as `max()`, `min()`, `pow()`, `sqrt()`, and `exp()`, and trigonometric functions such as `sin()`, `cos()`, and `arctan()`. In addition, many mathematical constants are defined such as `PI`, `E` (Euler's number), `SQRT2`, and some others as shown in Listing 6.7.

```
Math.PI             // 3.141592657
Math.sqrt(4);       // square root of 4 is 2.
Math.random();      // random number between 0 and 1
```

LISTING 6.7 Some constants and functions in the Math object

String

The String class has already been used without us even knowing it. That is because it is core to communicating with the user. Since it is so common, shortcuts have been defined for creating and concatenating strings. While one can use the new syntax to create a `String object`, it can also be defined using quotes as follows:

```
var greet = new String("Good");    // long form constructor
var greet = "Good";                // shortcut constructor
```

A common need is to get the length of a string. This is achieved through the `length` property (just as in arrays).

```
alert (greet.length); // will display "4"
```

Another common way to use strings is to concatenate them together. Since this is so common, the + operator has been overridden to allow for concatenation in place.

```
var str = greet.concat("Morning");    // Long form concatenation
var str = greet + "Morning";          // + operator concatenation
```

Many other useful methods exist within the String class, such as accessing a single character using `charAt()`, or searching for one using `indexOf()`. Strings allow splitting a string into an array, searching and matching with `split()`, `search()`, and `match()` methods.

Date

While not critical to JavaScript, the `Date` class is yet another helpful included object you should be aware of. It allows you to quickly calculate the current date or create date objects for particular dates. To display today's date as a string, we would simply create a new object and use the `toString()` method.

```
var d = new Date();
// This outputs Today is Mon Nov 12 2012 15:40:19  GMT-0700
alert ("Today is "+ d.toString());
```

6.5.4 Window Object

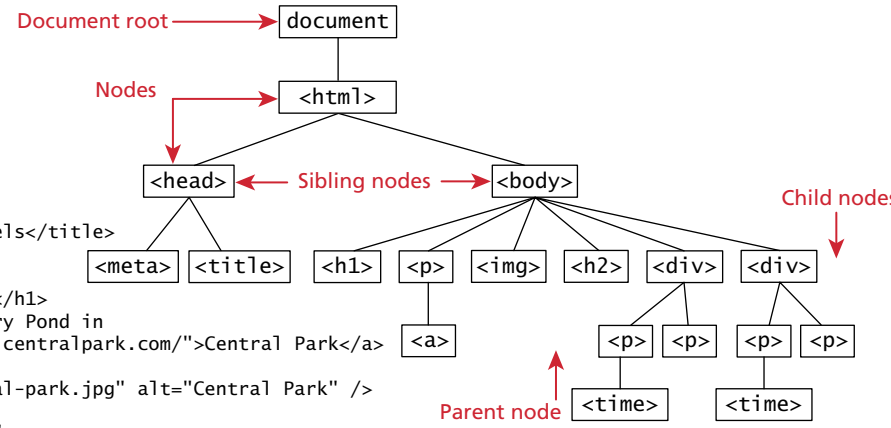
The `window` object in JavaScript corresponds to the browser itself. Through it, you can access the current page's URL, the browser's history, and what's being displayed in the status bar, as well as opening new browser windows. In fact, the `alert()` function mentioned earlier is actually a method of the `window` object.

6.6 The Document Object Model (DOM)

JavaScript is almost always used to interact with the HTML document in which it is contained. As such, there needs to be some way of programmatically accessing the elements and attributes within the HTML. This is accomplished through a programming interface (API) called the Document Object Model (DOM).

HANDS-ON EXERCISES

LAB 6 EXERCISE
Manipulate the DOM




```
<html>
<head lang="en">
   <meta charset="utf-8">
   <title>Share Your Travels</title>
</head>
<body>
   <h1>Share Your Travels</h1>
   <p>Photo of Conservatory Pond in
      <a href="http://www.centralpark.com/">Central Park</a>
   </p>
   <img src="images/central-park.jpg" alt="Central Park" />
   <h2>Reviews</h2>
   <div id="latestComment">
      <p>By Ricardo on <time>September 15, 2012</time></p>
      <p>Easy on the HDR buddy.</p>
   </div>
   <div>
      <p>By Susan on <time>October 1, 2012</time></p>
      <p>I love Central Park.</p>
   </div>
</body>
</html>
```

FIGURE 6.17 DOM tree

According to the W3C, the DOM is a:

> *Platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents.*[2]

We already know all about the DOM, but by another name. The tree structure from Chapter 2 (shown again in Figure 6.17) is formally called the **DOM Tree** with the root, or topmost object called the **Document Root**. You already know how to specify the style of documents using CSS; with JavaScript and the DOM, you now can do so dynamically as well at run time, in response to user events.

6.6.1 Nodes

In the DOM, each element within the HTML document is called a **node**. If the DOM is a tree, then each node is an individual branch. There are element nodes, text nodes, and attribute nodes, as shown in Figure 6.18.

All nodes in the DOM share a common set of properties and methods. Thus, most of the tasks that we typically perform in JavaScript involve finding a node, and then accessing or modifying it via those properties and methods. The most important of these are shown in Table 6.3.

```
<p>Photo of Conservatory Pond in
   <a href="http://www.centralpark.com/">Central Park</a>
</p>
```

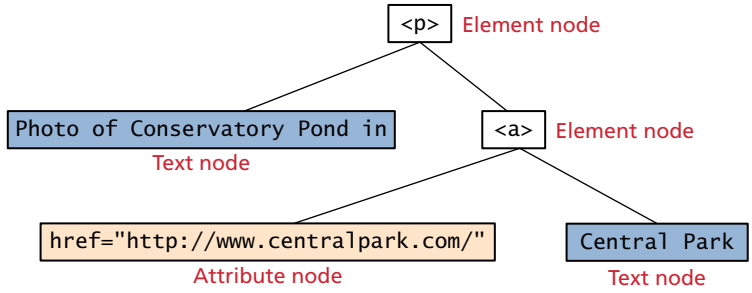


FIGURE 6.18 DOM nodes

Property	Description
attributes	Collection of node attributes
childNodes	A NodeList of child nodes for this node
firstChild	First child node of this node
lastChild	Last child of this node
nextSibling	Next sibling node for this node
nodeName	Name of the node
nodeType	Type of the node
nodeValue	Value of the node
parentNode	Parent node for this node
previousSibling	Previous sibling node for this node.

TABLE 6.3 Some Essential Node Object Properties

6.6.2 Document Object

The DOM document object is the root JavaScript object representing the entire HTML document. It contains some properties and methods that we will use extensively in our development and is globally accessible as `document`.

The attributes of this object include some information about the page including `doctype` and `inputEncoding`. Accessing the properties is done through the dot notation as illustrated on the next page.

Method	Description
`createAttribute()`	Creates an attribute node
`createElement()`	Creates an element node
`createTextNode()`	Creates a text node
`getElementById(id)`	Returns the element node whose `id` attribute matches the passed `id` parameter
`getElementsByTagName(name)`	Returns a NodeList of elements whose tag name matches the passed `name` parameter

TABLE 6.4 Some Essential Document Object Methods

```
// specify the doctype, for example html
var a = document.doctype.name;
// specify the page encoding, for example ISO-8859-1
var b = document.inputEncoding;
```

In addition to these moderately useful properties, there are some essential methods (see Table 6.4) you will use all the time. They include `getElementByTagName()` and the indispensable `getElementById()`. While the former method returns an array of DOM nodes (called a `NodeList`) matching the tag, the latter returns a single DOM element (covered below), that matches the id passed as a parameter as illustrated in Figure 6.19.

FIGURE 6.19 Relationship between HTML tags and getElementByID() and getElementsByTagName()

Selectors are generally poorly supported in pure JavaScript across the multitude of browsers and platforms available. The method `getElementById()` is universally implemented and thus used extensively. The newer `querySelector()` and `querySelectorAll()` methods allow us to query for DOM elements much the same way we specify CSS styles, but are only implemented in the newest browsers.[3] For this reason jQuery selectors (in Chapter 15) will be introduced to show more powerful selector mechanisms. Until then we will rely on `getElementByID()`.

6.6.3 Element Node Object

The type of object returned by the method `document.getElementById()` described in the previous section is an element node object. This represents an HTML element in the hierarchy, contained between the opening <> and closing </> tags for this element. As you may already have figured out, an element can itself contain more elements.

Since IDs must be unique in an HTML document, `getElementByID()` returns a single node, rather than a set of results which is the case with other selector functions. The returned `Element Node` object has the node properties shown in Table 6.3. It also has a variety of additional properties, the most important of which are shown in Table 6.5.

While these properties are available for all HTML elements, there are some HTML elements that have additional properties that can be accessed. Table 6.6 lists some common additional properties and the HTML tags that have these properties.

6.6.4 Modifying a DOM Element

In many introductory JavaScript textbooks the `document.write()` method is used to create output to the HTML page from JavaScript. While this is certainly valid, it

Property	Description
`className`	The current value for the `class` attribute of this HTML element.
`id`	The current value for the `id` of this element.
`innerHTML`	Represents all the things inside of the tags. This can be read or written to and is the primary way in which we update particular elements using JavaScript.
`style`	The `style` attribute of an element. We can read and modify this property.
`tagName`	The tag name for the element.

TABLE 6.5 Some Essential Element Node Properties

Property	Description	Tags
href	The href attribute used in a tag to specify a URL to link to.	a
name	The name property is a bookmark to identify this tag. Unlike id, which is available to all tags, name is limited to certain form-related tags.	a, input, textarea, form
src	Links to an external URL that should be loaded into the page (as opposed to href, which is a link to follow when clicked)	img, input, iframe, script
value	The value is related to the value attribute of input tags. Often the value of an input field is user defined, and we use value to get that user input.	input, textarea, submit

TABLE 6.6 Some Specific HTML DOM Element Properties for Certain Tag Types

always creates JavaScript at the bottom of the existing HTML page, and in practice is good for little more than debugging. The modern JavaScript programmer will want to write to the HTML page, but in a particular location, not always at the bottom.

Using the DOM document and HTML DOM element objects, we can do exactly that using the innerHTML property as shown in Listing 6.8 (using the HTML shown in Figure 6.19).

```
var latest = document.getElementById("latestComment");
var oldMessage = latest.innerHTML;
latest.innerHTML = oldMessage + "<p>Updated this div with JS</p>";
```

LISTING 6.8 Changing the HTML using innerHTML

Now the HTML of our document has been modified to reflect that change.

```
<div id="latestComment">
   <p>By Ricardo on <time>September 15, 2012</time></p>
   <p>Easy on the HDR buddy.</p>
   <p>Updated this div with JS</p>
</div>
```

A More Verbose Technique

Although the innerHTML technique works well (and is very fast), there is a more verbose technique available to us that builds output using the DOM. This more

explicit technique has the advantage of ensuring that only valid markup is created, while the innerHTML could output badly formed HTML. DOM functions `createTextNode()`, `removeChild()`, and `appendChild()` allow us to modify an element in a more rigorous way as shown in Listing 6.9.

```
var latest = document.getElementById("latestComment");
var oldMessage = latest.innerHTML;
var newMessage = oldMessage + "<p>Updated this div with JS</p>";
latest.removeChild(latest.firstChild);
latest.appendChild(document.createTextNode(newMessage));
```

LISTING 6.9 Changing the HTML using createTextNode( ) and appendChild( )

Changing an Element's Style

We can also modify the style associated with a particular block. We can add or remove any style using the `style` or `className` property of the `Element` node, which is something that you might want to do to dynamically change the appearance of an element. Its usage is shown below to change a node's background color and add a three-pixel border.

```
var commentTag = document.getElementById("specificTag");
commentTag.style.backgroundColour = "#FFFF00";
commentTag.style.borderWidth="3px";
```

Armed with knowledge of CSS attributes you can easily change any style attribute. Note that the `style` property is itself an object, specifically a `CSSStyleDeclaration` type, which includes all the CSS attributes as properties and computes the current style from inline, external, and embedded styles. Although you can directly access CSS style elements we suggest you use classes whenever possible.

The `className` property is normally a better choice, because it allows the styles to be created outside the code, and thus be better accessible to designers. Using this model we would change the background color by having two styles defined, and changing them in JavaScript code.

```
var commentTag = document.getElementById("specificTag");
commentTag.className = "someClassName";
```

HTML5 introduces the `classList` element, which allows you to add, remove, or toggle a CSS class on an element. You could add a class with

```
label.classList.addClass("someClassName");
```

6.6.5 Additional Properties

In addition to the global properties present in all tags, there are additional methods available when dealing with certain tags. Table 6.6 lists a few common ones.

To get the password out of the following input field and alert the user

```
<input type='password' name='pw' id='pw' />
```

We would use the following JavaScript code:

```
var pass = document.getelementById("pw");
alert (pass.value);
```

It should be obvious how getting the `src` or `href` properties out of appropriate tags could also be done. We leave it as an exercise to the reader.

6.7 JavaScript Events

HANDS-ON EXERCISES

LAB 6 EXERCISE
Handling JavaScript Events

At the core of all JavaScript programming is the concept of an **event**. A JavaScript event is an action that can be detected by JavaScript. Many of them are initiated by user actions but some are generated by the browser itself. We say then that an event is *triggered* and then it can be *caught* by JavaScript functions, which then do something in response.

In the original JavaScript world, events could be specified right in the HTML markup with *hooks* to the JavaScript code (and still can).[4] This mechanism was popular throughout the 1990s and 2000s because it worked. As more powerful frameworks were developed, and website design and best practices were refined, this original mechanism was supplanted by the **listener** approach.

A visual comparison of the old and new technique is shown in Figure 6.20. Note how the old method weaves the JavaScript right inside the HTML, while the listener technique has removed JavaScript from the markup, resulting in cleaner, easier to maintain HTML code.

6.7.1 Inline Event Handler Approach

JavaScript events allow the programmer to react to user interactions. In early web development, it made sense to weave code and HTML together and to this day, inline JavaScript calls are intuitive. For example, if you wanted an alert to pop-up when clicking a `<div>` you might program:

```
<div id="example1" onclick="alert('hello')">Click for pop-up</div>
```

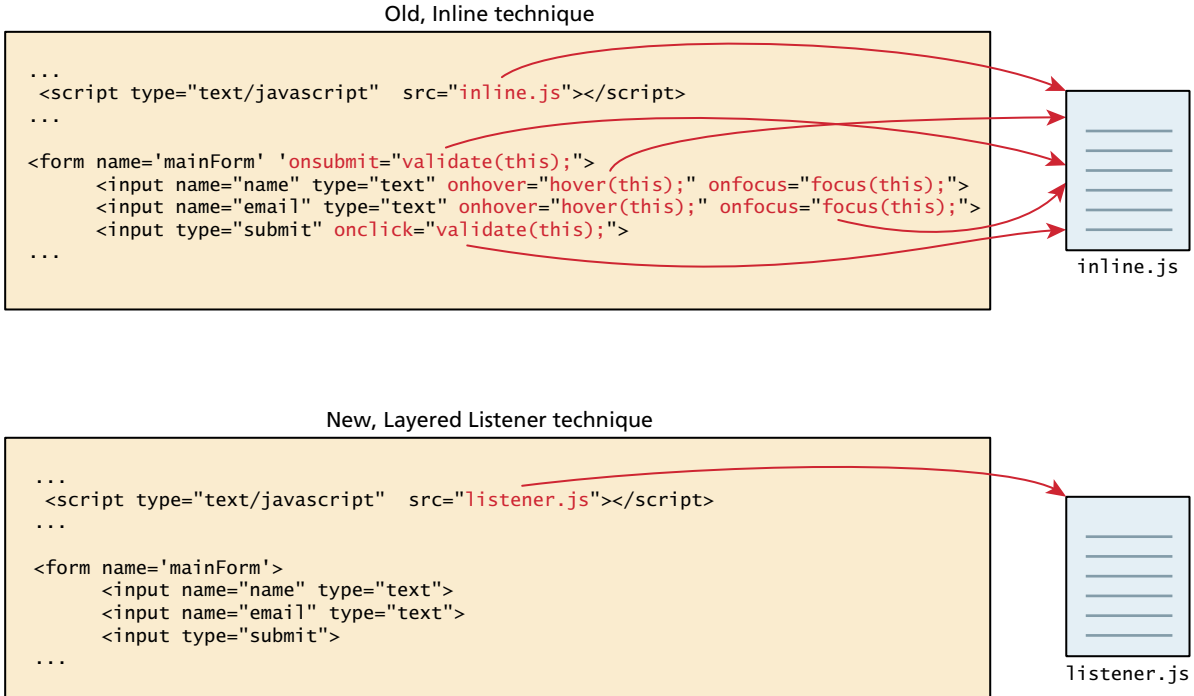


FIGURE 6.20 Inline hooks versus the Layered Listener technique

In this example the HTML attribute `onclick` is used to attach a handler to that event. When the user clicks the `<div>`, the event is triggered and the alert is executed. The problem with this type of programming is that the HTML markup and the corresponding JavaScript logic are woven together. This reduces the ability of designers to work separately from programmers, and generally complicates maintenance of applications. The better way to program this is to remove the JavaScript from the HTML.

