MODULE 3

JAVASCRIPT: CLIENT-SIDE SCRIPTING

SYLLABUS

JavaScript: Client-Side Scripting, What is JavaScript and What can it do?, JavaScript Design Principles, where does JavaScript Go?, Syntax, JavaScript Objects, The Document Object Model (DOM), JavaScript Events, Forms, Introduction to Server-Side Development with PHP, What is Server-Side Development, A Web Server's Responsibilities, Quick Tour of PHP, Program Control, Functions

6.1 WHAT IS JAVASCRIPT AND WHAT CAN IT DO?

Larry Ullman defines as JavaScript: an object-oriented, dynamically typed, scripting language.

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

6.1.1 CLIENT-SIDE SCRIPTING

- 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.
- Example - Flash, VBScript, Java, and JavaScript.
- Some of these technologies only work in certain browsers, while others require plug-ins to function.
- JavaScript is browser interoperability (that is, its ability to work/operate on most browsers).

ADVANTAGE 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.
- 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.

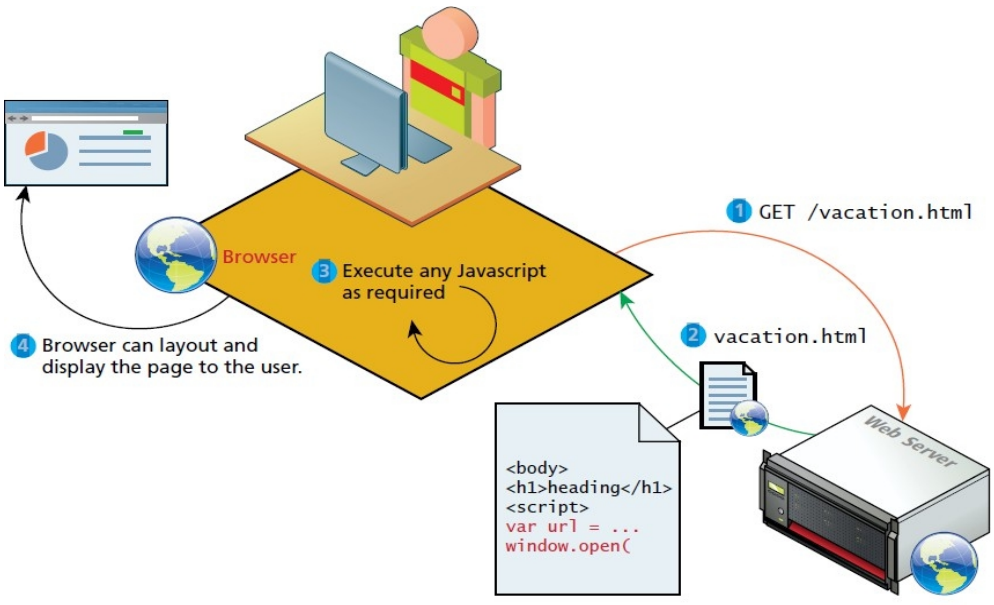

FIGURE 6.1 Downloading and executing a client-side JavaScript script

DISADVANTAGE OF CLIENT-SIDE SCRIPTING

1. 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.
2. 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.
3. 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. It has the distinct disadvantage of blending HTML and JavaScript together, which decreases code readability, and increases the difficulty of web development.

- There are two other noteworthy client-side approaches to web programming.
    1. 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.
        (a) Flash objects 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
        (b) 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.

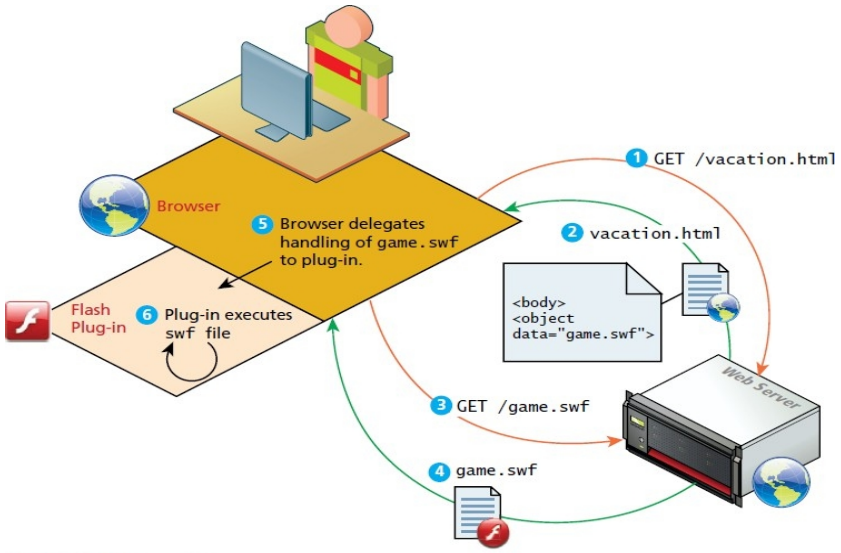

FIGURE 6.2 Adobe Flash

(2) Java applets – An applet is a term that refers to a small application that performs a relatively small task.

(a) 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.

(b) 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.

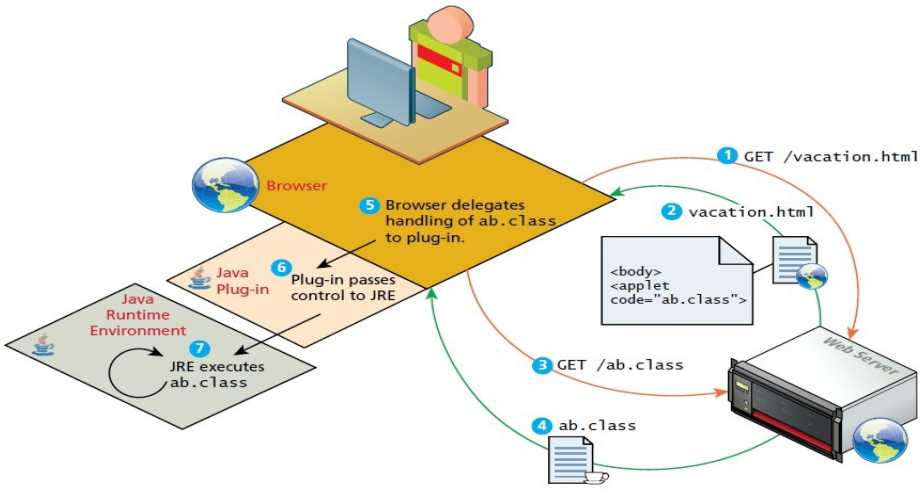

FIGURE 6.3 Java applets

DISADVANTAGE OF FLASH AND APPLETS

1) 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).
2) Flash and Java applets also require frequent updates, which can annoy the user and present security risks.
3) With the universal adoption of JavaScript and HTML5, JavaScript remains the most dynamic and important client-side scripting language for the modern web developer.

