

MODULE 3

Chapter 1: JavaScript: Client-Side Scripting

1. What is JavaScript and What can it do?
2. JavaScript Design Principles
3. Where does JavaScript Go?
4. Syntax
5. JavaScript Objects
6. The Document Object Model (DOM)
7. JavaScript Events
8. Forms

1.1 What Is JavaScript and What Can It Do?

- JavaScript is an object-oriented, dynamically typed, scripting language.
- 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.
- 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.
- JavaScript is object oriented, almost everything in the language is an object. **Ex:** variables are objects in that they have constructors, properties, and methods. JavaScript is dynamically typed in that variables can be easily 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 runtime as well.

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.

- There are many client-side languages like Flash, VBScript, Java, and JavaScript. Some of these technologies only work in certain browsers, while others require plug-ins to function.

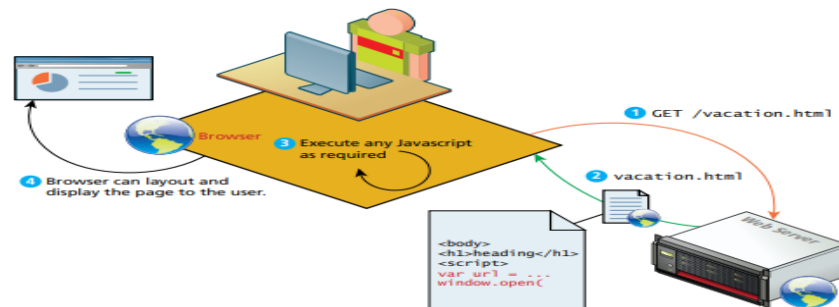


Figure 1.1: Downloading and executing a client-side JavaScript script

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.
- 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.

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 in to the HTML of a webpage.
- There are two other note worthy client-side approaches to web programming.

Adobe Flash

- It 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 **Action Script**.

- 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 (Shock wave 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 1.2.

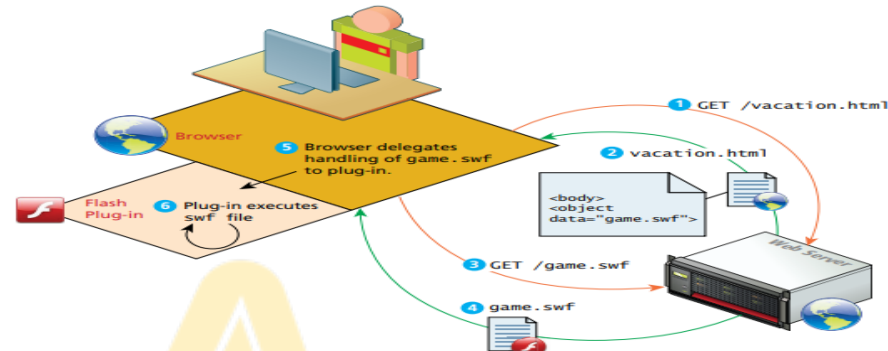


Figure 1.2: Adobe Flash

- 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. Browser plug-in is different than a **browser extension**.

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.

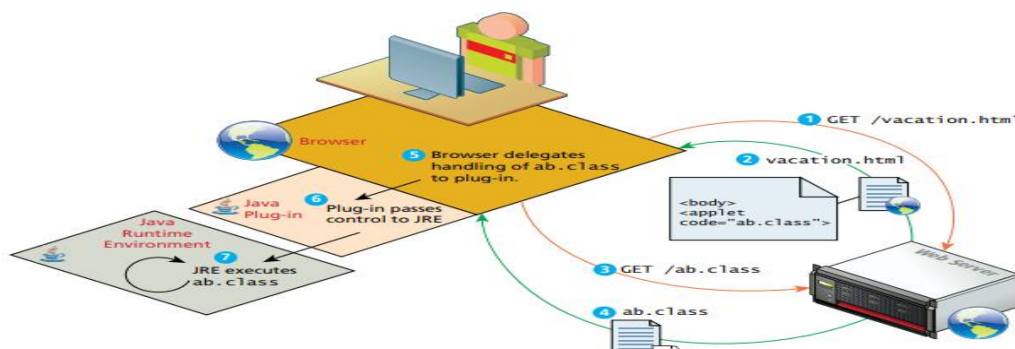


Figure 1.3: Java applets

- Both Flash plug-ins and Java applets are losing support by major players for a number of reasons.
 - Java applets require the JVM be installed and up to date.
 - 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.