NOTE

Formally, we use an **event handler** to react to an event. Event handlers are simply methods that are designed explicitly for responding to particular events. If no response to an event is defined, the event might be passed up to another object for handling.

6.7.2 Listener Approach

Section 6.2.1 argued that the design principle of layers is a proven way of increasing maintainability and simplifying markup. The problem with the inline handler approach is that it does not make use of layers; that is, it does not separate content from behavior.

For this reason, this book will advocate and use an approach that separates all JavaScript code from the HTML markup. Although the book and its labs may occasionally illustrate a quick concept with the old-style inline handler approach, the authors prefer to replace the inline approach using one of the two approaches shown in Listing 6.10 and Listing 6.11.

```
var greetingBox = document.getElementById('example1');
greetingBox.onclick = alert('Good Morning');
```

LISTING 6.10 The "old" style of registering a listener.

The approach shown in Listing 6.10 is widely supported by all browsers. The first line in the listing creates a temporary variable for the HTML element that will trigger the event. The next line attaches the `<div>` element's `onclick` event to the event handler, which invokes the JavaScript `alert()` method (and thus annoys the user with a pop-up hello message). The main advantage of this approach is that this code can be written anywhere, including an external file that helps *uncouple* the HTML from the JavaScript. However, the one limitation with this approach (and the inline approach) is that only one handler can respond to any given element event.

```
var greetingBox = document.getElementById('example1');
greetingBox.addEventListener('click', alert('Good Morning'));
greetingBox.addEventListener('mouseOut', alert('Goodbye'));

// IE 8
greetingBox.attachEvent('click', alert('Good Morning'));
```

LISTING 6.11 The "new" DOM2 approach to registering listeners.

The use of `addEventListener()` shown in Listing 6.11 was introduced in DOM Version 2, and as such is unfortunately not supported by IE 8 or earlier. This approach has all the other advantages of the approach shown in Listing 6.10, and has the additional advantage that multiple handlers can be assigned to a single object's event.

The examples in Listing 6.10 and Listing 6.11 simply used the built-in JavaScript `alert()` function. What if we wanted to do something more elaborate when an event is triggered? In such a case, the behavior would have to be encapsulated within a function, as shown in Listing 6.12.

```
function displayTheDate() {
   var d = new Date();
   alert ("You clicked this on "+ d.toString());
}
var element = document.getElementById('example1');
element.onclick = displayTheDate;

// or using the other approach
element.addEventListener('click',displayTheDate);
```

LISTING 6.12 Listening to an event with a function

An alternative to that shown in Listing 6.12 is to use an anonymous function (that is, one without a name), as shown in Listing 6.13. This approach is especially common when the event handling function will only ever be used as a listener.

```
var element = document.getElementById('example1');
element.onclick = function() {
   var d = new Date();
   alert ("You clicked this on " + d.toString());
};
```

LISTING 6.13 Listening to an event with an anonymous function

6.7.3 Event Object

No matter which type of event we encounter, they are all DOM event objects and the event handlers associated with them can access and manipulate them. Typically we see the events passed to the function handler as a parameter named *e*.

```
function someHandler(e) {
   // e is the event that triggered this handler.
}
```

These objects have many properties and methods. Many of these properties are not used, but several key properties and methods of the event object are worth knowing.

- **Bubbles**. The `bubbles` property is a Boolean value. If an event's `bubbles` property is set to `true` then there must be an event handler in place to handle

the event or it will bubble up to its parent and trigger an event handler there. If the parent has no handler it continues to bubble up until it hits the document root, and then it goes away, unhandled.

- **Cancelable**. The `Cancelable` property is also a Boolean value that indicates whether or not the event can be cancelled. If an event is cancelable, then the default action associated with it can be canceled. A common example is a user clicking on a link. The default action is to follow the link and load the new page.
- **preventDefault**. A cancelable default action for an event can be stopped using the `preventDefault()`method as shown in Listing 6.14. This is a common practice when you want to send data asynchronously when a form is submitted, for example, since the default event of a form submit click is to post to a new URL (which causes the browser to refresh the entire page).

```
function submitButtonClicked(e) {
  if (e.cancelable){
    e. preventDefault();
  }
}
```

LISTING 6.14 A sample event handler function that prevents the default event

6.7.4 Event Types

Perhaps the most obvious event is the click event, but JavaScript and the DOM support several others. In actuality there are several classes of event, with several types of event within each class specified by the W3C. The classes are mouse events, keyboard events, form events, and frame events.

Mouse Events

Mouse events are defined to capture a range of interactions driven by the mouse. These can be further categorized as mouse click and mouse move events. Table 6.7 lists the possible events one can listen for from the mouse.

Interestingly, many mouse events can be sent at a time. The user could be moving the mouse off one `<div>` and onto another in the same moment, triggering `onmouseon` and `onmouseout` events as well as the `onmousemove` event. The `Cancelable` and `Bubbles` properties can be used to handle these complexities.

Keyboard Events

Keyboard events are often overlooked by novice web developers, but are important tools for power users. Table 6.8 lists the possible keyboard events.

Event	Description
onclick	The mouse was clicked on an element
ondblclick	The mouse was double clicked on an element
onmousedown	The mouse was pressed down over an element
onmouseup	The mouse was released over an element
onmouseover	The mouse was moved (not clicked) over an element
onmouseout	The mouse was moved off of an element
onmousemove	The mouse was moved while over an element

TABLE 6.7 Mouse Events in JavaScript

These events are most useful within input fields. We could for example validate an email address, or send an asynchronous request for a dropdown list of suggestions with each key press.

```
<input type="text" id="keyExample">
```

The input box above, for example, could be listened to and each key pressed echoed back to the user as an alert as shown in Listing 6.15.

```
document.getElementById("keyExample").onkeydown = function
myFunction(e){
   var keyPressed=e.keyCode;      //get the raw key code
   var character=String.fromCharCode(keyPressed); //convert to string
   alert("Key " + character + " was pressed");
}
```

LISTING 6.15 Listener that hears and alerts keypresses

Event	Description
onkeydown	The user is pressing a key (this happens first)
onkeypress	The user presses a key (this happens after onkeydown)
onkeyup	The user releases a key that was down (this happens last)

TABLE 6.8 Keyboard Events in JavaScript

NOTE

Unfortunately various browsers implement keyboard properties differently. If we had changed the above code to listen to the `onkeypress` event, we would have to write code like this to get the `keyCode` out.

```
if (window.event) {          // IE
  keyPressed = e.keyCode;
} else if (e.which) {        // Netscape/Firefox/Opera
  keyPressed = e.which;
}
```

Rather than write browser-testing `if` statements throughout our code, we will soon adopt the jQuery framework, which handles these idiosyncrasies for you.

Form Events

Forms are the main means by which user input is collected and transmitted to the server. Table 6.9 lists the different form events.

The events triggered by forms allow us to do some timely processing in response to user input. The most common JavaScript listener for forms is the `onsubmit` event. In the code below we listen for that event on a form with id `loginForm`. If the password field (with id `pw`) is blank, we prevent submitting to the server using

Event	Description
onblur	A form element has lost focus (that is, control has moved to a different element), perhaps due to a click or Tab key press.
onchange	Some <input>, <textarea>, or <select> field had their value change. This could mean the user typed something, or selected a new choice.
onfocus	Complementing the onblur event, this is triggered when an element gets focus (the user clicks in the field or tabs to it).
onreset	HTML forms have the ability to be reset. This event is triggered when that happens.
onselect	When the users selects some text. This is often used to try and prevent copy/paste.
onsubmit	When the form is submitted this event is triggered. We can do some prevalidation when the user submits the form in JavaScript before sending the data on to the server.

TABLE 6.9 Form Events in JavaScript

preventDefault() and alert the user. Otherwise we do nothing, which allows the default event to happen (submitting the form) as shown in Listing 6.16.

```
document.getElementById("loginForm").onsubmit = function(e){
   var pass = document.getElementById("pw").value;
   if(pass==""){
      alert ("enter a password");
      e.preventDefault();
   }
}
```

LISTING 6.16 Catching the onsubmit event and validating a password to not be blank

Section 6.8 will examine form event handling in more detail.

Frame Events

Frame events (see Table 6.10) are the events related to the browser frame that contains your web page. The most important event is the onload event, which tells us an object is loaded and therefore ready to work with. In fact, every nontrivial event listener you write requires that the HTML be fully loaded.

However, a problem can occur if the JavaScript tries to reference a particular in the HTML page that has not yet been loaded. If the code attempts to set up a listener on this not-yet-loaded , then an error will be triggered. For this reason it is common practice to use the window.onload event to trigger the execution of the rest of the page's scripts.

```
window.onload= function(){
     //all JavaScript initialization here.
}
```

Event	Description
onabort	An object was stopped from loading
onerror	An object or image did not properly load
onload	When a document or object has been loaded
onresize	The document view was resized
onscroll	The document view was scrolled
onunload	The document has unloaded

TABLE 6.10 Frame Events in JavaScript

This code will only run once the page is fully loaded and therefore all references to the page's HTML elements will be valid.

6.8 Forms

HANDS-ON EXERCISES

LAB 6 EXERCISE
Working with Forms

Chapter 4 covered the HTML for data entry forms. In that chapter it was mentioned that user form input should be validated on both the client side and the server side.

To illustrate some form-related JavaScript concepts, consider the simple HTML form depicted in Listing 6.17.

```
<form action='login.php' method='post' id='loginForm'>
  <input type='text' name='username' id='username'/>
  <input type='password' name='password' id='password'/>
  <input type='submit'></input>
</form>
```

LISTING 6.17 A basic HTML form for a login example

6.8.1 Validating Forms

Form validation is one of the most common applications of JavaScript. Writing code to prevalidate forms on the client side will reduce the number of incorrect submissions, thereby reducing server load. Although validation must still happen on the server side (in case JavaScript was circumvented), JavaScript prevalidation is a best practice. There are a number of common validation activities including email validation, number validation, and data validation. In practice regular expressions are used (covered in Chapter 12), and allow for more complete and concise scripts to validate particular fields. However, the novice programmer may not be familiar or comfortable using regex, and will often resort to copying a regex from the Internet, without understanding how it works, and therefore, will be unable to determine if it is correct. In this chapter we will write basic validation scripts without using regex to demonstrate how prevalidation client side works, leaving complicated regular expressions until Chapter 12.

Empty Field Validation

A common application of a client-side validation is to make sure the user entered something into a field. There's certainly no point sending a request to log in if the username was left blank, so why not prevent the request from working? The way to check for an empty field in JavaScript is to compare a value to both null and the empty string ("") to ensure it is not empty, as shown in Listing 6.18.

```
document.getElementById("loginForm").onsubmit = function(e){
   var fieldValue=document.getElementByID("username").value;
   if(fieldValue==null || fieldValue== ""){
      // the field was empty. Stop form submission
      e.preventDefault();
      // Now tell the user something went wrong
      alert("you must enter a username");
   }
}
```

LISTING 6.18 A simple validation script to check for empty fields

Some additional things to consider are fields like checkboxes, whose value is always set to "on". If you want to ensure a checkbox is ticked, use code like that below.

```
var inputField=document.getElementByID("license");
if (inputField.type=="checkbox"){
 if (inputField.checked)
 //Now we know the box is checked, otherwise it isn't
}
```

Number Validation

Number validation can take many forms. You might be asking users for their age for example, and then allow them to type it rather than select it. Unfortunately, no simple functions exist for number validation like one might expect from a full-fledged library. Using `parseInt()`, `isNAN()`, and `isFinite()`, you can write your own number validation function.

Part of the problem is that JavaScript is dynamically typed, so `"2" !== 2`, but `"2"==2`. jQuery and a number of programmers have worked extensively on this issue and have come up with the function `isNumeric()` shown in Listing 6.19. Note: This function will not parse "European" style numbers with commas (i.e., 12.00 vs. 12,00).

```
function isNumeric(n) {
    return !isNaN(parseFloat(n)) && isFinite(n);
}
```

LISTING 6.19 A function to test for a numeric value

More involved examples to validate email, phone numbers, or social security numbers would include checking for blank fields and making use of `isNumeric` and regular expressions as illustrated in chapter 12.

6.8.2 Submitting Forms

Submitting a form using JavaScript requires having a node variable for the form element. Once the variable, say, `formExample` is acquired, one can simply call the `submit()` method:

```
var formExample = document.getElementById("loginForm");
formExample.submit();
```

This is often done in conjunction with calling `preventDefault()` on the `onsubmit` event. This can be used to submit a form when the user did not click the submit button, or to submit forms with no submit buttons at all (say we want to use an image instead). Also, this can allow JavaScript to do some processing before submitting a form, perhaps updating some values before transmitting.

It is possible to submit a form multiple times by clicking buttons quickly, which means your server-side scripts should be designed to handle that eventuality. Clicking a submit button twice on a form should not result in a double order, double email, or double account creation, so keep that in mind as you design your applications.

6.9 Chapter Summary

This chapter has introduced the concept of client-side scripting as (optional) layers added to HTML and CSS. In addition to covering the basic syntactic elements of JavaScript, some common JavaScript objects were introduced. Techniques to listen for events, and best practices regarding fail-safe design, ensure that JavaScript will enhance existing web pages, rather than replace server-side functionality. Some form-handling examples were illustrated, and the reader is well prepared for the advanced asynchronous JavaScript and jQuery libraries that will be introduced in Chapter 15.

6.9.1 Key Terms

ActionScript
Adobe Flash
assignment
AJAX
applet
arrays
browser extension
browser plug-in
client-side scripting
conditional assignment
conditionals
constructor
document object model (DOM)
DOM document object
document root
DOM event objects
DOM tree
dot notation
dynamically typed
ECMAScript
element node
embedded JavaScript
event
event handler
exception
external JavaScript

fail-safe design	layer	node
for loops	libraries	progressive enhancement
functions	listener	property
graceful degradation	loops	String class
inline JavaScript	loop-control variable	try-catch block
Java applet	Math class	variables
JavaScript frameworks	method	XMLHttpRequest

6.9.2 Review Questions

1. What is JavaScript?
2. Discuss the advantages and disadvantages of client-side scripting.
3. How is a browser plug-in different from a browser extension?
4. How do AJAX requests differ from normal requests in the HTTP request-response loop?
5. What are software layers, and what benefit do they provide?
6. What are some reasons a user might have JavaScript disabled?
7. What kind of variable typing is used in JavaScript? What benefits and dangers arise from this?
8. What is fail-safe design, and why does it matter?
9. Compare graceful degradation with progressive enhancement.
10. What are some key DOM objects?
11. How does one access a particular HTML tag through JavaScript?
12. What is a listener?
13. When should one *throw* an object?
14. Why is JavaScript form validation not sufficient?

For additional questions, including code tracing and writing questions, please refer to the online materials included with your online access key.

6.9.3 Hands-On Practice

PROJECT 1: Simple Login Form Prevalidation

DIFFICULTY LEVEL: Beginner

Overview

You will create JavaScript prevalidation for the form in Chapter06-project01.html. This project builds on Chapter 4 Project 3 (the Photo sharing site upload form).

HANDS-ON EXERCISES
PROJECT 6.1

Instructions

1. You will need to link to an external JavaScript file in the head of the page so that you can write code in its own file.