6.1.2 JAVASCRIPT'S HISTORY AND USES

- JavaScript was introduced by Netscape in their Navigator browser back in 1996. It originally was called Live Script, 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. The current version for JavaScript at the time of writing is 1.8.5.
- In the middle of the 2000s with the emergence of so-called AJAX (Asynchronous JavaScript and XML) sites that JavaScript became a much more important part of web development. AJAX was accurate for some time; but since XML is no longer always the data format for data transport in AJAX sites.
- 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 in 1999, but it wasn't until sophisticated websites by Google 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.
- 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.
- 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 each request is invoking a server-side script.

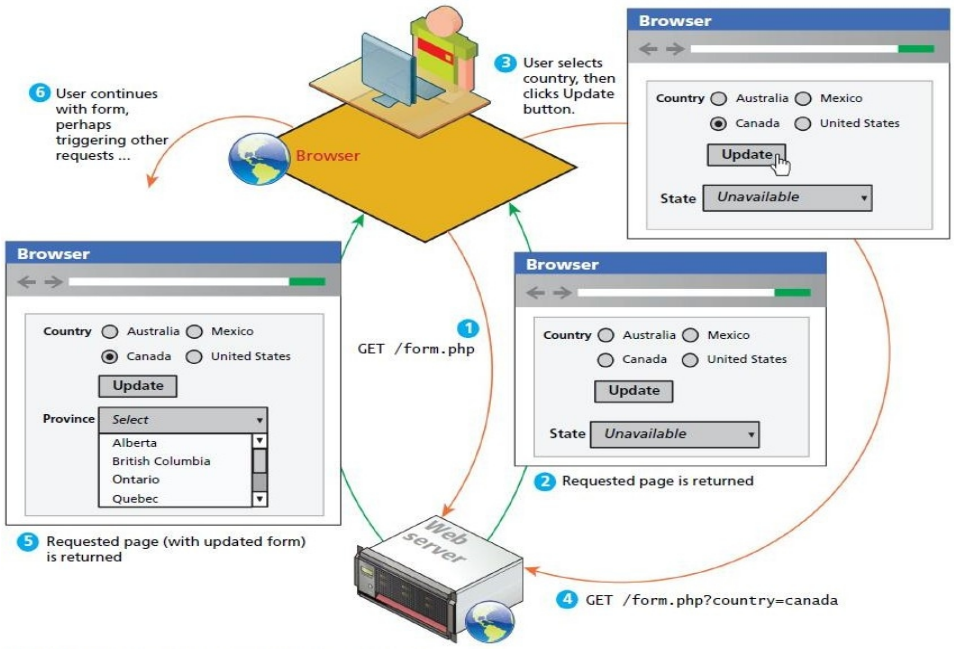

FIGURE 6.4 Normal HTTP request-response loop

- 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.

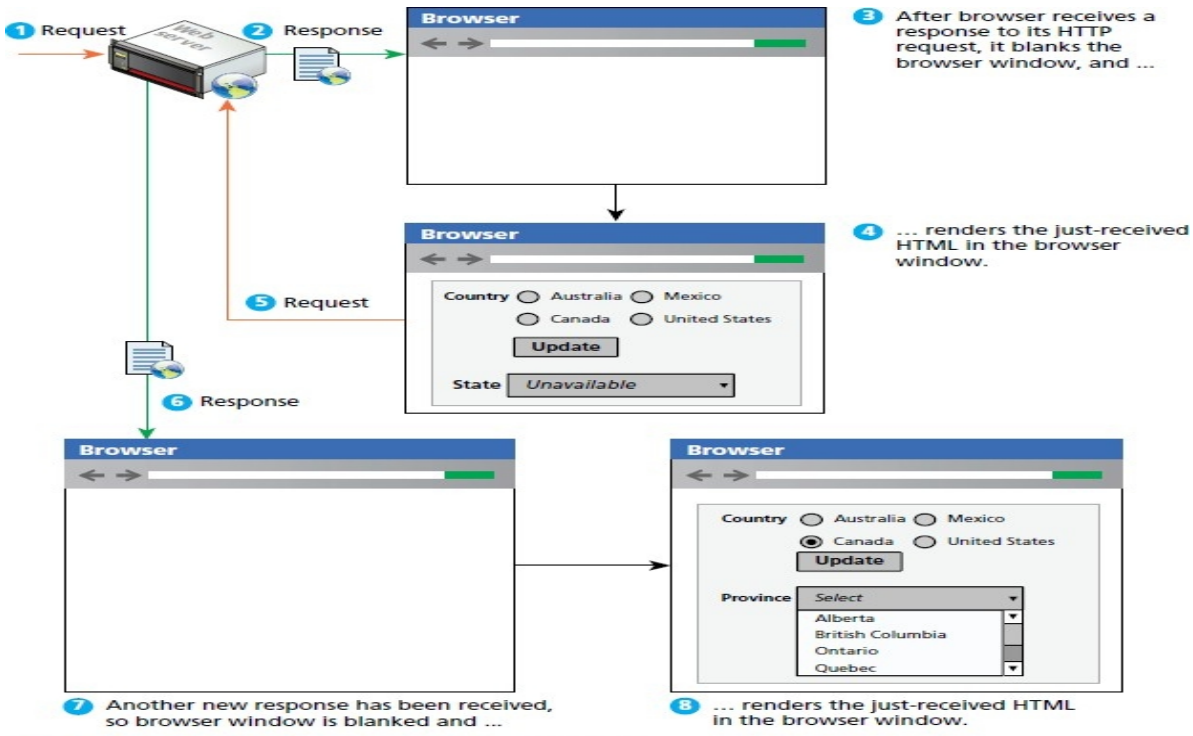

FIGURE 6.5 Normal HTTP request-response loop, take two

- 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 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.
- 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 from the view design pattern.