JavaScript's History and Uses

- JavaScript was introduced by Netscape in their Navigator browser back in 1996. It originally was called Live Script.
- Java Script is in fact an implementation of a standardized scripting language called **ECMA Script**.
- Internet Explorer (IE) at first did not support Java Script, but instead had its own browser-based scripting language (VB Script).
- While IE now does support JavaScript, Microsoft sometimes refers to it as Jscript, primarily for trademark reasons (Oracle currently owns the trademark for JavaScript).

Common uses of java script:

- 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
 - pre-validating user data in online forms.
- AJAX sites that Java Script 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 sometime; 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.
 - Figure1.4 illustrates the processing flow for a page that requires updates based on user input using the normal synchronous non-AJAX page request-response loop.

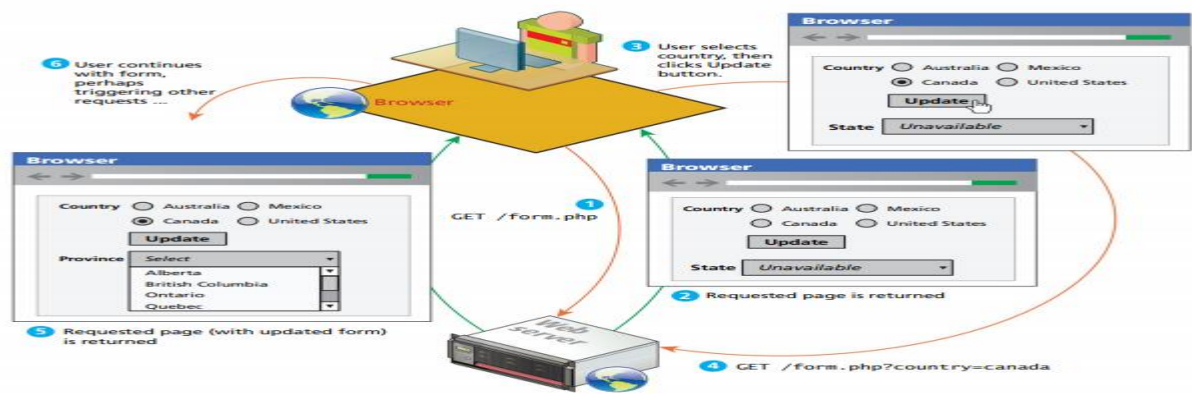


Figure 1.4: Normal HTTP request-response loop

- AJAX provides web authors with away to avoid the visual and temporal deficiencies Of normal HTTP interactions. With AJAX webpages, it is possible to update sections of a page by making special requests of the server in the background.
- This type of AJAX development can be difficult, s o the other key development in the history of JavaScript has made the creation of **JavaScript frameworks**, such as jQuery, Prototype, ASP.NET AJAX, and Moo Tools.
- These JavaScript frameworks reduce the amount of JavaScript code required to perform typical AJAX tasks.
- MVC JavaScript frameworks such as Angular JS, Backbone, and Knock out have gained a lot of interest from developers using a software engineering, namely the separation of the model (data representation) from the view (presentation of data) design pattern.

1.2 JavaScript Design Principles

- JavaScript principles increases the quality and reusability of the code while making it easier to understand, and hence have more maintainability.

Layers

- 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.

- Figure 1.5 illustrates the idea of JavaScript layers.

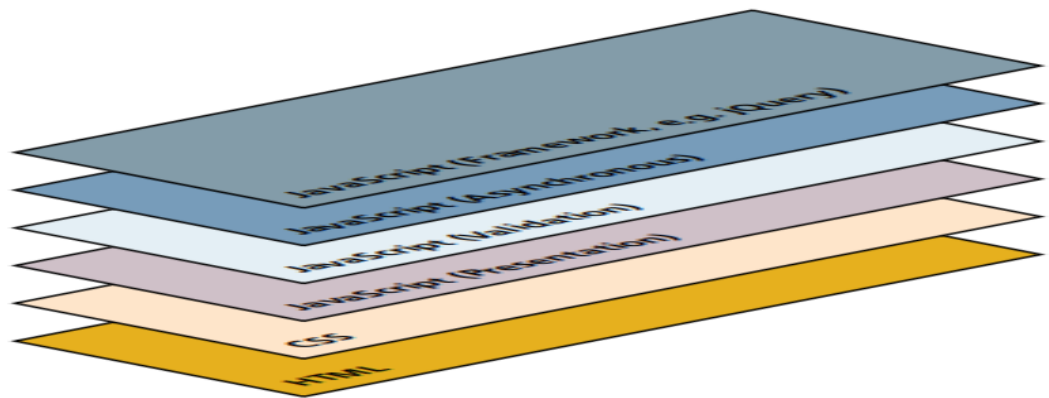


Figure 1.5: JavaScript layers

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.
- This layer enables 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, where a message to the presentation layer highlights bad fields.
- Both layers exist on the client machine, although the intention is to pre validate forms before making transmissions back to the server.