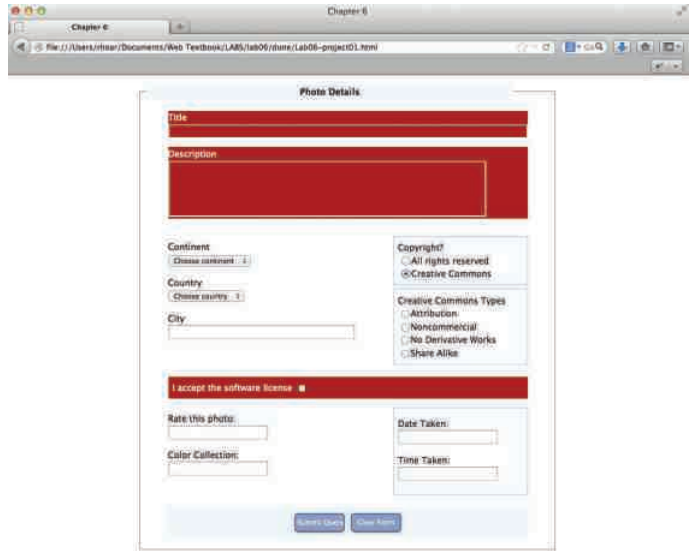


FIGURE 6.21 Screenshot of the Photo form, being prevalidated to detect blank fields

2. You should define a CSS style to use when highlighting a blank field.
3. Set up a listener on the form's submit event so that the code prevents submission of the form (`preventDefault()`) if either the title or description field is left blank or the accept license box is not checked, but otherwise submits the form.
4. Enhance the JavaScript so that blank fields trigger a change in the appearance of the form (using the style defined earlier).
5. Add another listener to the fields so that when the user types into a field (changed event) JavaScript removes the red color you just added.

Test

1. Test the form in the browser. Try submitting the form with either field blank. You should see the field highlighted and notice the page will not be refreshed as shown in Figure 6.21.
2. Type into one of the highlighted fields, and the error color should be immediately removed.

HANDS-ON EXERCISES
PROJECT 6.2

PROJECT 2: Write a Node Highlighting Helper Script

DIFFICULTY LEVEL: Intermediate

Overview

This exercise will be to write a helper script that could theoretically be used on any web page to help identify the elements, simply by including your JavaScript file! For the sake of illustration we will use Chapter 4 Project 1 as the basis for testing.

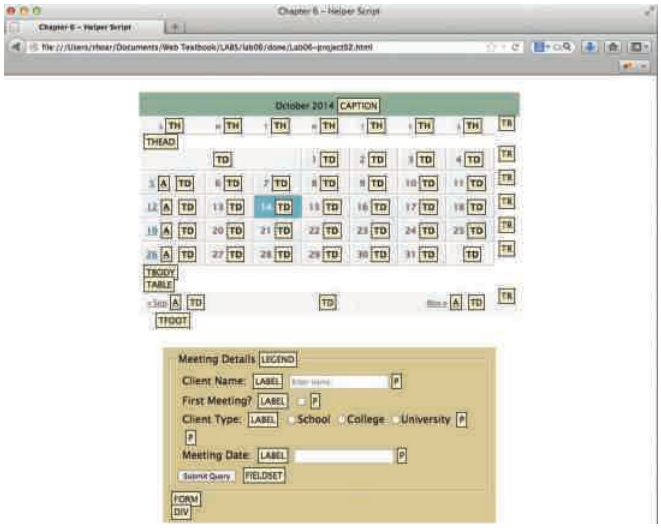


FIGURE 6.22 Screenshot of the helper script in action on the book database calendar page

Instructions

1. Define the script in a source file called highlightNodes.js.
2. This script should navigate every element in the DOM, and for each element in the body determine whether it is a textNode (type 3) or not.
3. Now add to your script code to create a new child node for every non-text node encountered. This new node should take on the class `"hoverNode"` and `innerHTML` equal to the parent tag name. Define appropriate styles for that CSS class.
4. Now add listeners so that when you click on the newly created nodes, they will alert you to information about the tag name, so that when a node is clicked a pop-up alerts us to the details about that node including its `ID` and `innerHTML`.

Test

1. By loading this script onto any page, all the tags should be identified and yellow boxes pop-up as shown in Figure 6.22.
2. Reflect on how you could enhance this script into a more useful tool to help with web development and debugging.

PROJECT 3: Progressive Enhancement Art Gallery Search

DIFFICULTY LEVEL: Intermediate

Overview

You will build upon your existing HTML page designed in Chapter 4 Project 2, but replace the existing content with an enhanced version (progressive enhancement) where a small piece of JavaScript is added so that every image can be hovered on to see a larger thumbnail as needed.

HANDS-ON EXERCISES
PROJECT 6.3

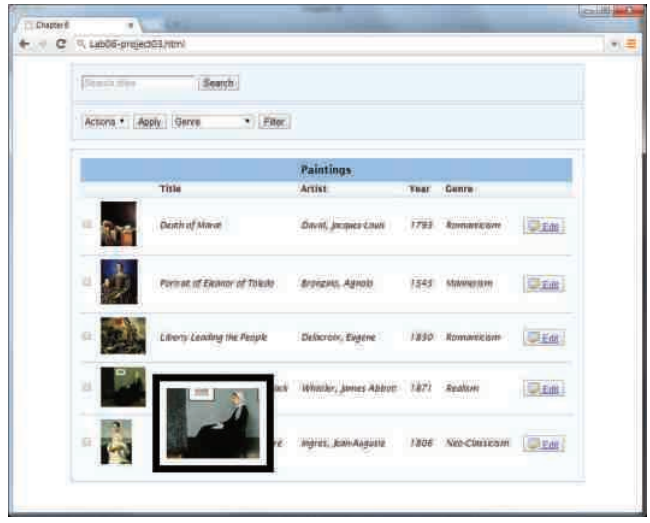


FIGURE 6.23 Screenshot of the progressive enhancement mouseover image close-up

Instructions

1. Like the previous two projects, begin by adding a link to a JavaScript file in the head of your page (or right before the `</body>` tag).
2. Slightly modify the HTML to add a class for each thumbnail image.
3. In your JavaScript file, write a loop to seek out all the img tags with the newly defined class (hint: `querySelectorAll()`).
4. For each image, attach a listener on the `mouseIn` event to create a new `<span>` with a larger image inside (based on the `src` attribute). Add another listener on the `mouseOut` event to hide the newly created `<span>`.

Test

1. Reload the page, and see that as you hover over images, larger quality thumbnails are fetched and seen in a `<div>` over the mouse location as shown in Figure 6.23.
2. As you move your mouse out the page should return to the way it was before you hovered.

6.9.4 References

1. Google Developers. [Online]. https://developers.google.com/webmasters/ajax-crawling/docs/specification.
2. W3C. Document Object Model. [Online]. http://www.w3.org/DOM/.
3. W3C. Selectors API. [Online]. http://www.w3.org/TR/selectors-api/#examples.
4. W3C. Document Object Model Events. [Online]. http://www.w3.org/TR/DOM-Level-2-Events/events.html.

8 Introduction to Server-Side Development with PHP

CHAPTER OBJECTIVES

In this chapter you will learn . . .

- What server-side development is
- What the main server-side technologies are
- The responsibilities of a web server as well as how PHP works
- PHP syntax through numerous examples
- PHP control structures
- PHP functions

This chapter introduces the principles and practices of server-side development using the LAMP (Linux, Apache, MySQL, and PHP) environment. Previous chapters have demonstrated how HTML, CSS, and JavaScript can be used to build attractive, well-defined documents for consumption through web browsers. These next few chapters will teach you how to generate HTML programmatically using PHP in response to client requests.

8.1 What Is Server-Side Development?

While the basic relationship of a client-server model was covered in Chapters 1, 4, and 6, the role of server-side development is perhaps still unclear. The basic hosting of your files is achieved through a web server whose responsibilities are described below. Server-side development is much more than web hosting: it involves the use of a programming technology like PHP or ASP.NET to create scripts that dynamically generate content.

It is important to remember that when developing server-side scripts, you are writing software, just like a C or Java programmer would do, with the major distinction that your software runs on a web server and uses the HTTP request-response loop for most interactions with the clients. This distinction is significant, since it invalidates many classic software development patterns, and requires different thinking for many seemingly simple software principles like data storage and memory management.

8.1.1 Comparing Client and Server Scripts

In Chapter 6 you encountered JavaScript, a client-side web programming language (or simply a **script**). The fundamental difference between client and server scripts is that in a client-side script the code is executed on the client browser, whereas in a server-side script, it is executed on the web server. As you saw in Chapter 6, client-side JavaScript code is downloaded to the client and is executed there. The server sends the JavaScript (that the user could look at), but you have no guarantee that the script will even execute.

In contrast, server-side source code remains hidden from the client as it is processed on the server. The clients never get to see the code, just the HTML output from the script. Figure 8.1 illustrates how client and server scripts differ.

The location of the script also impacts what resources it can access. Server scripts cannot manipulate the HTML or DOM of a page in the client browser as is possible with client scripts. Conversely, a server script can access resources on the web server whereas the client cannot. Understanding where the scripts reside and what they can access is essential to writing quality web applications.

8.1.2 Server-Side Script Resources

A server-side script can access any resources made available to it by the server. These resources can be categorized as data storage resources, web services, and software applications, as can be seen in Figure 8.2.

The most commonly used resource is **data storage**, often in the form of a connection to a database management system. A **database management system (DBMS)** is a software system for storing, retrieving, and organizing large amounts of data.

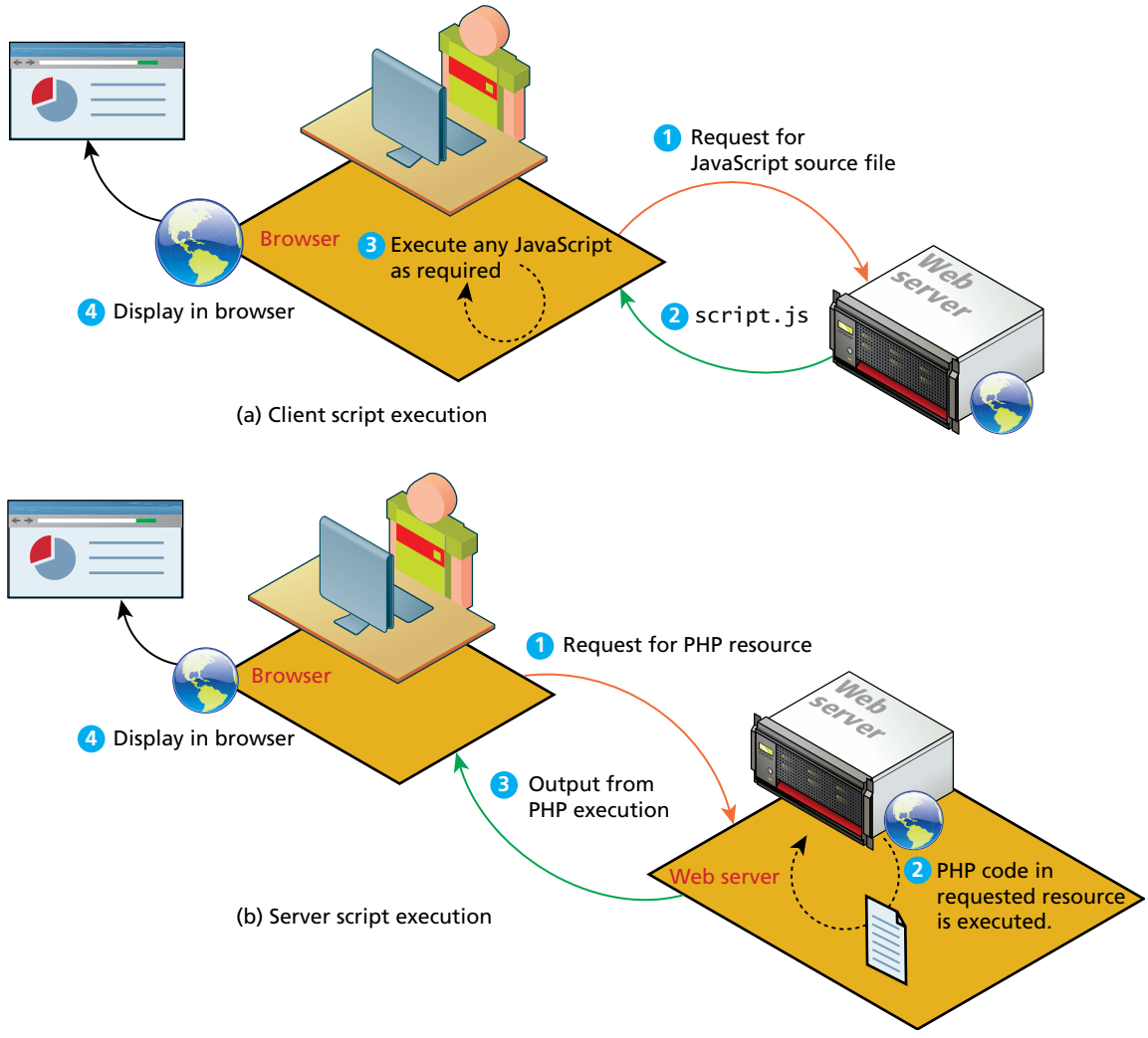


FIGURE 8.1 Comparison of (a) client script execution and (b) server script execution

The term database is often used interchangeably to refer to a DBMS, but it is also used to refer to organized data in general, or even to the files used by the DBMS. Chapter 10 will introduce databases; most subsequent chapters will make use of databases as well. While almost every significant real-world website uses some type of database, many websites also make use of the server's file system; for example, as a place to store user uploads.

The next suites of resources are web services, often offered by third-party providers. Web services use the HTTP protocol to return XML or other data formats

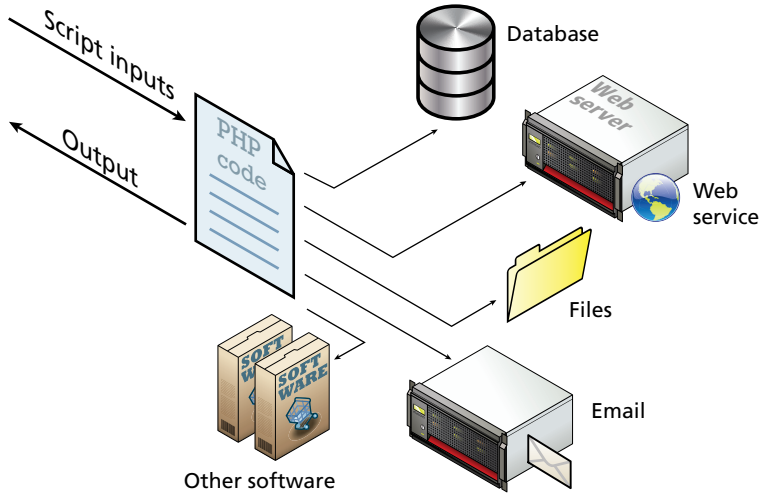


FIGURE 8.2 Server scripts have access to many resources.

and are often used to extend the functionality of a website. An example is a geo-location service that returns city and country names in response to geographic coordinates. Chapter 17 covers the consumption and creation of web services.

Finally, there is any additional software that can be installed on a server or accessed via a network connection. Using other software means, server applications can send and receive email, access user authentication services, and use network accessible storage. You could connect a web application to the regular telephone network to send texts or make calls.

8.1.3 Comparing Server-Side Technologies

As you learned in Chapter 1, there are several different server-side technologies for creating web applications. The most common include:

- ASP (Active Server Pages). This was Microsoft's first server-side technology (also called ASP Classic). Like PHP, ASP code (using the VBScript programming language) can be embedded within the HTML; though it supported classes and *some* object-oriented features, most developers did not make use of these features. ASP programming code is interpreted at run time, hence it can be slow in comparison to other technologies.
- ASP.NET. This replaced Microsoft's older ASP technology. ASP.NET is part of Microsoft's .NET Framework and can use any .NET programming language (though C# is the most commonly used). ASP.NET uses an explicitly object-oriented approach that typically takes longer to learn than ASP or PHP, and is often used in larger corporate web application systems. It also uses special

markup called web server controls that encapsulate common web functionality such as database-driven lists, form validation, and user registration wizards. A recent extension called ASP.NET MVC makes use of the Model-View-Controller design pattern (this pattern will be covered in Chapter 14). ASP.NET pages are compiled into an intermediary file format called MSIL that is analogous to Java's byte-code. ASP.NET then uses a JIT (Just-In-Time) compiler to compile the MSIL into machine executable code so its performance can be excellent. However, ASP.NET is essentially limited to Windows servers.