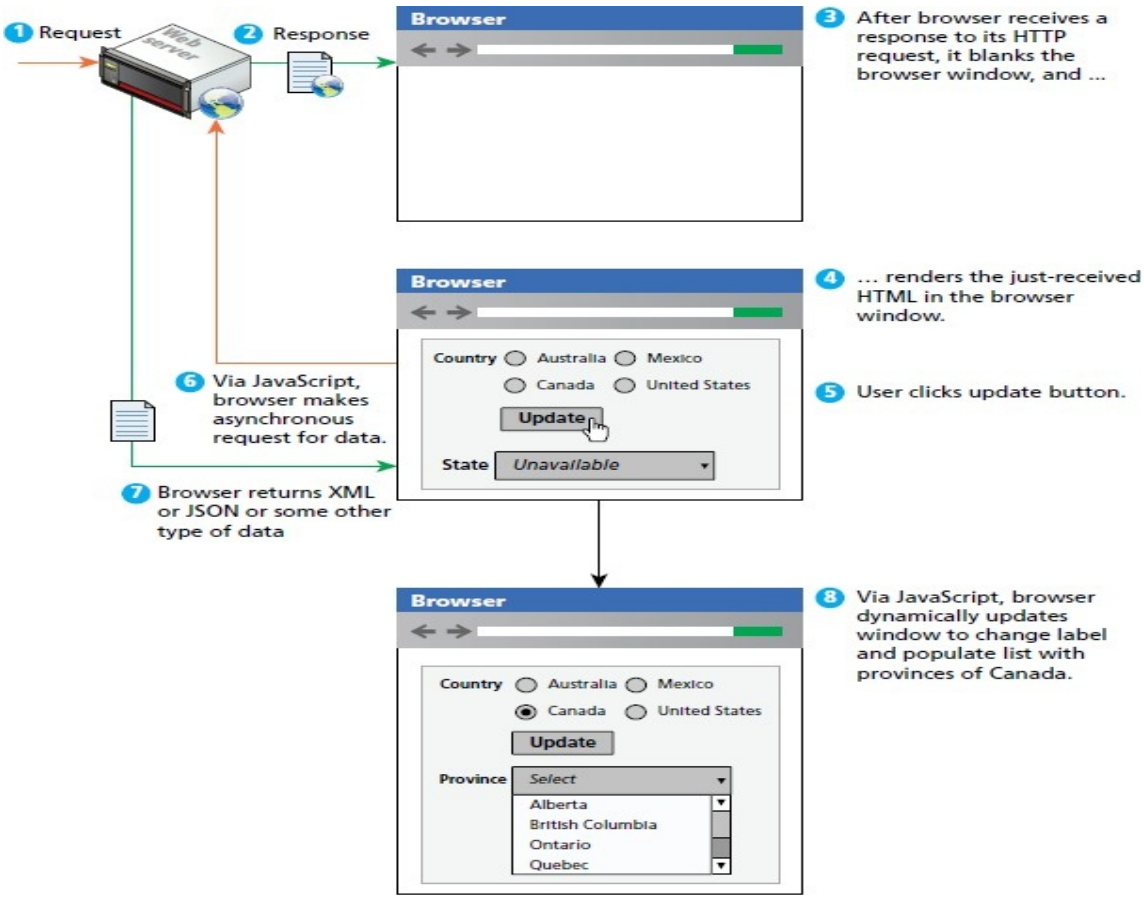

FIGURE 6.6 Asynchronous data requests

FIGURE 6.7 Example jQuery plug-ins

6.2 JAVASCRIPT DESIGN PRINCIPLES

6.2.1 LAYERS

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.

These layers have different capabilities and responsibilities, but are always considered optional. Although each layer can perform many tasks, it is helpful to visualize and understand the types of conceptual layers that are common.

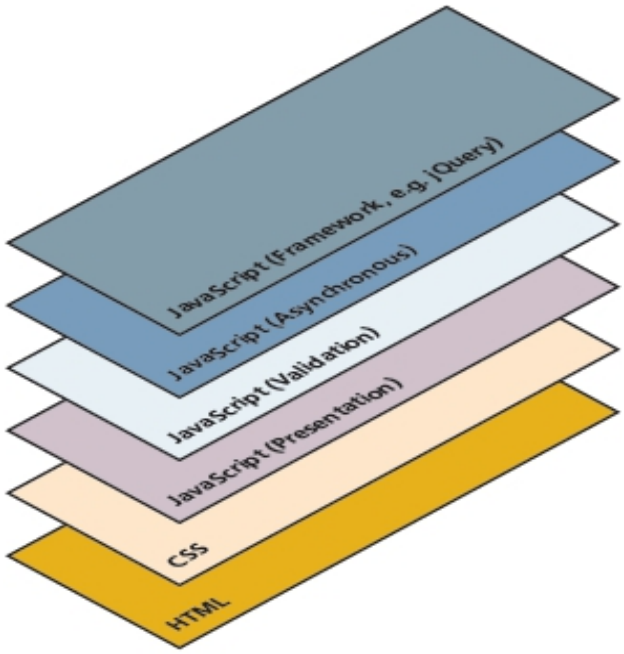

FIGURE 6.8 JavaScript layers

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.
- For example, validating a form to make sure the email entered is valid before sending it.

- It is often used in conjunction with the presentation layer 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 update a portion of the page. Asynchronous layers are considered advanced versions of the presentation and validation layers above.

6.2.2 USERS WITHOUT JAVASCRIPT

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. 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.
- **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.

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.

FAIL-SAFE DESIGN: 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 popup 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).

6.2.3 GRACEFUL DEGRADATION AND PROGRESSIVE ENHANCEMENT

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.

- 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, One 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).
- 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. 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.

6.3 WHERE DOES JAVASCRIPT GO?

- JavaScript can be linked to an HTML as 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.

6.3.1 INLINE JAVASCRIPT

Inline JavaScript refers to the practice of including JavaScript code directly within certain HTML attributes. However, inline JavaScript is not a good practice.

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

LISTING 6.1 Inline JavaScript example

6.3.2 EMBEDDED JAVASCRIPT

Embedded JavaScript refers to the practice of placing JavaScript code within a <script> Element.

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

LISTING 6.2 Embedded JavaScript example

6.3.3 EXTERNAL JAVASCRIPT

JavaScript supports links to an external file that contains the JavaScript. JavaScript external files have the extension .js. Modern websites often have links to several, 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

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.

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.

6.4 SYNTAX

Problems encountered by JavaScript developers:

- 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.

6.4.1 VARIABLES

- **Variables** in JavaScript are **dynamically typed.**
- To declare a variable x, we use the var keyword, the name, and a semicolon If we specify no value, then 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.
- The **conditional assignment** operator, can also be used to assign based on condition

```
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

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

FIGURE 6.14 The conditional assignment operator

6.4.2 COMPARISON OPERATORS

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

6.4.3 LOGICAL OPERATORS

Syntactically logical operators are represented with && (and), || (or), and ! (Not).

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

6.4.4 CONDITIONALS

Conditionals statement in JavaScript are **if and else if**. 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.

```
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

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

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

6.4.6 FUNCTIONS

- **Functions** are the building block for modular code in JavaScript, and are even used to build **pseudo-classes**.
- 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);
```

ALERT

The alert() function makes the browser show a pop-up to the user, with whatever is passed being the message displayed. The pop-up obscures the underlying web page, and no actions can be done until the pop-up is dismissed.

The following JavaScript code displays a simple hello world message in a pop-up:

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

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.**

```
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.

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

LISTING 6.6 Throwing a user-defined exception

6.5 JAVASCRIPT OBJECTS

JavaScript is not a full-fledged object-oriented programming language. It does not have classes 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.

Objects can have **constructors**, **properties**, and **methods** associated with them. There are objects that are included in the JavaScript language; we can also define our own kind of objects.

6.5.1 CONSTRUCTORS

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");
```

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
```

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

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. The following code creates a new, empty array named greetings:

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

To **initialize** the array with values,

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

or, using the square bracket notation:

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

Accessing and Traversing an Array

To access an element in the we use square bracket notation from with the index we 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]);
}
```

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

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

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().

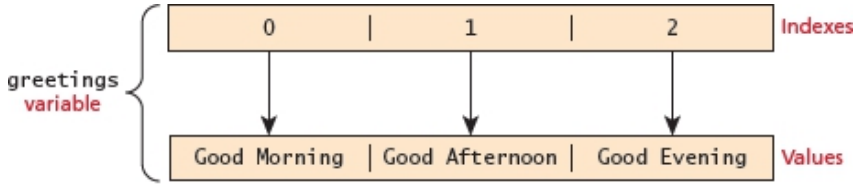

FIGURE 6.16 JavaScript array with indexes and values illustrated

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.