- **JSP (Java Server Pages)**. JSP uses Java as its programming language and like ASP.NET it uses an explicit object-oriented approach and is used in large enterprise web systems and is integrated into the J2EE environment. Since JSP uses the Java Runtime Engine, it also uses a JIT compiler for fast execution time and is cross-platform. While JSP's usage in the web as a whole is small, it has a substantial market share in the intranet environment, as well as with very large and busy sites.
- **Node.js**. This is a more recent server environment that uses JavaScript on the server side, thus allowing developers already familiar with JavaScript to use just a single language for both client-side and server-side development. Unlike the other development technologies listed here, node.js is also its own web server software, thus eliminating the need for Apache, IIS, or some other web server software.
- **Perl**. Until the development and popularization of ASP, PHP, and JSP, Perl was the language typically used for early server-side web development. As a language, it excels in the manipulation of text. It was commonly used in conjunction with the **Common Gateway Interface (CGI)**, an early standard API for communication between applications and web server software.
- **PHP**. Like ASP, PHP is a dynamically typed language that can be embedded directly within the HTML, though it now supports most common object-oriented features, such as classes and inheritance. By default, PHP pages are compiled into an intermediary representation called **opcodes** that are analogous to Java's byte-code or the .NET Framework's MSIL. Originally, PHP stood for *personal home pages*, although it now is a recursive acronym that means *PHP: Hypertext Processor*.
- **Python**. This terse, object-oriented programming language has many uses, including being used to create web applications. It is also used in a variety of web development frameworks such as Django and Pyramid.
- **Ruby on Rails**. This is a web development framework that uses the Ruby programming language. Like ASP.NET and JSP, Ruby on Rails emphasizes the use of common software development approaches, in particular the MVC design pattern. It integrates features such as templates and engines that aim to reduce the amount of development work required in the creation of a new site.

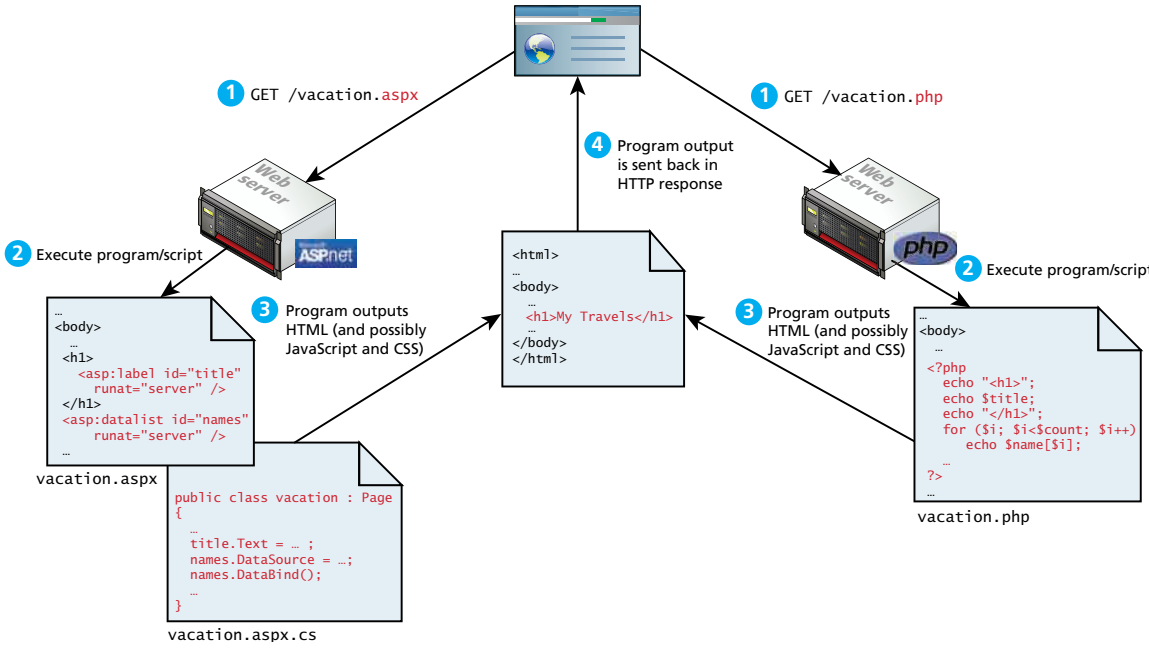


FIGURE 8.3 Web development technologies

All of these technologies share one thing in common: using programming logic, they generate HTML and possibly CSS and JavaScript on the server and send it back to the requesting browser, as shown in Figure 8.3.

Of these server-side technologies, ASP.NET and PHP appear to have the largest market share. ASP.NET tends to be more commonly used for enterprise applications and within intranets. Partly due to the massive user base of WordPress, PHP is the most commonly used web development technology, and will be the technology we will use in this book.

NOTE

Determining the market share of different development environments is not straightforward. Because server-side technology is used on the server and does not show up on the browser, analytic companies such as **builtwith.com** must use various proxy measures such as the file extensions (which can be absent) and "fingerprints" within the generated HTML to determine the server environment that created a given site. Doing so allows you to see that different technologies (for instance JSP) have quite different market share depending on the popularity of the site (which is a rough measure of not only the site's user load but its size and complexity as well), as can be seen in Figure 8.4.

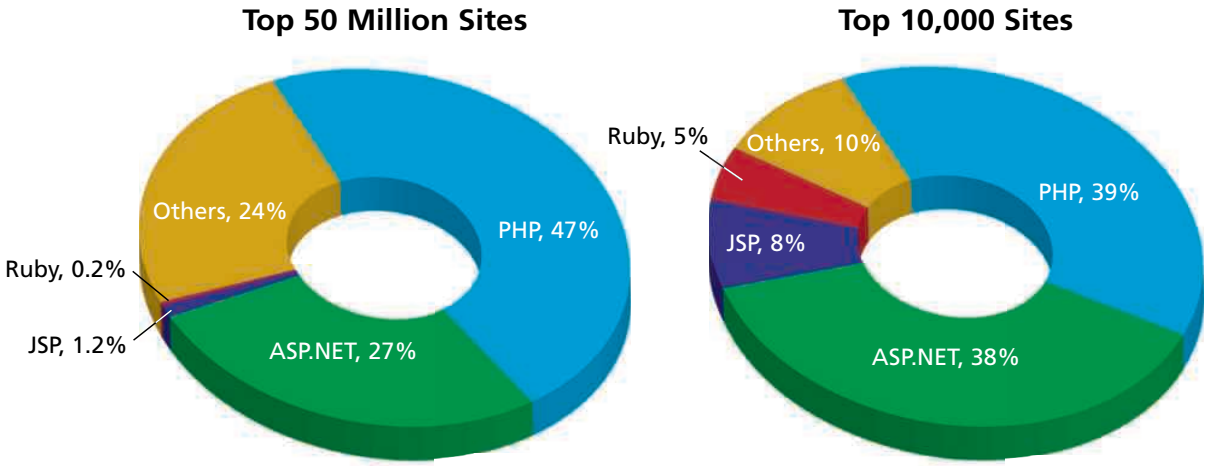


FIGURE 8.4 Market share of web development environments (data courtesy of BuiltWith.com)

8.2 A Web Server's Responsibilities

As you learned in Chapter 1, in the client-server model the server is responsible for answering all client requests. No matter how static or simple the website is, there must be a web server somewhere configured to answer requests for that domain. Once a web server is configured and the IP address associated through a DNS server (see Chapter 1), it can then start listening for and answering HTTP requests. In the very simplest case the server is hosting static HTML files, and in response to a request sends the content of the file back to the requester.

A web server has many responsibilities beyond responding to requests for HTML files. These include handling HTTP connections, responding to requests for static and dynamic resources, managing permissions and access for certain resources, encrypting and compressing data, managing multiple domains and URLs, managing database connections, cookies, and state, and uploading and managing files.

As mentioned in Chapter 1, throughout this textbook you will be using the LAMP software stack, which refers to the Linux operating system, the Apache web server, the MySQL DBMS, and the PHP scripting language. Outside of the chapters on security and deployment, this book will not examine the Linux operating system in any detail. However, since the Apache web server is an essential part of the web development pipeline, one should have some insight into how it works and how it interacts with PHP.

NOTE

To run the examples in this book you will need to use a LAMP stack or variant. Since the server code relies entirely on the web-hosting environment, some code written for LAMP may not run on a Windows/IIS server and vice versa. Selecting the hosting environment is a critical decision since it will influence how you write your software.

There are several free packages such as XAMPP that let you run the LAMP stack on your Windows or Mac computer. Section 8.2.4 provides more information about installing one of these on your computer.

8.2.1 Apache and Linux

You can consider the Apache web server as the intermediary that interprets HTTP requests that arrive through a network port and decides how to handle the request, which often requires working in conjunction with PHP; both Apache and PHP make use of configuration files that determine exactly how requests are handled, as shown in Figure 8.5.

Apache runs as a daemon on the server. A daemon is an executing instance of a program (also called a process) that runs in the background, waiting for a specific event that will activate it. As a background process, the Apache daemon (also

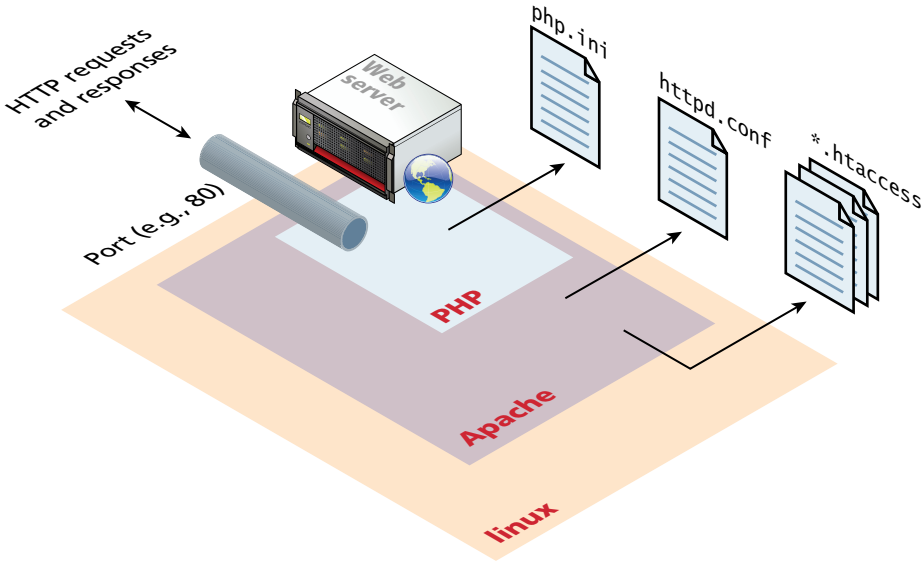


FIGURE 8.5 Linux, Apache, and PHP together

known by its OS name, `httpd`) waits for incoming HTTP requests. When a request arrives, Apache then uses modules to determine how to respond to the request.

PRO TIP

In Linux, daemons are usually configured to start running when the OS boots and can be manually started and stopped by the root user. Whenever a configuration option is changed (or a server process is hung up), you must restart Apache.

On many Linux systems only the root user can restart Apache using a command like `/etc/init.d/httpd restart` (CentOS) or `/usr/sbin/apachectl restart` (on Mac). Plug and play environments will have a GUI option to restart the Apache server.

In Apache, a **module** is a compiled extension (usually written in the C programming language) to Apache that helps it *handle* requests. For this reason, these modules are also sometimes referred to as **handlers**. Figure 8.6 illustrates that when a request comes into Apache, each module is given an opportunity to handle some aspect of the request.

Some modules handle authorization, others handle URL rewriting, while others handle specific extensions. In Chapter 20, you will learn more about how Apache configures these handlers.

8.2.2 Apache and PHP

As can be seen in Figure 8.6, PHP is usually installed as an Apache module (though it can alternately be installed as a CGI binary). The PHP module `mod_php5` is sometimes referred to as the **SAPI** (Server Application Programming Interface) layer since it handles the interaction between the PHP environment and the web server environment.

Apache runs in two possible modes: **multi-process** (also called **preforked**) or **multi-threaded** (also called **worker**), which are shown in Figure 8.7.

The default installation of Apache runs using the multi-process mode. That is, each request is handled by a separate process of Apache; the term **fork** refers to the operating system creating a copy of an already running process. Since forking is time intensive, Apache will prefork a set number of additional processes in advance of their being needed. Forking is relatively efficient on Unix-based operating systems, but is slower on Windows-based operating systems. As well, a key advantage of multi-processing mode is that each process is insulated from other processes; that is, problems in one process can't affect other processes.

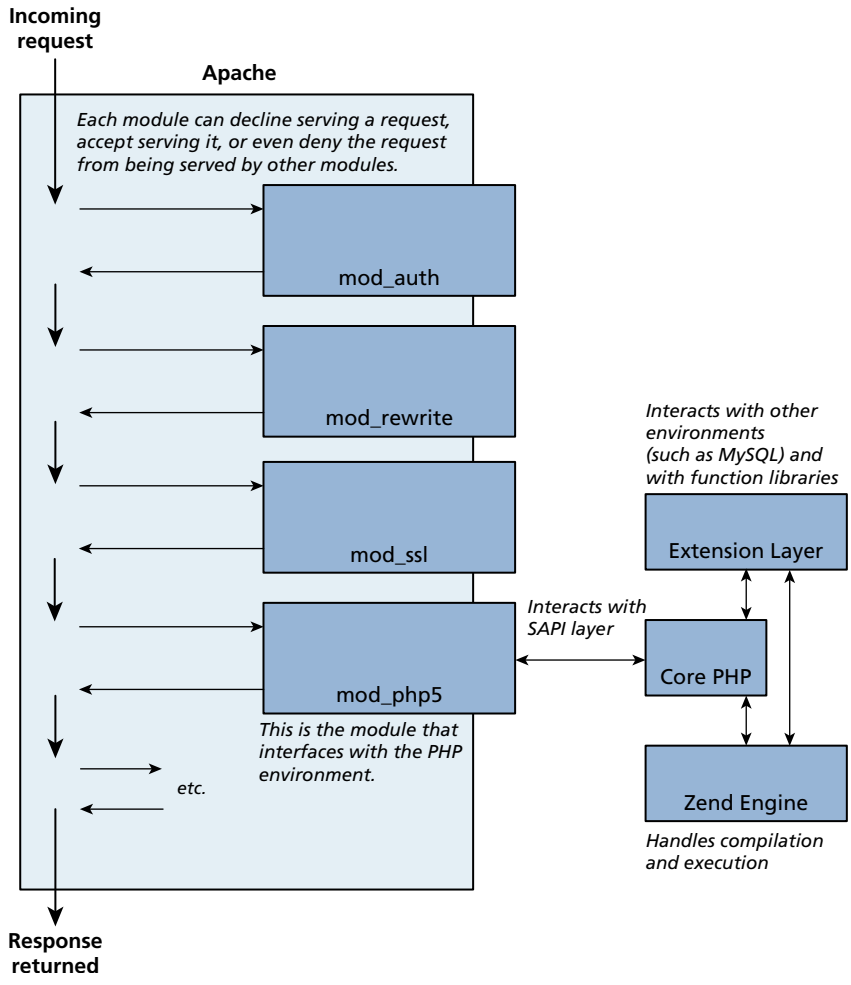


FIGURE 8.6 Apache modules and PHP

In the multi-threaded mode, a smaller number of Apache processes are forked. Each of the processes runs multiple threads. A **thread** is like a light-weight process that is contained within an operating system process. A thread uses less memory than a process, and typically threads share memory and code; as a consequence, the multi-threaded mode typically scales better to large loads. When using this mode, all modules running within Apache have to be thread-safe. Unfortunately, not every PHP module is thread-safe, and the thread safety of PHP in general is quite disputed.