```
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. 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.

```
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

Date 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).

The tree is formally called the **DOM Tree** with the root, or topmost object called the **Document Root**.

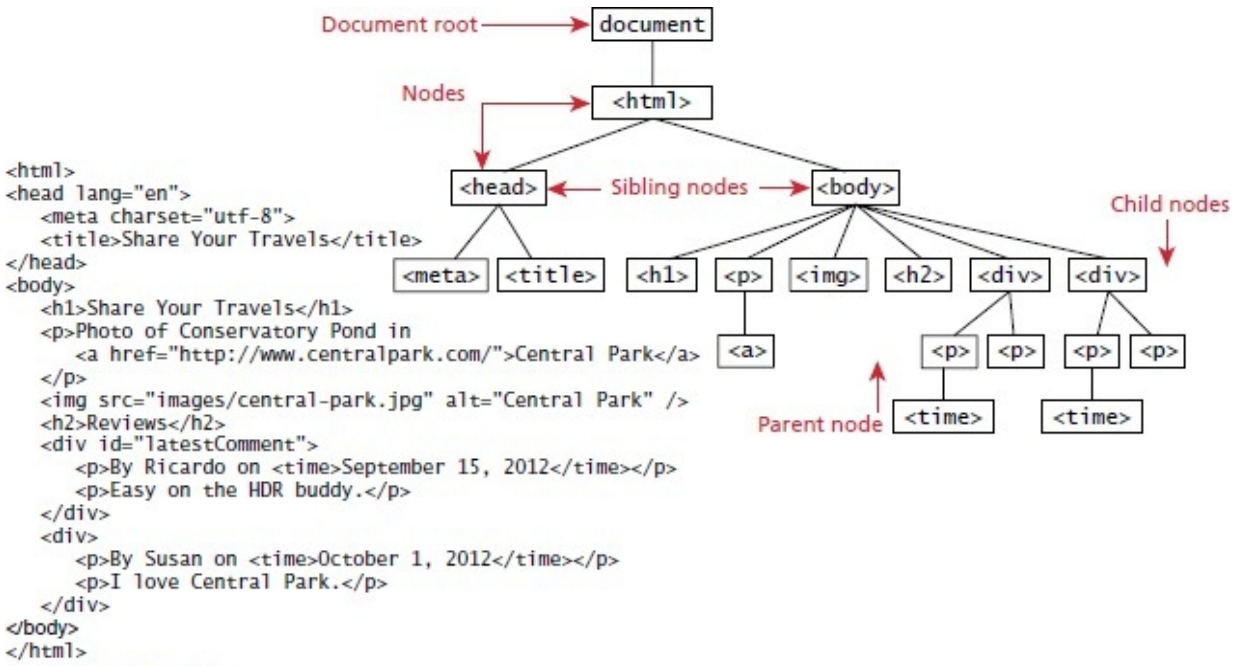

FIGURE 6.17 DOM tree

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.

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.

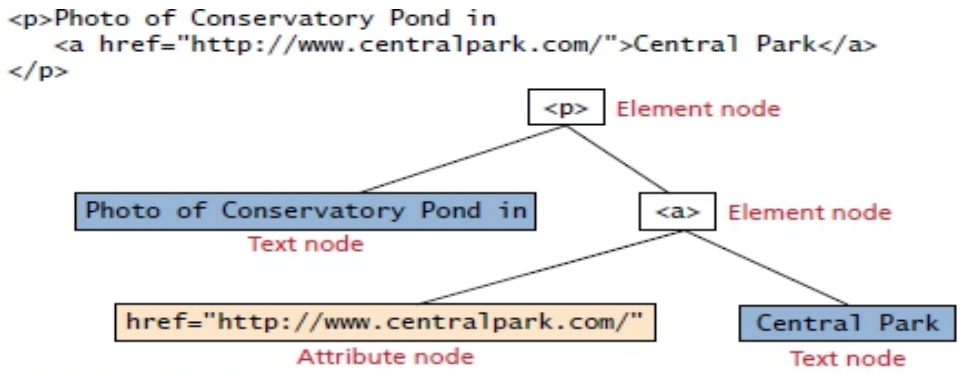

FIGURE 6.18 DOM nodes

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.
- There are some essential methods that we 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

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;
```

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

6.6.3 ELEMENT NODE OBJECT

- The type of object returned by the method document.getElementById() 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.
- 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.

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

6.6.4 MODIFYING A DOM ELEMENT

The document.write() method is used to create output to the HTML page from JavaScript. While this is certainly valid, it 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

```
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, 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 create-TextNode(), removeChild(), and appendChild() allow us to modify an element in a more rigorous way

```
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";
```

Note that the style property is itself an object, specifically a CSS Style Declaration type, which includes all the CSS attributes as properties and computes the current style from inline, external, and embedded styles.

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.

6.7.3 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. These properties are listed below

- **Bubble:** 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. 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.

```
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

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. Many mouse events can be sent at a time. The user could be moving the mouse off one 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.

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

Keyboard Events

Keyboard events are often overlooked by novice web developers, but are important tools for power users. 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

Form Events

Forms are the main means by which user input is collected and transmitted to the server. 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 preventDefault() and alert the user. Otherwise we do nothing, which allows the default event to happen.

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

```
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

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.

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

```
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

6.8 FORMS

6.8.1 VALIDATING FORMS

- Form validation is one of the most common applications of JavaScript.
- Writing code to pre-validate 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 pre-validation 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 and allow for more complete and concise scripts to validate particular fields.

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. 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.

```
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

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.
- One 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(), one 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()

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

LISTING 6.19 A function to test for a numeric value

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.
- 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.

8. INTRODUCTION TO SERVER-SIDE DEVELOPMENT WITH PHP

8.1 WHAT IS SERVER-SIDE DEVELOPMENT?

It involves the use of a programming technology like PHP or ASP.NET to create scripts that dynamically generate content.

8.1.1 COMPARING CLIENT AND SERVER SCRIPTS

- 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.
- 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.

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
- 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.

- 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 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.
- 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.

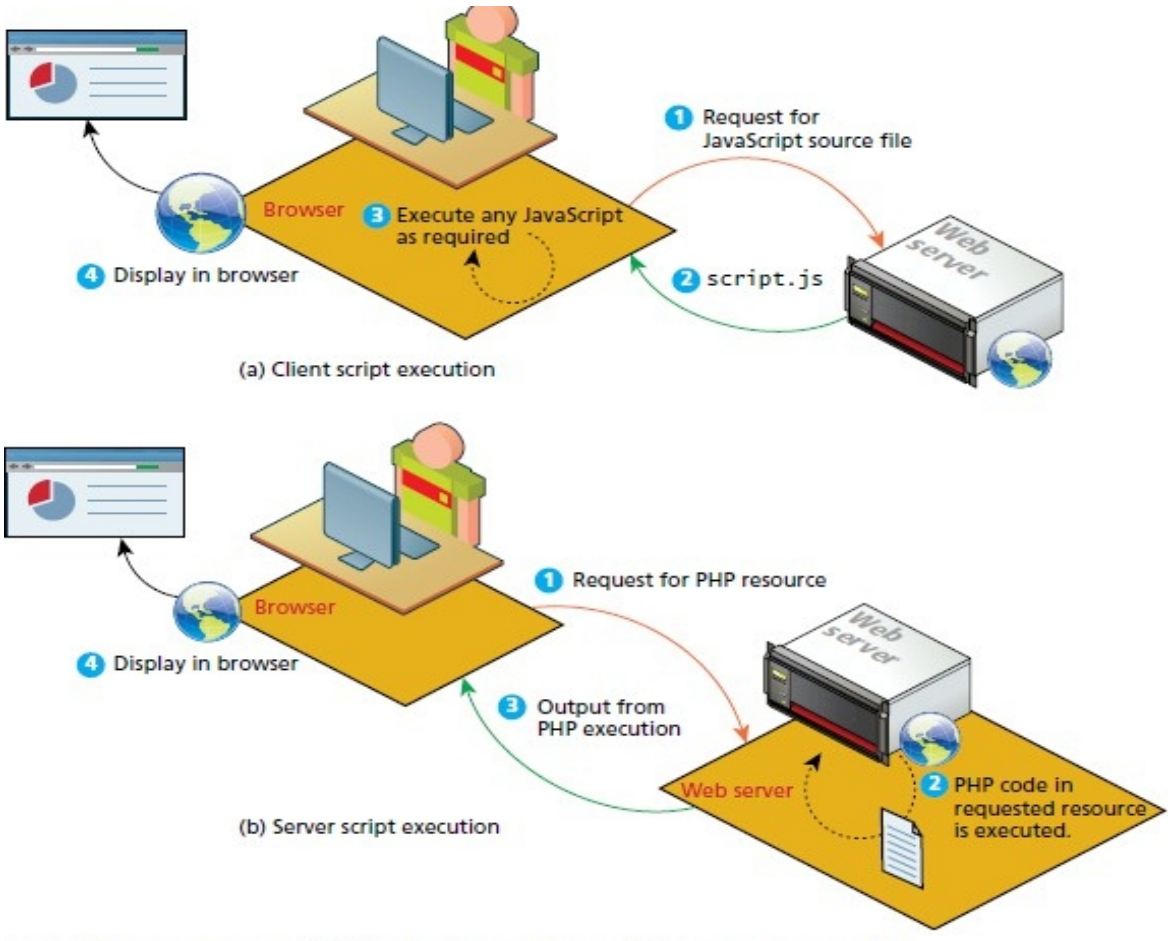

FIGURE 8.1 Comparison of (a) client script execution and (b) server script execution

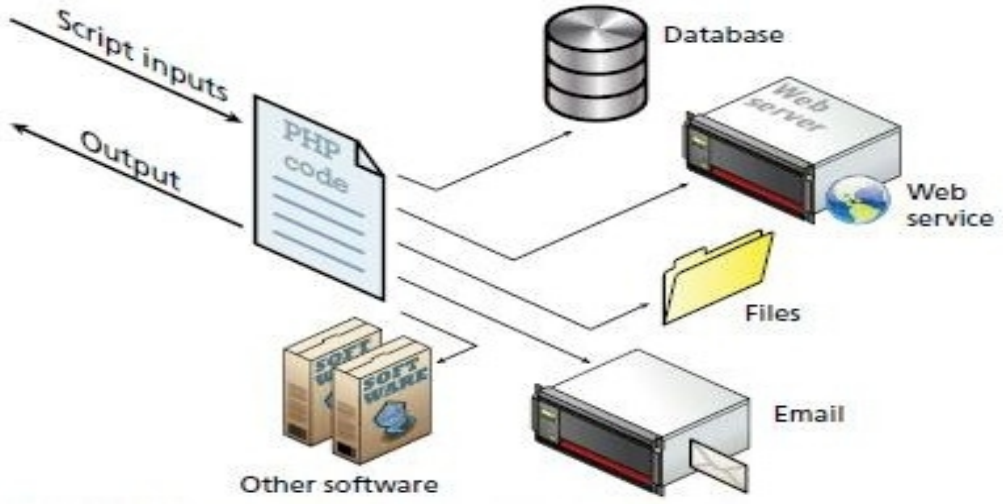

FIGURE 8.2 Server scripts have access to many resources.

8.1.3 COMPARING SERVER-SIDE TECHNOLOGIES

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**. Like PHP, ASP code 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. 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. 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.

■ **Python**. This 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.

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.

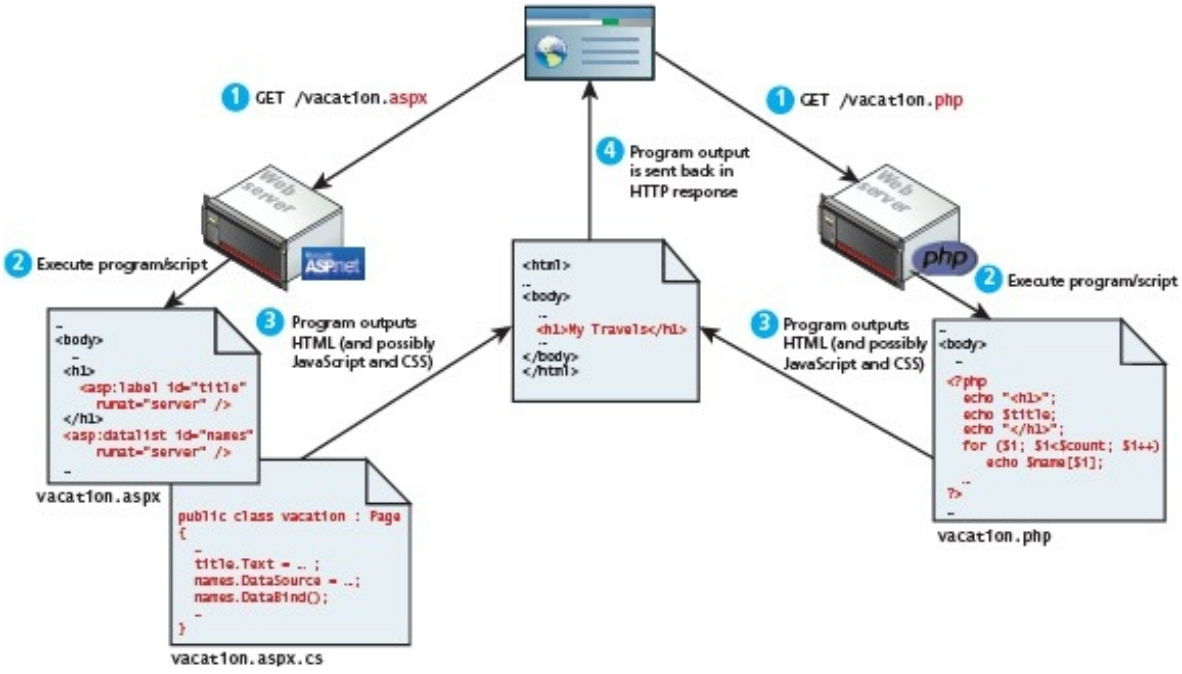

FIGURE 8.3 Web development technologies

8.2 A WEB SERVER'S RESPONSIBILITIES

- 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, 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.

8.2.1 APACHE AND LINUX