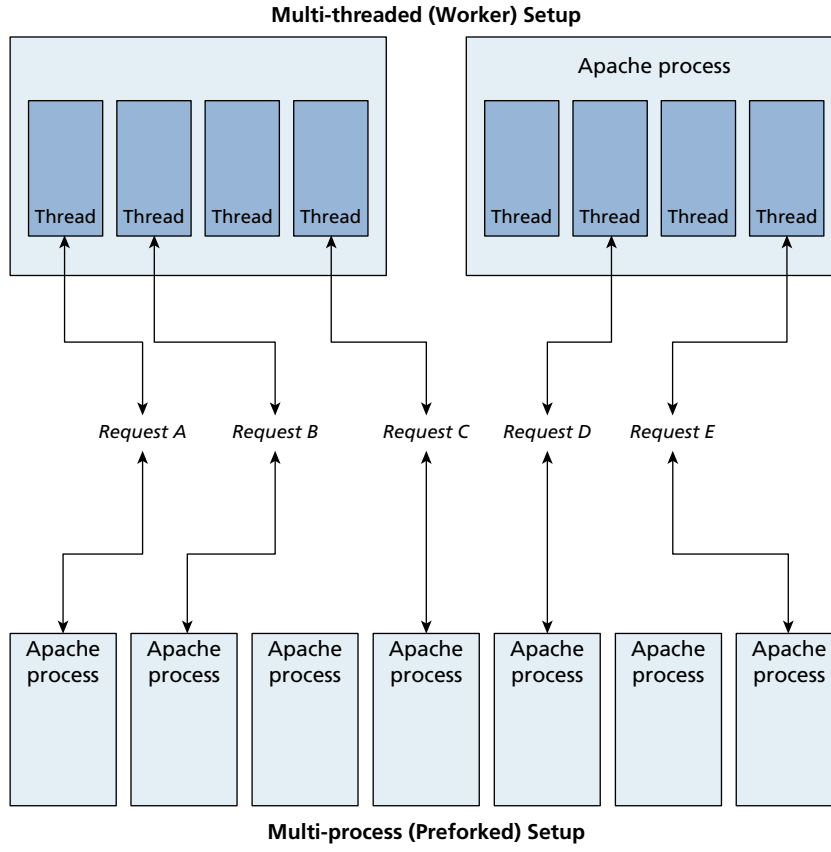


FIGURE 8.7 Multi-threaded versus multi-process

8.2.3 PHP Internals

PHP itself is written in the C programming language and is composed of three main modules:

PHP core. The Core module defines the main features of the PHP environment, including essential functions for variable handling, arrays, strings, classes, math, and other core features.

Extension layer. This module defines functions for interacting with services outside of PHP. This includes libraries for MySQL (and other databases), FTP, SOAP web services, and XML processing, among others.

Zend Engine. This module handles the reading in of a requested PHP file, compiling it, and executing it. Figure 8.8 illustrates (somewhat imaginatively) how the Zend Engine operates behind the scenes when a PHP page is requested. The Zend Engine is a virtual machine (VM) analogous to the Java Virtual Machine or the Common

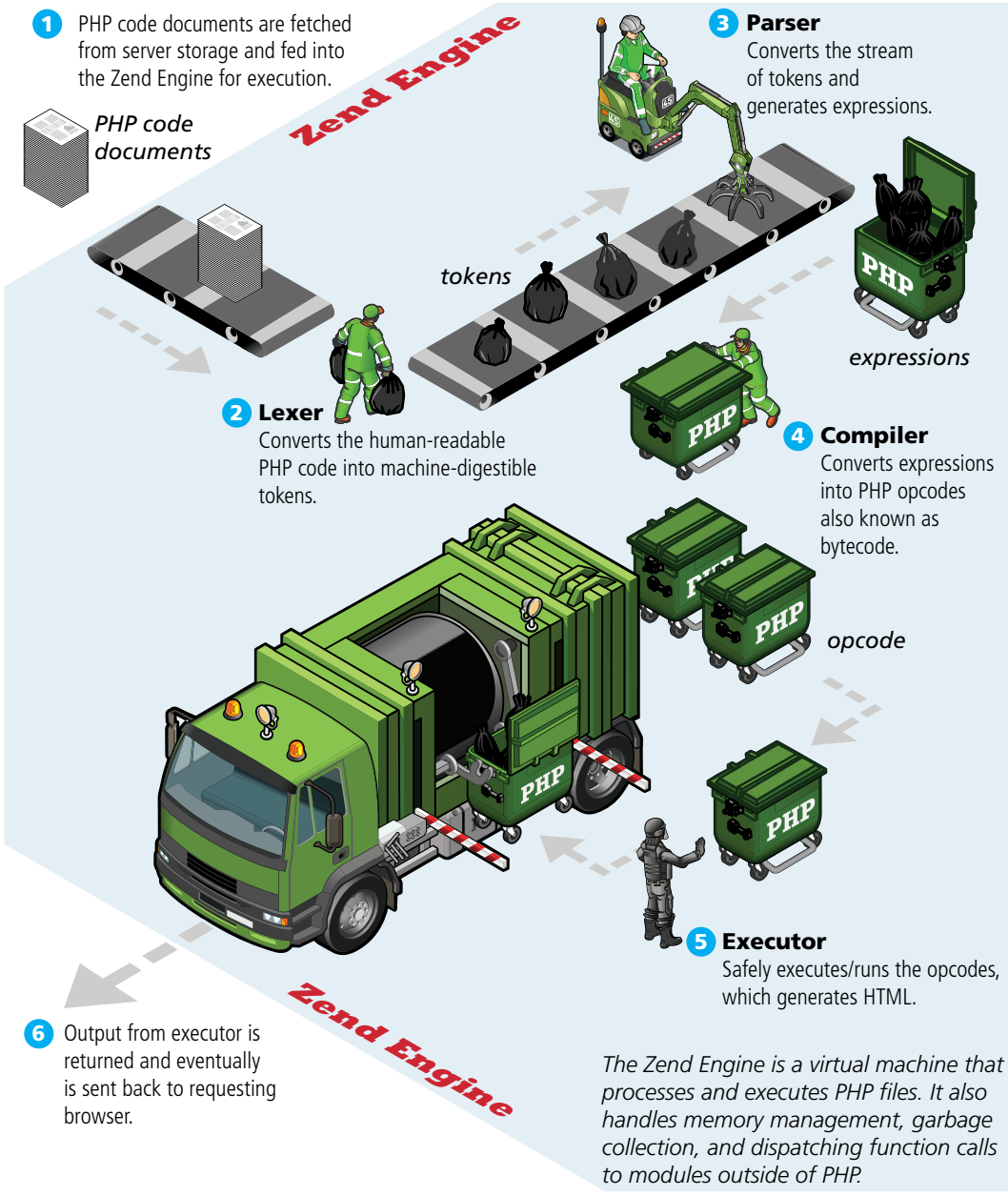


FIGURE 8.8 Zend Engine

Language Runtime in the .NET Framework. A VM is a software program that simulates a physical computer; while a VM can operate on multiple platforms, it has the disadvantage of executing slower than a native binary application.

8.2.4 Installing Apache, PHP, and MySQL for Local Development

HANDS-ON EXERCISES

LAB 8 EXERCISE
Install LAMP

One of the true benefits of the LAMP web development stack is that it can run on almost any computer platform. Similarly, the AMP part of LAMP can run on most operating systems, including Windows and the Mac OS. Thus it is possible to install Apache, PHP, and MySQL on your own computer.

While there are many different ways that one can go about installing this software, you may find that the easiest and quickest way to do so is to use the XAMPP For Windows installation package (available at http://www.apachefriends.org/en/xampp-windows.html) or the MAMP for Mac installation package (available at http://www.mamp.info/en/index.html). Both of these installation packages install and configure Apache, PHP, and MySQL.

Once the XAMPP package is installed in Windows, you can then run the XAMPP control panel, which looks similar to that shown in Figure 8.9 (as you can see in this screen capture, we did not install all the components). You may need to click the appropriate Start buttons to launch Apache (and later MySQL).

Once Apache has started, any subsequent PHP requests in your browser will need to use the localhost domain, as shown in Figure 8.10.

Now you are ready to start creating your own PHP pages. If you used the default XAMPP installation location, your PHP files will have to be saved somewhere within the C:\xampp\htdocs folder. On a Mac computer, Apache comes installed (though not activated). The http.conf file is found in /etc/apache2/ and the default location for your PHP files is /Library/Webserver/Documents. If you are using a lab server or an external web host, then check the appropriate documentation to find out where you will need to save or upload your PHP files.

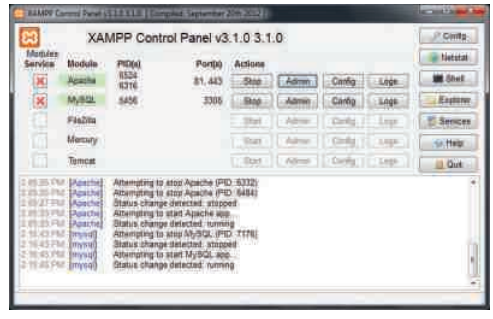


FIGURE 8.9 XAMPP control panel

FIGURE 8.10 localhost in browser

NOTE

You may notice a port specification in many of our screen captures, which is something the authors had to do because of conflicts with Microsoft Internet Information Server (IIS).

If you also have IIS or Microsoft Visual Studio (VS) installed on your computer, there is likely to be a conflict between Apache and IIS/VS as both will try to make use of the default port 80 for web requests. In that case, you will have to change the port settings in IIS and VS, which is relatively complicated, or change the port settings for Apache, which is relatively easy.

To change the port settings for Apache, you will need to edit the **http.conf** file (in the **apache\conf** folder), which is easily accessible via the Config button in the XAMPP control panel. You then need to change the port from 80 to 81 (or to something else) on the following two lines:

```
Listen 81
...
ServerName localhost:81
```

After making the changes, restart Apache. You will then need to change all subsequent PHP-related requests to include the port number. For instance:

```
http://localhost:81/xampp/index.php
```

8.3 Quick Tour of PHP

PHP, like JavaScript, is a dynamically typed language. Unlike JavaScript it uses classes and functions in a way consistent with other object-oriented languages such as C++, C#, and Java, though with some minor exceptions. The syntax for loops, conditionals, and assignment is identical to JavaScript, only differing when you get to functions, classes, and in how you define variables. This section will cover the essential features of PHP; some of it will be quite cursory and will leave to the reader the responsibility of delving further into language specifics. There are a wide variety of PHP books that cover PHP in significantly more detail than is possible here, and the reader is encouraged to explore some of these books and online resources.[1,2,3]

8.3.1 PHP Tags

HANDS-ON EXERCISES

LAB 8 EXERCISE
Your First PHP Script

The most important fact about PHP is that the programming code can be embedded directly within an HTML file. However, instead of having an .html extension, a PHP file will usually have the extension .php. As can be seen in Listing 8.1, PHP programming code must be contained within an opening <?php tag and a matching closing ?> tag in order to differentiate it from the HTML. The programming code within the <?php and the ?> tags is interpreted and executed, while any code outside the tags is echoed directly out to the client.

```
<?php
$user = "Randy";
?>
<!DOCTYPE html>
<html>
<body>
<h1>Welcome <?php echo $user; ?></h1>
<p>
The server time is
<?php
echo "<strong>";
echo date("H:i:s");
echo "</strong>";
?>
</p>
</body>
</html>
```

LISTING 8.1 PHP tags

You may be wondering what the code in Listing 8.1 would look like when requested by a browser. Listing 8.2 illustrates the HTML output from the PHP script in Listing 8.1. Notice that no PHP is sent back to the browser.

```
<!DOCTYPE html>
<html>
<body>
<h1>Welcome Randy</h1>
<p>
The server time is <strong>02:59:09</strong>
</p>
</body>
</html>
```

LISTING 8.2 Listing 8.1 in the browser

Listing 8.1 also illustrates the very common practice (especially when first learning PHP) for a PHP file to have HTML markup and PHP programming woven together. While this is convenient when first learning PHP, as your code becomes more complex, doing so will make your PHP pages very difficult to understand and modify. Indeed, the authors have seen PHP files that are several thousands of lines long, which are quite a nightmare to maintain. In software design lingo, we would say that such PHP files tightly couple presentation and logic, and as such, are less than ideal from a design and maintainability standpoint.

Alternating between HTML and PHP should instead be done in a deliberate and logical way. In this book, we will recommend a layered approach where PHP classes are developed to handle the business and data logic, while smaller presentation PHP files will be used for presenting the information. These presentation scripts will alternate between HTML and PHP, leaving the majority of our PHP files HTML free.

You may recall that in Chapter 6 you learned about a similar set of design issues with JavaScript. Rather than intermixing HTML and JavaScript, it was recommended to use the layer design principle, which keeps most JavaScript within separate .js files and uses event listeners to associate HTML elements with JavaScript events so as to remove JavaScript code from appearing within the HTML.

Figure 8.11 illustrates how these two approaches might look; notice how the second approach makes use of `include` statements, which insert the content of the specified file into the current file (see Section 8.4.6 for more information). Since PHP is not a compiled environment like C# or Java, libraries of user-defined functions and classes must be included in any PHP file that will be using them. As can be seen in Figure 8.11, this approach reduces the intermixing of HTML and PHP.

8.3.2 PHP Comments

Programmers are supposed to write documentation to provide other developers (and themselves) guidance on certain parts of a program. In PHP any writing that

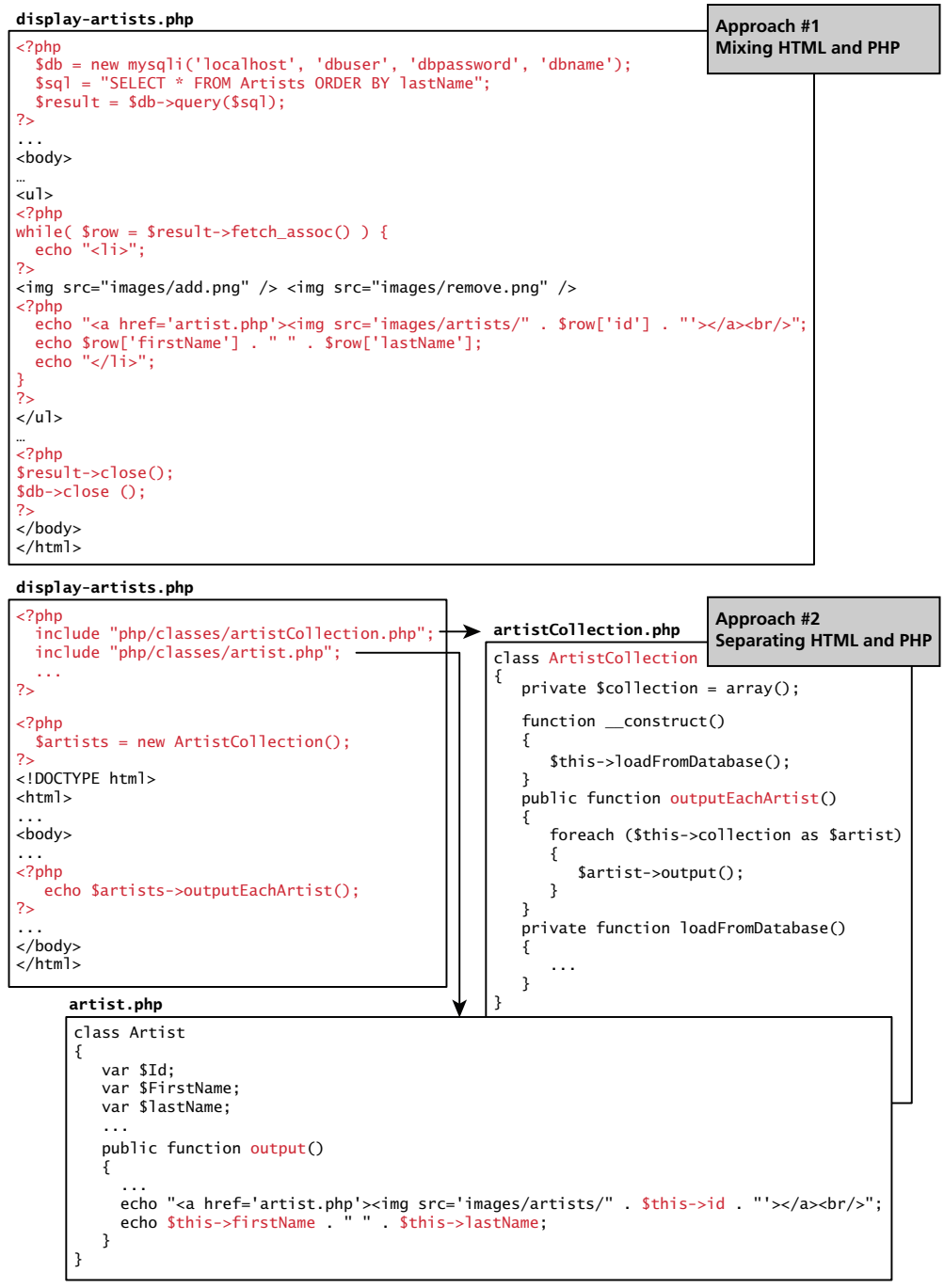


FIGURE 8.11 Two approaches to PHP coding

is a comment is ignored when the script is interpreted, but visible to developers who need to write and maintain the software. The types of comment styles in PHP are:

- **Single-line comments.** Lines that begin with a # are comment lines and will not be executed.
- **Multiline (block) comments.** Each PHP script and each function within it are ideal places to include a large comment block. These comments begin with a /* and encompass everything that is encountered until a closing */ tag is found. These tags cannot be nested.