- 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 that runs in the background, waiting for a specific event that will activate it.
- As a background process, the Apache daemon waits for incoming HTTP requests. When a request arrives, Apache then uses modules to determine how to respond to the request.
- In Apache, a **module** is a compiled extension 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.

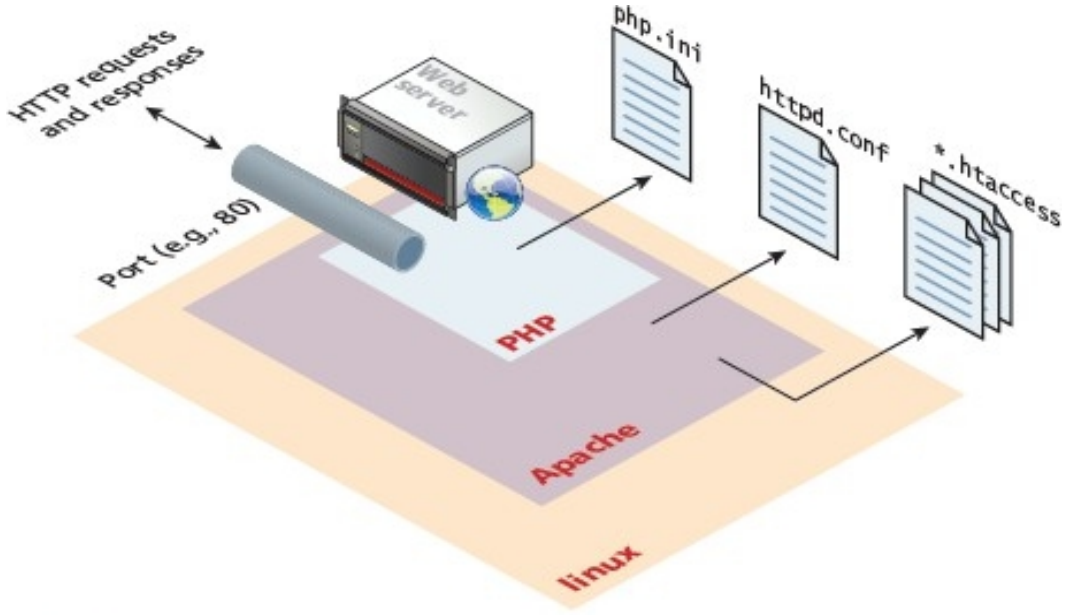

FIGURE 8.5 Linux, Apache, and PHP together

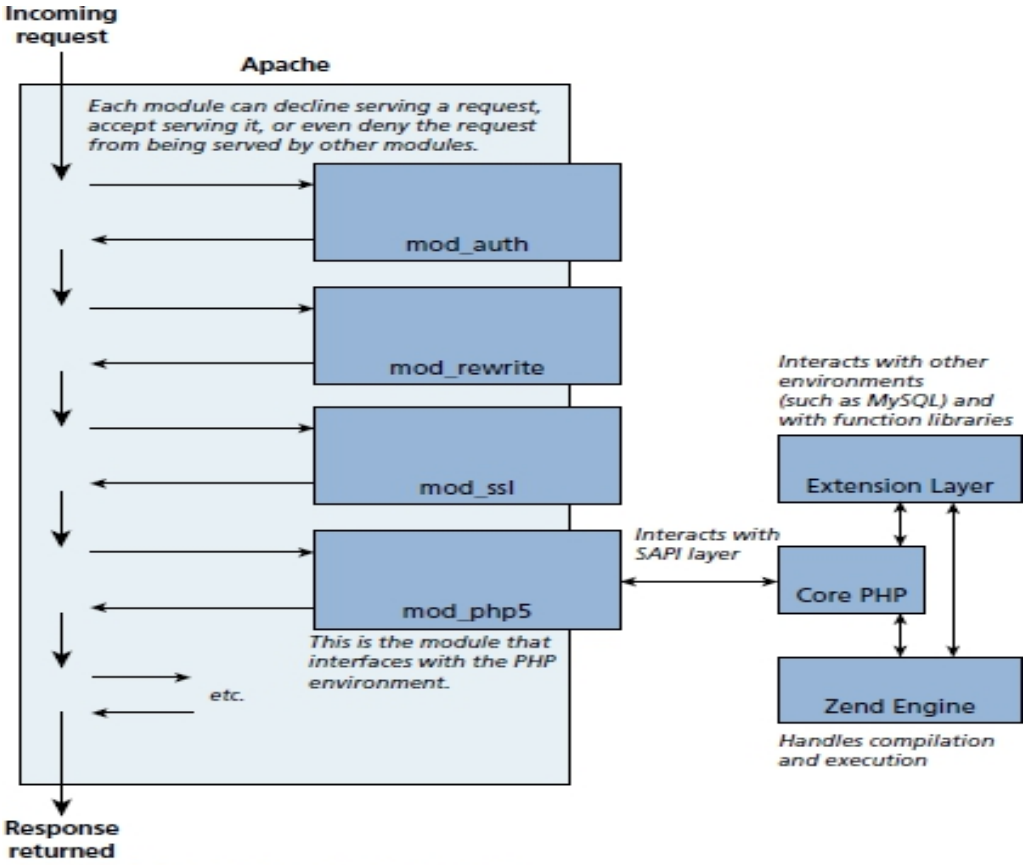

FIGURE 8.6 Apache modules and PHP

8.2.2 APACHE AND PHP

- PHP is usually installed as an Apache module. 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 **pre-forked**) 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 pre-fork 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.
- 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 lightweight 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.

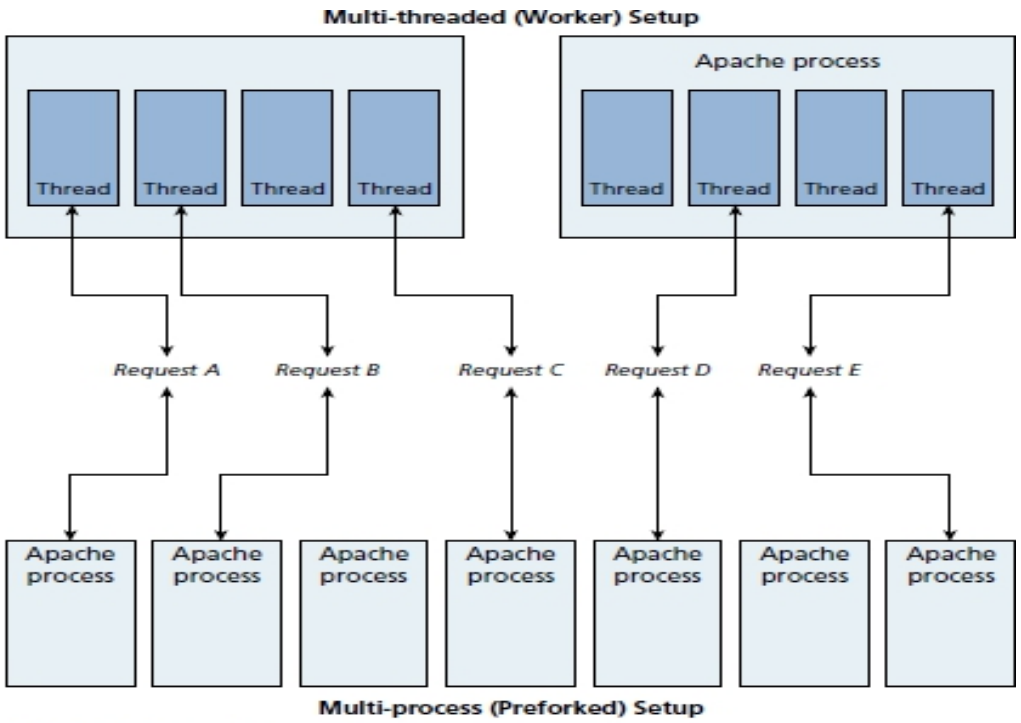

FIGURE 8.7 Multi-threaded versus multi-process

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 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

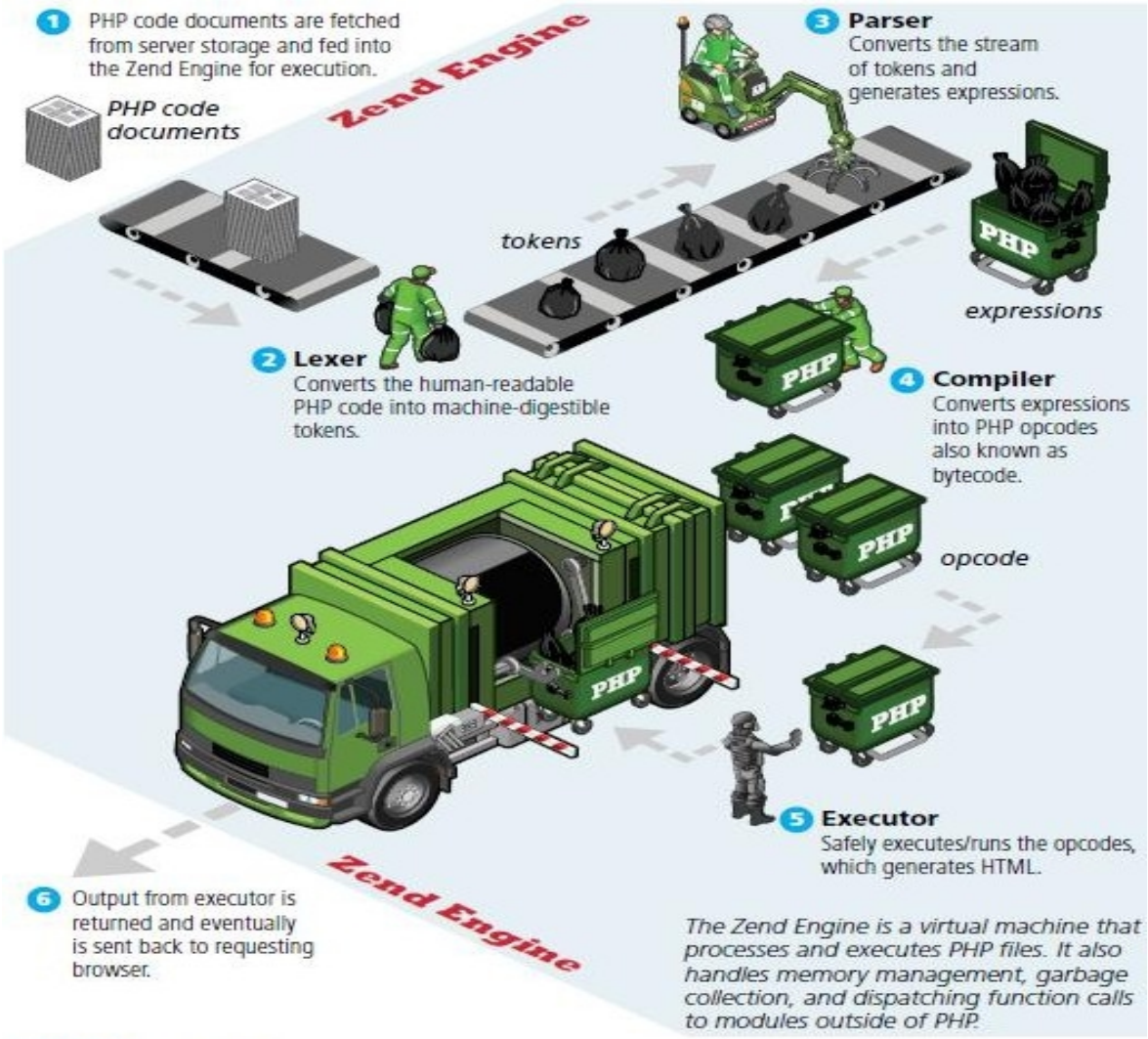

FIGURE 8.8 Zend Engine

8.2.4 INSTALLING APACHE, PHP, AND MYSQL FOR LOCAL DEVELOPMENT

- For Windows installation package (available at http://www.apachefriends.org/en/xampp-windows.html) or the MAMP for Mac installation package (available athttp://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.

- You may need to click the appropriate Start buttons to launch Apache. Once Apache has started, any subsequent PHP requests in your browser will need to use the **localhost** domain.
- 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. 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.

8.3 QUICK TOUR OF PHP

8.3.1 PHP TAGS

- 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

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 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 # are comment lines and will not be executed.
- **Multi-line (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.

```php
<?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 a programmer do not have to declare the data type of a variable.
- Variables are also **loosely typed** in that a variable can be assigned different data types over time.
- To declare a variable one must preface the variable name with the dollar ($) symbol. To use that variable, one must also include the $ symbol with it.
- Creating a variable named count and assigning it the value of 42 would be done with:

$count = 42;

- 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.
- While PHP is loosely typed, it still does have **data types,** which describe the type of content that a variable can contain.

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

- 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 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

8.3.4 WRITING TO OUTPUT

- To output something that will be seen by the browser, We 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.
- Listing 8.5 illustrates the fact that variable references can appear within string literals (but only if the literal is defined using double quotes).

```php
<?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

printf

- While echo is quite simple, more complex output can get confusing. As an alternative, we can use the printf() function.
- This function 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.
- 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 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.

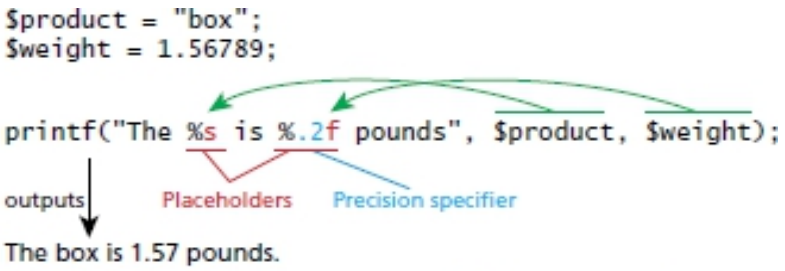

FIGURE 8.13 Illustration of components in a printf statement and output

- 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.

8.4 PROGRAM CONTROL

8.4.1 IF . . . ELSE

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.
- 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 can be difficult to match curly brackets up with this format, as perhaps can be 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 } ?>


<?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.