  A comment block above a function or at the start of a file is a good place to write, in normal language, what this function does. By using the /** tag to open the comment instead of the standard /*, you are identifying blocks of comment that can later be parsed for inclusion in generated documents.
- **End-of-line comments.** Comments need not always be large blocks of natural language. Sometimes a variable needs a little blurb to tell the developer what it's for, or a complex portion of code needs a few comments to help the programmer understand the logic. Whenever // is encountered in code, everything up to the end of the line is considered a comment. These comments are sometimes preferable to the block comments because they do not interfere with one another, but are unable to span multiple lines of code.

These different commenting styles are also shown in Listing 8.3.

```
<?php

# single-line comment

/*
This is a multiline comment.
They are a good way to document functions or complicated blocks of code
*/

$artist = readDatabase(); // end-of-line comment

?>
```

LISTING 8.3 PHP comments

8.3.3 Variables, Data Types, and Constants

Variables in PHP are dynamically typed, which means that you as a programmer do not have to declare the data type of a variable. Instead the PHP engine makes a best

HANDS-ON EXERCISES

LAB 8 EXERCISE
PHP Variables

guess as to the intended type based on what it is being assigned. Variables are also **loosely typed** in that a variable can be assigned different data types over time.

To declare a variable you must preface the variable name with the dollar ($) symbol. Whenever you use that variable, you must also include the $ symbol with it. You can assign a value to a variable as in JavaScript's right-to-left assignment, so creating a variable named `count` and assigning it the value of 42 would be done with:

```
$count = 42;
```

You should note that in PHP the name of a variable is case-sensitive, so `$count` and `$Count` are references to two different variables. In PHP, variable names can also contain the underscore character, which is useful for readability reasons.

NOTE

If you do not assign a value to a variable and simply define its name, it will be undefined. You can check to see whether a variable has been set using the `isset()` function, but what's important to realize is that there are no "useful" default values in PHP. Since PHP is loosely typed, you should always define your own default values in initialization.

While PHP is loosely typed, it still does have **data types,** which describe the type of content that a variable can contain. Table 8.1 lists the main data types within PHP. As mentioned above, however, you do not declare a data type. Instead the PHP engine determines the data type when the variable is assigned a value.

A **constant** is somewhat similar to a variable, except a constant's value never changes . . . in other words it stays constant. A constant can be defined anywhere

Data Type	Description
`Boolean`	A logical true or false value
`Integer`	Whole numbers
`Float`	Decimal numbers
`String`	Letters
`Array`	A collection of data of any type (covered in the next chapter)
`Object`	Instances of classes

TABLE 8.1 PHP Data Types

Sequence	Description
\n	Line feed
\t	Horizontal tab
\\	Backslash
\$	Dollar sign
\"	Double quote

TABLE 8.2 String Escape Sequences

NOTE

String literals in PHP can be defined using either the single quote or the double quote character. If a literal is defined using double quotes, then you can also specify escape sequences using the backslash. For instance, the string "Good\nMorning" contains a newline character between the two words. Table 8.2 lists some of the common string escape sequences.

but is typically defined near the top of a PHP file via the `define()` function, as shown in Listing 8.4. The `define()` function generally takes two parameters: the name of the constant and its value. Notice that once it is defined, it can be referenced without using the `$` symbol.

```
<?php

# uppercase for constants is a programming convention
define("DATABASE_LOCAL", "localhost");
define("DATABASE_NAME", "ArtStore");
define("DATABASE_USER", "Fred");
define("DATABASE_PASSWD", "F5^7%ad");
...
# notice that no $ prefaces constant names
$db = new mysqli(DATABASE_LOCAL, DATABASE_NAME, DATABASE_USER,
  DATABASE_NAME);

?>
```

LISTING 8.4 PHP constants

PRO TIP

PHP allows variable names to also be specified at run time. This type of variable is sometimes referred to as a "variable variable" and can be convenient at times. For instance, imagine you have a set of variables named as follows:

```
<?php

$artist1 = "picasso";
$artist2 = "raphael";
$artist3 = "cezanne";
$artist4 = "rembrandt";
$artist5 = "giotto";

?>
```

If you wanted to output each of these variables within a loop, you can do so by programmatically constructing the variable name within curly brackets, as shown in the following loop:

```
for ($i = 1; $i <= 5; $i++) {
   echo ${"artist". $i};
   echo "<br/>";
}
```

8.3.4 Writing to Output

HANDS-ON EXERCISES

LAB 8 EXERCISE
PHP Output

Remember that PHP pages are programs that output HTML. To output something that will be seen by the browser, you can use the `echo()` function.

```
echo ("hello");
```

There is also an equivalent shortcut version that does not require the parentheses.

```
echo "hello";
```

Strings can easily be appended together using the concatenate operator, which is the period (.) symbol. Consider the following code:

```
$username = "Ricardo";
echo "Hello". $username;
```

This code will output `Hello Ricardo` to the browser. While this no doubt appears rather straightforward and uncomplicated, it is quite common for PHP programs to have significantly more complicated uses of the concatenation operator.

Before we get to those more complicated examples, pay particular attention to the first example in Listing 8.5. It illustrates the fact that variable references can appear within string literals (but only if the literal is defined using double quotes), which is quite unlike traditional programming languages such as Java.

Concatenation is an important part of almost any PHP program, and, based on our experience as teachers, one of the main stumbling blocks for new PHP students.

```
<?php

$firstName = "Pablo";
$lastName = "Picasso";

/*
  Example one:
  These two lines are equivalent. Notice that you can reference PHP
  variables within a string literal defined with double quotes.

  The resulting output for both lines is:

    <em>Pablo Picasso</em>

*/
echo "<em>" . $firstName . " ". $lastName. "</em>";
echo "<em> $firstName $lastName </em>";

/*
  Example two:
  These two lines are also equivalent. Notice that you can use
  either the single quote symbol or double quote symbol for string
  literals.
*/
echo "<h1>";
echo '<h1>';

/*
  Example three:
  These two lines are also equivalent. In the second example, the
  escape character (the backslash) is used to embed a double quote
  within a string literal defined within double quotes.
*/
echo '<img src="23.jpg" >';
echo "<img src=\"23.jpg\" >";

?>
```

LISTING 8.5 PHP quote usage and concatenation approaches

As such, it is important to take some time to experiment and evaluate some sample concatenation statements as shown in Listing 8.6.

```
<?php

$id = 23;
$firstName = "Pablo";
$lastName = "Picasso";

echo "<img src='23.jpg' alt='". $firstName . " ". $lastName . "' >";
echo "<img src='$id.jpg' alt='$firstName  $lastName' >";
echo "<img src=\"$id.jpg\" alt=\"$firstName  $lastName\" >";
echo '<img src="' . $id . '.jpg" alt="' . $firstName . ' ' .
     $lastName . '" >';
echo '<a href="artist.php?id=' . $id .'">' . $firstName . ' ' .
     $lastName . '</a>';

?>
```

LISTING 8.6 More complicated concatenation examples

Try to figure out the output of each line without looking at the solutions in Figure 8.12. We cannot stress enough how important it is for the reader to be completely comfortable with these examples.

printf

As the examples in Listing 8.6 illustrate, while `echo` is quite simple, more complex output can get confusing. As an alternative, you can use the `printf()` function. This function is derived from the same-named function in the C programming language and includes variations to print to string and files (`sprintf`, `fprintf`). The function takes at least one parameter, which is a string, and that string optionally references parameters, which are then integrated into the first string by placeholder substitution.[4] The `printf()` function also allows a developer to apply special formatting, for instance, specific date/time formats or number of decimal places.

Figure 8.13 illustrates the relationship between the first parameter string, its placeholders and subsequent parameters, precision, and output.

The `printf()` function (or something similar to it) is nearly ubiquitous in programming, appearing in many languages including Java, MATLAB, Perl, Ruby, and others. The advantage of using it is that you can take advantage of built-in output formatting that allows you to specify the type to interpret each parameter as well as being able to succinctly specify the precision of floating-point numbers.

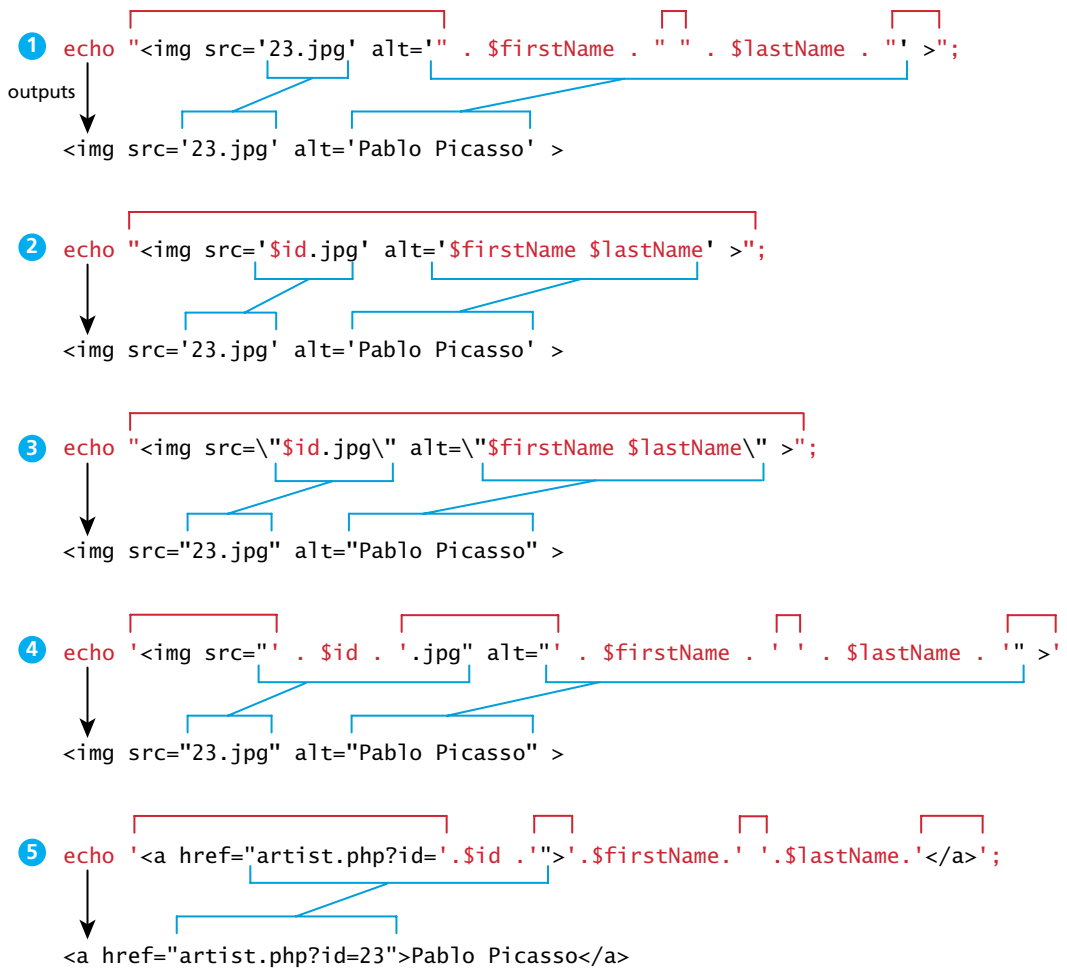


FIGURE 8.12 More complicated concatenation examples explained

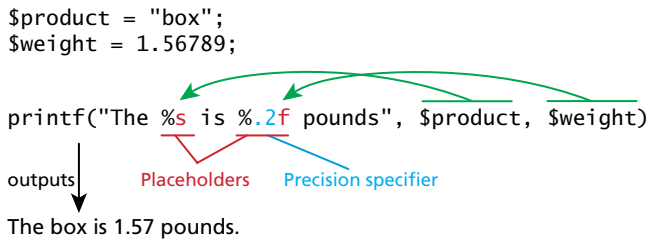


FIGURE 8.13 Illustration of components in a printf statement and output

Each placeholder requires the percent (%) symbol in the first parameter string followed by a type specifier. Common type specifiers are `b` for binary, `d` for signed integer, `f` for float, `o` for octal, and `x` for hexadecimal. Precision is achieved in the string with a period (`.`) followed by a number specifying how many digits should be displayed for floating-point numbers.

For a complete listing of the `printf()` function, refer the function at php.net.[4] When programming, you may prefer to use `printf()` for more complicated formatted output, and use `echo` for simpler output.

8.4 Program Control

Just as with most other programming languages there are a number of conditional and iteration constructs in PHP. There are `if` and `switch`, and `while`, `do while`, and `for` loops familiar to most languages as well as the `foreach` loop.

8.4.1 if . . . else

The syntax for conditionals in PHP is almost identical to that of JavaScript. In this syntax the condition to test is contained within `()` brackets with the body contained in `{}` blocks. Optional `else if` statements can follow, with an `else` ending the branch. Listing 8.7 uses a conditional to set a greeting variable, depending on the hour of the day.

```
// if statement with condition
if ( $hourOfDay > 6 && $hourOfDay < 12 ) {
   $greeting = "Good Morning";
}
else if ($hourOfDay == 12) {   // optional else if
   $greeting = "Good Noon Time";
}
else {                         // optional else branch
   $greeting = "Good Afternoon or Evening";
}
```

LISTING 8.7 Conditional statement using if . . . else

It is also possible to place the body of an `if` or an `else` outside of PHP. For instance, in Listing 8.8, an alternate form of an `if . . . else` is illustrated (along with its equivalent PHP-only form). This approach will sometimes be used when the body of a conditional contains nothing but markup with no logic, though because it mixes markup and logic, it may not be ideal from a design standpoint. As well, it

NOTE

Just like with JavaScript, Java, and C#, PHP expressions use the double equals (==) for comparison. If you use the single equals in an expression, then variable assignment will occur.

As well, like those other programming languages, it is up to the programmer to decide how she or he wishes to place the first curly bracket on the same line with the statement it is connected to or on its own line.

can be difficult to match curly brackets up with this format, as perhaps can be seen in Listing 8.8. At the end of the current section an alternate syntax for program control statements is described (and shown in Listing 8.12), which makes the type of code in Listing 8.8 more readable.

```
<?php if ($userStatus == "loggedin") {  ?>
   <a href="account.php">Account</a>
   <a href="logout.php">Logout</a>
<?php }  else { ?>
   <a href="login.php">Login</a>
   <a href="register.php">Register</a>
<?php } ?>


<?php
   // equivalent to the above conditional
   if ($userStatus == "loggedin") {
      echo '<a href="account.php">Account</a> ';
      echo '<a href="logout.php">Logout</a>';
   }
   else {
      echo '<a href="login.php">Login</a> ';
      echo '<a href="register.php">Register</a>';
   }
?>
```

LISTING 8.8 Combining PHP and HTML in the same script

8.4.2 switch . . . case

The `switch` statement is similar to a series of `if . . . else` statements. An example using `switch` is shown in Listing 8.9.

HANDS-ON EXERCISES

LAB 8 EXERCISE
PHP Conditionals

```
switch ($artType) {
   case "PT":
      $output = "Painting";
      break;
   case "SC":
      $output = "Sculpture";
      break;
   default:
      $output = "Other";
}

// equivalent
if ($artType == "PT")
   $output = "Painting";
else if ($artType == "SC")
   $output = "Sculpture";
else
   $output = "Other";
```

LISTING 8.9 Conditional statement using switch

NOTE

Be careful with mixing types when using the `switch` statement: if the variable being compared has an integer value, but a case value is a string, then there will be type conversions that will create some unexpected results. For instance, the following example will output `"Painting"` because it first converts the `"PT"` to an integer (since `$code` currently contains an integer value), which is equal to the integer 0 (zero).

```
$code = 0;
switch($code) {
  case "PT":
    echo "Painting";
    break;
  case 1:
    echo "Sculpture";
    break;
  default:
    echo "Other";
}
```

HANDS-ON EXERCISES

LAB 8 EXERCISE
PHP Loops

8.4.3 while and do . . . while

The `while` loop and the `do . . . while` loop are quite similar. Both will execute nested statements repeatedly as long as the `while` expression evaluates to `true`. In the `while`

loop, the condition is tested at the beginning of the loop; in the `do ... while` loop the condition is tested at the end of each iteration of the loop. Listing 8.10 provides examples of each type of loop.

```
$count = 0;
while ($count < 10)
{
   echo $count;
   $count++;
}

$count = 0;
do
{
   echo $count;
   $count++;
} while ($count < 10);
```

LISTING 8.10 while loops

8.4.4 for

The `for` loop in PHP has the same syntax as the `for` loop in JavaScript that we examined in Chapter 6. As can be seen in Listing 8.11, the `for` loop contains the same loop initialization, condition, and post-loop operations as in JavaScript.

```
for ($count=0; $count < 10; $count++)
{
   echo $count;
}
```

LISTING 8.11 for loops

There is another type of `for` loop: the `foreach` loop. This loop is especially useful for iterating through arrays and so this book will cover `foreach` loops in the array section of the next chapter.