```
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

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.

```
$count = 0;
while ($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 contains the loop initialization, condition, and post-loop operations as in JavaScript. There is another type of for loop: the foreach loop. This loop is especially useful for iterating through arrays

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

LISTING 8.11 for loops

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;.

```
<?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 non-web programming languages, namely the ability to include or insert content from one file into another.
- 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.

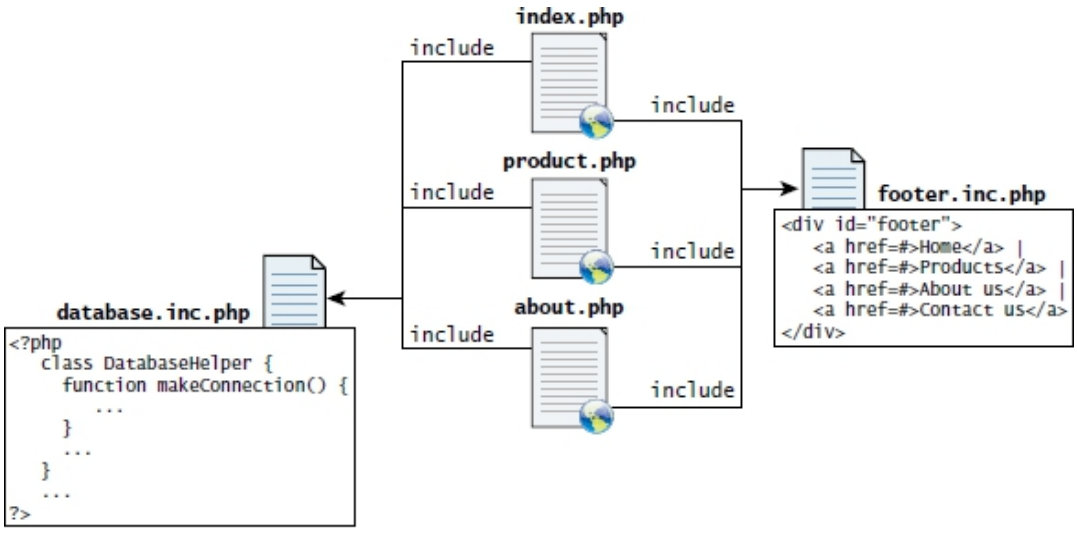

FIGURE 8.14 Include files

- Older web development technologies also supported include files, and were typically called **server-side includes (SSI)**.
- In a non-compiled 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.
- 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.

8.5 FUNCTIONS

- PHP allows you to define functions. 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.

8.5.2 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.
- 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.

```php
/**
* 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

```php
/**
* 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

8.5.3 CALLING A FUNCTION

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.

$output = getNiceTime();

echo getNiceTime();

If the function doesn't return a value, you can just call the function: outputFooterMenu();

8.5.4 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.
- 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.

```
/**
* 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

PARAMETER DEFAULT VALUES

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

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

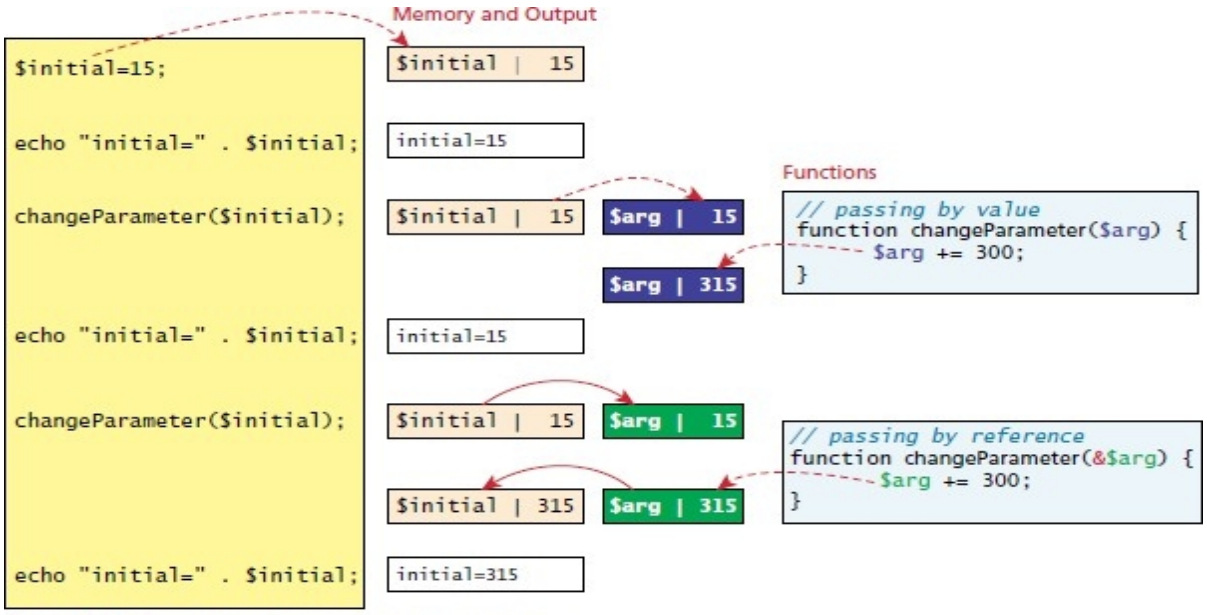

FIGURE 8.15 Pass by value versus pass by reference

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.

8.5.5 Variable Scope within Functions

It will come as no surprise that all variables defined within a function have **function scope**, meaning that they are only accessible within the function. Any variables created outside of the function in the main script are unavailable within a function. For instance, 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**, 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. But PHP does allow variables with global scope to be accessed within a function using the global keyword, as shown in Listing 8.19. From a programming design standpoint, the use of global variables should be minimized, and only used for vital application objects that are truly global.

```
$count= 56;

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

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

LISTING 8.19 Using the global keyword

QUESTIONS FROM PREVIOUS YEAR QP

1. Discuss the advantage and disadvantage of client-side scripting. **(8marks)**

2. Write a JavaScript code that displays text "VTU BELAGAVI" with increasing font size in the interval of 100ms in blue color, when the font size reaches 50pt it should stop. **(8marks)**

3. With a neat diagram, explain client and server script execution. **(8marks)**

4. Write a PHP program to greet the user based on the time. **(8marks)**

5. Explain three forms of linking JavaScript to HTML page with suitable code segments. **(8marks)**

6. With suitable diagram, explain PHP module in apache. Describe the role of apache threads in web application execution. **(8marks)**

7. Explain two methods in JavaScript to access DOM nodes with example. **(4marks)**

8. Explain two approaches for event handling in JavaScript with suitable code segments. **(6marks)**

9. With relevant code segments, explain two approaches to embed PHP scripts in HTML. **(6marks)**

10. What is JavaScript and listener? Discus the advantage and disadvantage of client – side scripting. **(8marks)**

11. What are software layers? What benefits do they provide? Explain in details. **(8marks)**

12. Compare the server – side technologies in details. **(8marks)**

13. Write a PHP program to demonstrate the session program: store page view count on refresh. **(8marks)**