8.4.5 Alternate Syntax for Control Structures

PHP has an alternative syntax for most of its control structures (namely, the `if`, `while`, `for`, `foreach`, and `switch` statements). In this alternate syntax (shown in Listing 8.12), the colon (`:`) replaces the opening curly bracket, while the closing

brace is replaced with `endif;`, `endwhile;`, `endfor;`, `endforeach;`, or `endswitch;`. While this may seem strange and unnecessary, it can actually improve the readability of your PHP code when it intermixes PHP and markup within a control structure, as was seen in Listing 8.8.

```
<?php if ($userStatus == "loggedin") :  ?>
   <a href="account.php">Account</a>
   <a href="logout.php">Logout</a>
<?php else : ?>
   <a href="login.php">Login</a>
   <a href="register.php">Register</a>
<?php endif; ?>
```

LISTING 8.12 Alternate syntax for control structures

8.4.6 Include Files

PHP does have one important facility that is generally unlike other nonweb programming languages, namely the ability to include or insert content from one file into another.[5] Almost every PHP page beyond simple practice exercises makes use of this include facility. Include files provide a mechanism for reusing both markup and PHP code, as shown in Figure 8.14.

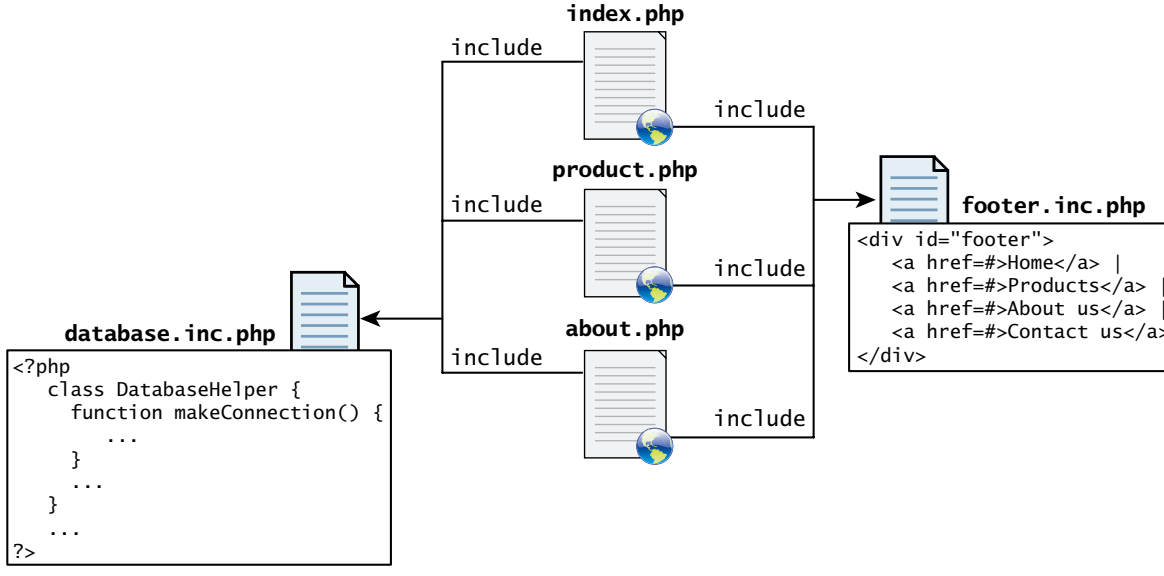


FIGURE 8.14 Include files

Older web development technologies also supported include files, and were typically called server-side includes (SSI). In a noncompiled environment such as PHP, include files are essentially the only way to achieve code and markup reuse.

PHP provides four different statements for including files, as shown below.

```
include "somefile.php";
include_once "somefile.php";

require "somefile.php";
require_once "somefile.php";
```

The difference between `include` and `require` lies in what happens when the specified file cannot be included (generally because it doesn't exist or the server doesn't have permission to access it). With `include`, a warning is displayed and then execution continues. With `require`, an error is displayed and execution stops. The `include_once` and `require_once` statements work just like `include` and `require` but if the requested file has already been included once, then it will not be included again. This might seem an unnecessary addition, but in a complex PHP application written by a team of developers, it can be difficult to keep track of if a given file has been included. It is not uncommon for a PHP page to include a file that includes other files that may include other files, and in such an environment the `include_once` and `require_once` statements are certainly recommended.

Scope within Include Files

Include files appear to provide a type of encapsulation, but it is important to realize that they are the equivalent of copying and pasting, though in this case it is performed by the server. This can be quite clearly seen by considering the scope of code within an include file. Variables defined within an include file will have the scope of the line on which the include occurs. Any variables available at that line in the calling file will be available within the called file. If the include occurs inside a function, then all of the code contained in the called file will behave as though it had been defined inside that function. Thus, for true encapsulation, you will have to use functions (covered next) and classes (covered in the next chapter).

8.5 Functions

Just as with any language, writing code in the main function (which in PHP is equivalent to coding in the markup between `<?php` and `?>` tags) is not a good habit to get into. Having all your code in the main body of a script makes it hard to reuse, maintain, and understand. As an alternative, PHP allows you to define functions. Just like with JavaScript, a function in PHP contains a small bit of code that accomplishes one thing. These functions can be made to behave differently based on the values of their parameters.

Functions can exist all on their own, and can then be called from anywhere that needs to make use of them, so long as they are in scope. Later you will write functions inside of classes, which we will call methods.

In PHP there are two types of function: user-defined functions and built-in functions. A **user-defined function** is one that you the programmer define. A **built-in function** is one of the functions that come with the PHP environment (or with one of its extensions). One of the real strengths of PHP is its rich library of built-in functions that you can use.

8.5.1 Function Syntax

To create a new function you must think of a name for it, and consider what it will do. Functions can return values to the caller, or not return a value. They can be set up to take or not take parameters. To illustrate function syntax, let us examine a function called `getNiceTime()`, which will return a formatted string containing the current server time, and is shown in Listing 8.13. You will notice that the definition requires the use of the `function` keyword followed by the function's name, round ( ) brackets for parameters, and then the body of the function inside curly { } brackets.[6]

```
/**
* This function returns a nicely formatted string using the current
* system time.
*/
function getNiceTime() {
   return date("H:i:s");
}
```

LISTING 8.13 The definition of a function to return the current time as a string

While the example function in Listing 8.13 returns a value, there is no requirement for this to be the case. Listing 8.14 illustrates a function definition that doesn't return a value but just performs a task.

```
/**
* This function outputs the footer menu
*/
function outputFooterMenu() {
   echo '<div id="footer">';
   echo '<a href=#>Home</a> | <a href=#>Products</a> | ';
   echo '<a href=#>About us</a> | <a href=#>Contact us</a>';
   echo '</div>';
}
```

LISTING 8.14 The definition of a function without a return value

8.5.2 Calling a Function

Now that you have defined a function, you are able to use it whenever you want to. To call a function you must use its name with the () brackets. Since `getNiceTime()` returns a string, you can assign that return value to a variable, or echo that return value directly, as shown below.

HANDS-ON EXERCISES

LAB 8 EXERCISE
Writing Functions

```
$output = getNiceTime();
echo getNiceTime();
```

If the function doesn't return a value, you can just call the function:

```
outputFooterMenu();
```

8.5.3 Parameters

It is more common to define functions with parameters, since functions are more powerful and reusable when their output depends on the input they get. Parameters are the mechanism by which values are passed into functions, and there are some complexities that allow us to have multiple parameters, default values, and to pass objects by reference instead of value.

To define a function with parameters, you must decide how many parameters you want to pass in, and in what order they will be passed. Each parameter must be named. To illustrate, let us write another version of `getNiceTime()` that takes an integer as a parameter to control whether to show seconds. You will call the parameter `showSeconds`, and write our function as shown in Listing 8.15. Notice that parameters, being a type of variable, must be prefaced with a $ symbol like any other PHP variable.

```
/**
* This function returns a nicely formatted string using the current
* system time. The showSeconds parameter controls whether or not to
* include the seconds in the returned string.
*/
function getNiceTime($showSeconds) {
  if ($showSeconds==true)
    return date("H:i:s");
  else
    return date("H:i");
}
```

LISTING 8.15 A function to return the current time as a string with an integer parameter

Thus to call our function, you can now do it in two ways:

```
echo getNiceTime(1);    // this will print seconds
echo getNiceTime(0);   //  will not print seconds
```

In fact any nonzero number passed in to the function will be interpreted as `true` since the parameter is not type specific.

NOTE

Now you may be asking how you can that use the same function name that you used before. Well, to be honest, we are replacing the old function definition with this one. If you are familiar with other programming languages, you might wonder whether we couldn't overload the function, that is, define a new version with a different set of input parameters.

In PHP, the signature of a function is based on its name, and not its parameters. Thus it is **not** possible to do the same function overloading as in other object-oriented languages. PHP does have class method overloading, but it means something quite different than in other object-oriented languages.

Parameter Default Values

You may wonder if you could not simply combine the two overloaded functions together into one so that if you call it with no parameter, it uses a default value. The answer is yes you can!

In PHP you can set parameter default values for any parameter in a function. However, once you start having default values, all subsequent parameters must also have defaults. Applying this principle, you can combine our two functions from Listing 8.13 and Listing 8.15 together by adding a default value in the parameter definition as shown in Listing 8.16.

```
/**
* This function returns a nicely formatted string using the current
* system time. The showSeconds parameter controls whether or not
* to show the seconds.
*/
function getNiceTime($showSeconds=1){
   if ($showSeconds==true)
      return date("H:i:s");
   else
      return date("H:i");
}
```

LISTING 8.16 A function to return the current time with a parameter that includes a default

Now if you were to call the function with no values, the `$showSeconds` parameter would take on the default value, which we have set to 1, and return the string with seconds. If you do include a value in your function call, the default will be overridden by whatever that value was. Either way you now have a single function that can be called with or without values passed.

Passing Parameters by Reference

By default, arguments passed to functions are **passed by value** in PHP. This means that PHP passes a copy of the variable so if the parameter is modified within the function, it does not change the original. Listing 8.17 illustrates a simple example of passing by value. Notice that even though the function modifies the parameter value, the contents of the variable passed to the function remain unchanged after the function has been called.

```
function changeParameter($arg) {
   $arg += 300;
   echo "<br/>arg=" . $arg;
}

$initial = 15;
echo "<br/>initial=" . $initial;   // output: initial=15
changeParameter($initial);         // output: arg=315
echo "<br/>initial=" . $initial;   // output: initial=15
```

LISTING 8.17 Passing a parameter by value

Like many other programming languages, PHP also allows arguments to functions to be **passed by reference**, which will allow a function to change the contents of a passed variable. A parameter passed by reference points the local variable to the same place as the original, so if the function changes it, the original variable is changed as well. The mechanism in PHP to specify that a parameter is passed by reference is to add an ampersand (&) symbol next to the parameter name in the function declaration. Listing 8.18 illustrates an example of passing by reference.

```
function changeParameter(&$arg) {
   $arg += 300;
   echo "<br/>arg=". $arg;
}

$initial = 15;
echo "<br/>initial=" . $initial;   // output: initial=15
changeParameter($initial);         // output: arg=315
echo "<br/>initial=" . $initial;   // output: initial=315
```

LISTING 8.18 Passing a parameter by reference

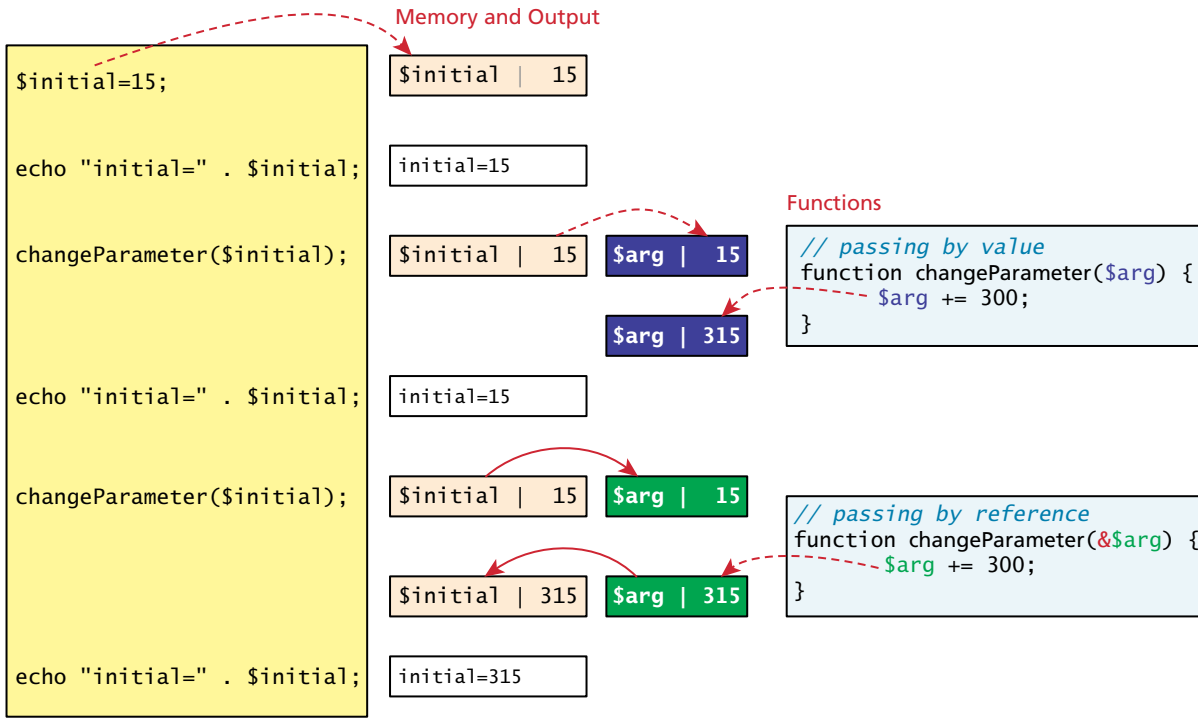


FIGURE 8.15 Pass by value versus pass by reference

Figure 8.15 illustrates visually the memory differences between pass-by value and pass-by reference.

The possibilities opened up by the pass-by reference mechanism are significant, since you can now decide whether to have your function use a local copy of a variable, or modify the original. By and large, you will likely find that most of the time you will use pass-by value in the majority of your functions. When we introduce classes and methods, we will come back to this particular issue again.

8.5.4 Variable Scope within Functions

It will come as no surprise that all variables defined within a function (such as parameter variables) have function scope, meaning that they are only accessible within the function. It might be surprising though to learn that any variables created outside of the function in the main script are unavailable within a function. For instance, in the following example, the output of the `echo` within the function is 0 and not 56 since the reference to `$count` within the function is assumed to be a new variable named `$count` with function scope.

```
$count= 56;

function testScope() {
   echo $count;     // outputs 0 or generates run-time warning/error
}
testScope();
echo $count;       // outputs 56
```

While variables defined in the main script are said to have **global scope**, unlike in other programming languages, a global variable is not, by default, available within functions. Of course, in the above example, one could simply have passed `$count` to the function. However, there are times when such a strategy is unworkable. For instance, most web applications will have important data values such as connections, application constants, and logging/debugging switches that need to be available throughout the application, and passing them to every function that might need them is often impractical. This is actually a tricky design problem that we will return to in Chapter 9, but PHP does allow variables with global scope to be accessed within a function using the `global` keyword, as shown in Listing 8.19.

```
$count= 56;

function testScope() {
   global $count;
   echo $count;    // outputs 56
}

testScope();
echo $count;       // outputs 56
```

LISTING 8.19 Using the global keyword

From a programming design standpoint, the use of global variables should be minimized, and only used for vital application objects that are truly global.

PRO TIP

There is in fact another way to have global variables, which is the preferred mechanism for using globals in PHP. In the next chapter you will learn about the superglobal variables in PHP, which are used for accessing query string data, server data, and session storage. One of these is the `$GLOBALS` associative array, which is always available and is a convenient storage location for any data that must be available globally.

8.6 Chapter Summary

In this chapter we have covered two key aspects of server-side development in PHP. We began by exploring what server-side development is in general in the context of the LAMP software stack. The latter half of the chapter focused on introductory PHP syntax, covering all the core programming concepts including variables, functions, and program flow.

8.6.1 Key Terms

ASP
ASP.NET
built-in function
Common Gateway Interface (CGI)
constant
daemon
data storage
data types
database
database management system (DBMS)
dynamically typed
extension layer
fork
function
function scope
global scope
handlers
Java Server Pages (JSP)
loosely typed
module
multi-process
multi-threaded
opcodes
overloading
parameters
parameter default values
passed by reference
passed by value
Perl
PHP
PHP core
preforked
process
Python
Ruby On Rails
SAPI
script
server-side includes (SSI)
thread
user-defined function
virtual machine
web services
worker
Zend Engine

8.6.2 Review Questions

1. In the LAMP stack, what software is responsible for responding to HTTP requests?
2. Describe one alternative to the LAMP stack.
3. Identify and briefly describe at least four different server-side development technologies.
4. Describe the difference between the multi-threaded and multi-process setup of PHP in Apache.
5. Describe the steps taken by the Zend Engine when it receives a PHP request.
6. What does it mean that PHP is dynamically typed?
7. What are server-side include files? Why are they important in PHP?
8. Can we have two functions with the same name in PHP? Why or why not?
9. How do we define default function parameters in PHP?
10. How are parameters passed by reference different than those passed by value?

8.6.3 Hands-On Practice

PROJECT 1: Book Rep Customer Relations Management

DIFFICULTY LEVEL: Beginner

HANDS-ON EXERCISES
PROJECT 8.1

Overview

Demonstrate your ability to work with PHP by converting Chapter08-project01.html into a PHP file that looks similar to that shown in Figure 8.16.

Instructions

1. You have been provided with an HTML file (Chapter08-project01.html) that includes all the necessary markup. Save this file as Chapter08-project01.php.
2. Use the PHP `include()` function to include the file book-data.php. This file sets the values of two variables: `$email` and `$password`.
3. Use a `for` loop to output the `<option>` elements (see Figure 8.16).
4. Use an `if...else` statement to display an error message and add the `has-error` CSS class to the appropriate `<div class="form-group">` element if the `$email` variable is empty.
5. Do the same thing for the `$password` variable.

Test

1. Test the page. Remember that you cannot simply open a local PHP page in the browser using its open command. Instead you must have the browser request the page from a server. If you are using a local server such as XAMMP, the file must exist within the `htdocs` folder of the server, and then the request will be localhost/some-path/chapter08-project01.php.
2. Verify that the logic works by editing the values of the two variables in the book-data.php file.

PROJECT 2: Art Store

DIFFICULTY LEVEL: Intermediate

HANDS-ON EXERCISES
PROJECT 8.2

Overview

Demonstrate your ability to work with PHP by converting Chapter08-project02.html into a PHP file that looks similar to that shown in Figure 8.17.

Instructions

1. You have been provided with an HTML file (Chapter08-project02.html) that includes all the necessary markup. Save this file as Chapter08-project02.php.
2. Move the header and footer into two separate include files named art-header.inc.php and art-footer.inc.php. Use the PHP `include()` function to include each of these files back into the original file.
3. Use a `for` loop to output the special list (see Figure 8.17).

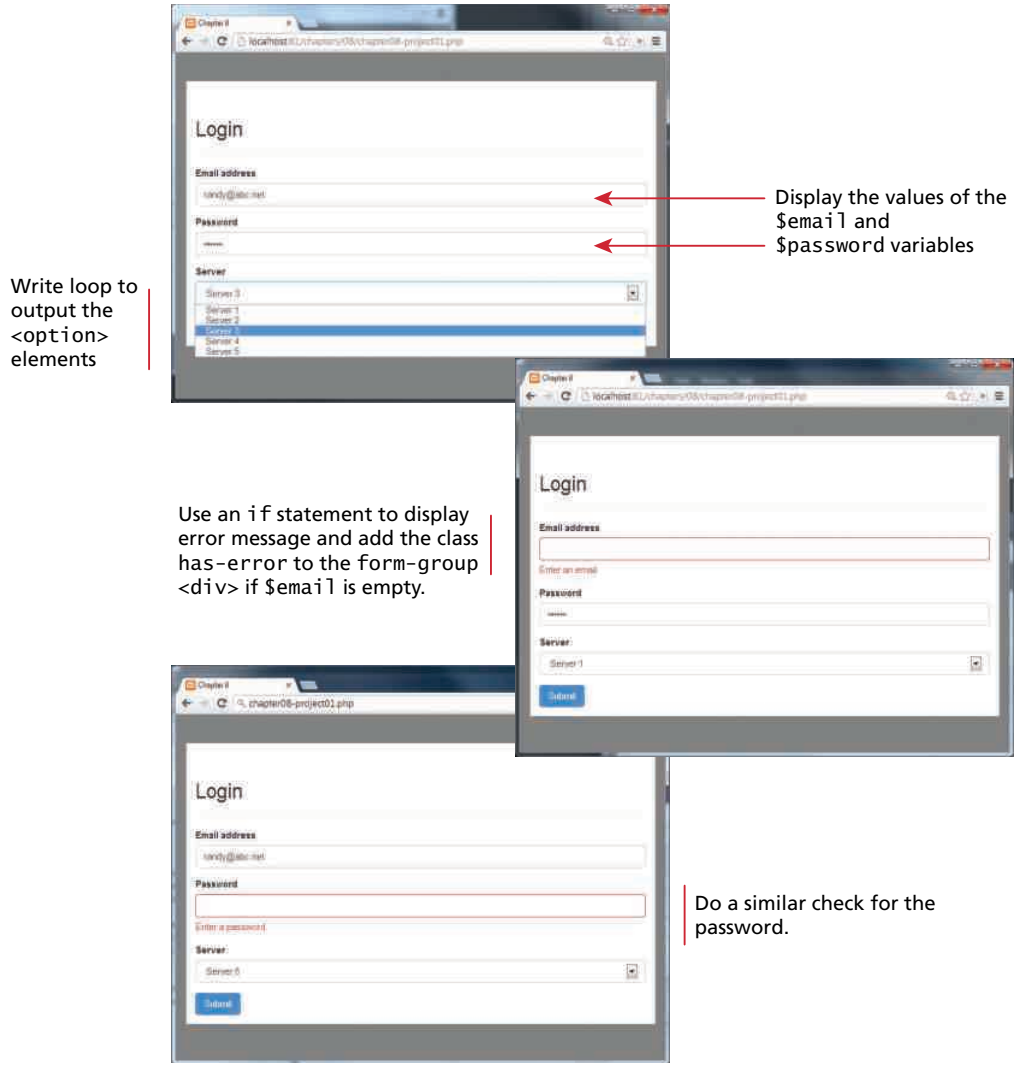


FIGURE 8.16 Completed Project 1

4. Create a function called `outputCartRow()` that has the following signature:

```
function outputCartRow($file, $product, $quantity, $price) { }
```

5. Implement the body of the `outputCartRow()` function. It should echo the passed information as a table row. Use the `number_format()` function to format the currency values with two decimal places. Calculate the value for the amount column.

6. Replace the two cart table rows in the original with the following calls:

```
outputCartRow($file1, $product1, $quantity1, $price1);

outputCartRow($file2, $product2, $quantity2, $price2);
```

7. The above variables are defined in the file art-data.php. You will need to include this file.
8. Calculate the subtotal, tax, shipping, and grand total using PHP. Replace the hard-coded values with your variables that contain the calculations. Use 10 percent as the tax amount. The shipping value will be $100 unless the subtotal is above $2000, in which case it will be $0.

Test

1. Test the page in the browser (see the test section of the previous section to remind yourself about how to do this). Verify that the calculations work appropriately by changing the values in the art-data.php file.

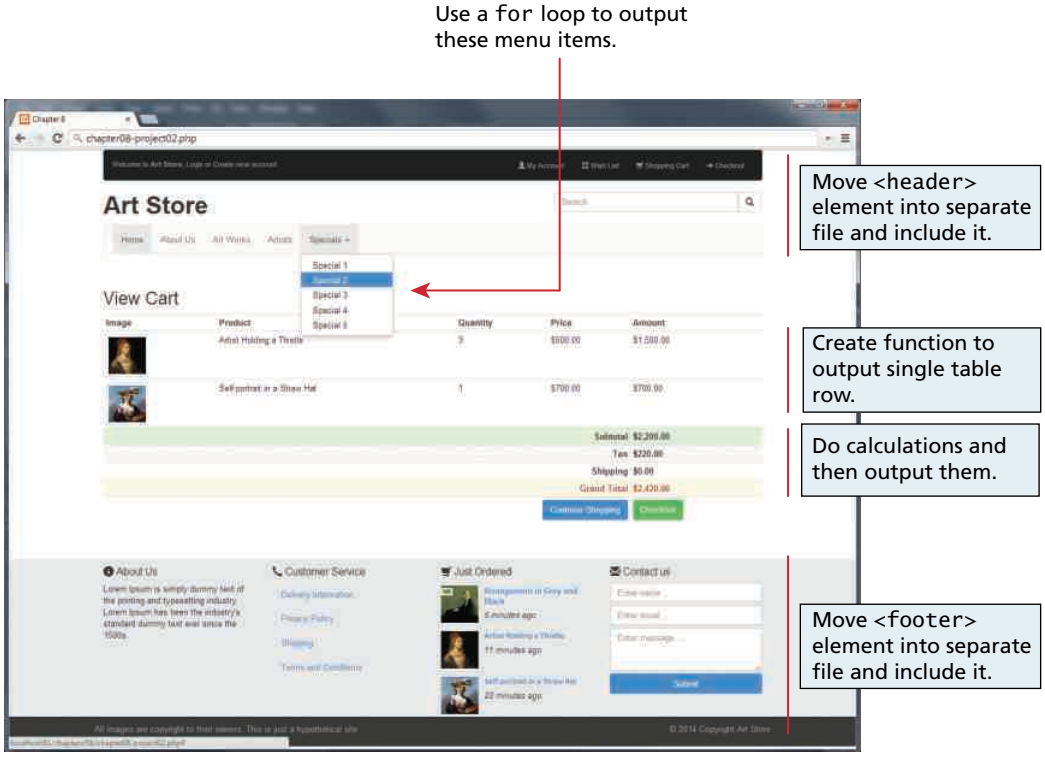


FIGURE 8.17 Completed Project 2

PROJECT 3: Share Your Travel Photos

DIFFICULTY LEVEL: Advanced

HANDS-ON EXERCISES
PROJECT 8.3

Overview

Demonstrate your ability to work with PHP by creating PHP functions and include files so that Chapter08-project03.php looks similar to that shown in Figure 8.18.

Instructions

1. You have been provided with a PHP file (Chapter08-project03.php) that includes all the necessary markup. Move the header, footer, and left navigation boxes into three separate include files. Use the PHP `include()` function to include each of these files back into the original file.
2. Create a function called `generateLink()` that takes three arguments: `$url`, `$label`, and `$class`, which will echo a properly formed hyperlink in the following form:

```
<a href="$url"
class="$class">$label</a>
```

3. Create a function called `outputPostRow()` that takes a single argument: `$number`. This function will echo the necessary markup for a single post. For it to work, you will need to include a file called travel-data.php. (Hint: remember PHP's scope rules). This is a provided file that defines variables containing the post data for all three posts. Your function will need to use dynamic variable names. Be sure to also use your `generateLink()` function for the three links (image, user name, read more) in each post. Notice that these links contain query strings making use of the `userId` or `postId`.
4. Two of the user names contain special characters. Be sure to use the `utf8_encode()` function.
5. Remove the existing post markup and replace with calls to `outputPostRow()`, for instance: `outputPostRow(1);`
6. Move the pagination markup into a function called `ouputPagination()` that takes two parameters: `$startNum`, `$currentNum`. This function will use a loop to output ten page numbers that start with the value provided in `$startNum`. For the page number indicated by `$currentNum`, the function must add the `class="active"` attribute to the appropriate `<li>` element. Finally, the function should add the `class="disabled"` attribute to the first `<li>` element if the `$startNum` is less than or equal to ten.

Test

1. Test the page in the browser. Verify that `ouputPagination()` works appropriately with different `$startNum` and `$currentNum` parameters.

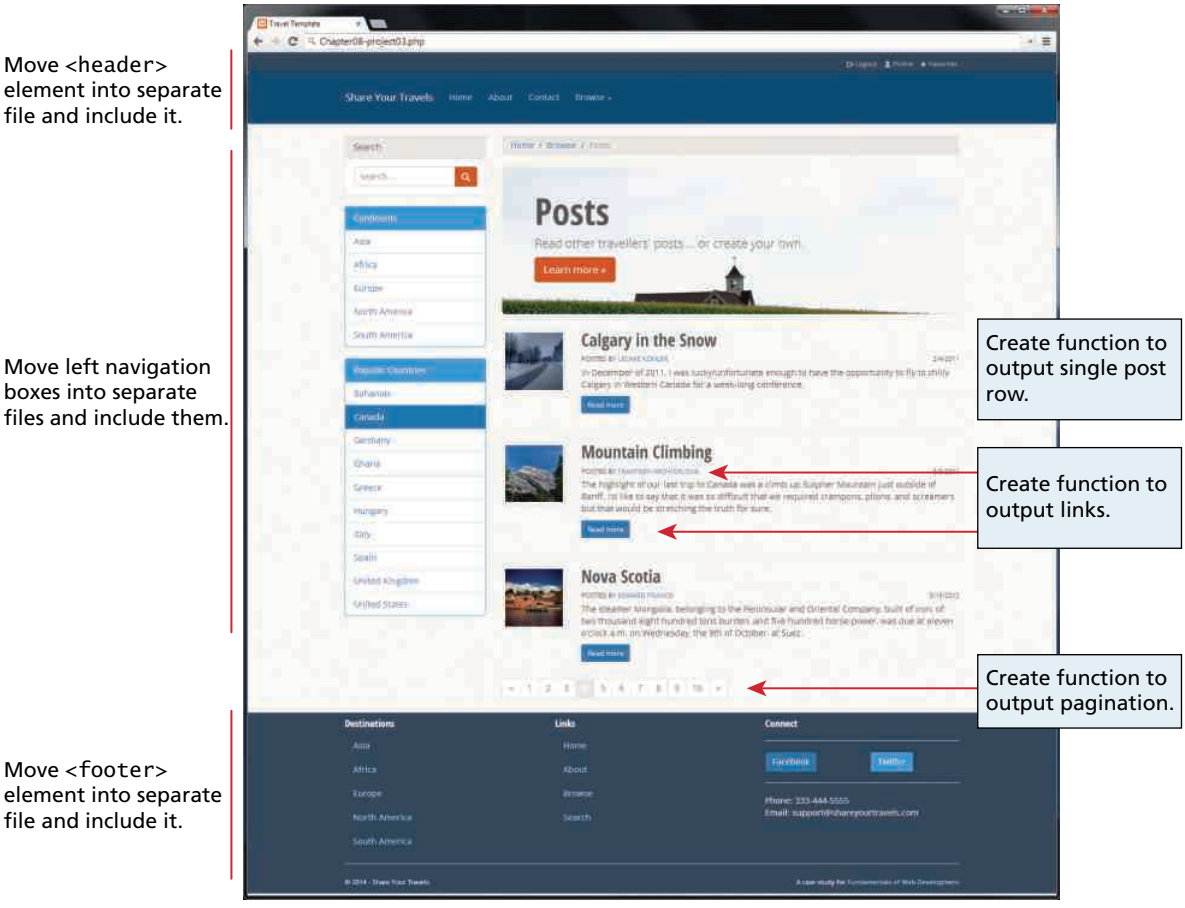


FIGURE 8.18 Completed Project 3

8.6.4 References

1. L. Welling and L. Thomson, *PHP and MySQL Web Development*, 5th ed., Addison-Wesley Professional, 2013.
2. PHP. [Online]. http://php.net/manual/en/language.oop5.basic.php.
3. M. Doyle, *Beginning PHP 5.3*, Wrox, 2009.
4. PHP, "printf." [Online]. http://ca2.php.net/manual/en/function.printf.php.
5. PHP, "include." [Online]. http://ca2.php.net/manual/en/function.include.php.
6. PHP, "Functions." [Online]. http://ca2.php.net/manual/en/language.functions.php.