Asynchronous Layers

- JavaScript operates in asynchronous 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.
- 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.

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 our 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 with in 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.
- **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.
- **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.

The <NoScript> Tag

- 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.
- we should create websites with all the basic functionality enabled using regular HTML. For majority of users with JavaScript enabled we can then enhance the basic layout using JavaScript.
- 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 fails, then the system's design will still work.

Graceful Degradation and Progressive Enhancement

- The principle of fail-safe design can still apply even to browsers that have enabled JavaScript.

➤ Graceful degradation

- With this strategy we develop our site for the abilities of current browsers. For those users who are not using current browsers, we might provide an alternate site or pages for those using older browsers that lack the JavaScript used on the main site.
- Figure 1.6 (LEFT) illustrates the idea of graceful degradation.

➤ 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.
- Figure 1.6 (RIGHT) illustrates the idea of progressive enhancement.

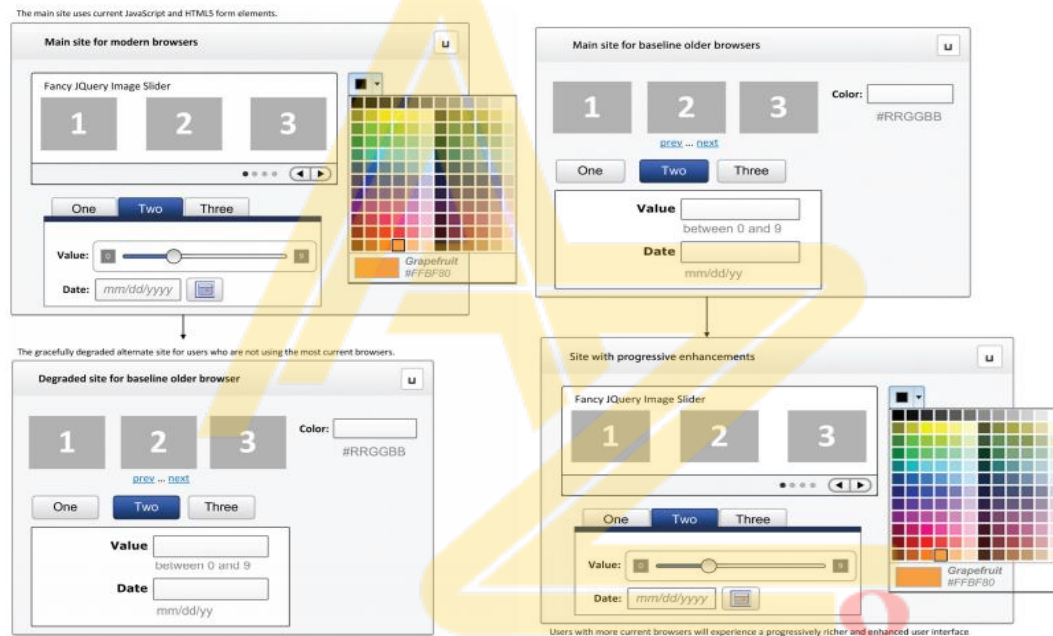


Figure 1.6: Examples of graceful degradation(left) and progressive enhancement(right)

1.3 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**, but external is the preferred method for cleanliness and ease of maintenance.

Inline JavaScript

- Inline JavaScript includes JavaScript code directly within certain HTML attributes, such as that shown in Listing 1.1.
- 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.

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

Listing 1.1: Inline JavaScript example

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

Listing 1.2: embedded JavaScript example

Embedded JavaScript

- Embedded JavaScript includes JavaScript code within a `<script>` element as shown in Listing 1.2. 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

External JavaScript

- JavaScript supports this separation by allowing links to an external file that contains the JavaScript.
- This is the recommended way of including JavaScript scripts in our HTML pages. By convention, JavaScript external files have the extension .js.
- These external files typically contain function definitions, data definitions, and other blocks of JavaScript code.
- In Listing 1.3, the link to the external JavaScript file is placed within the `<head>` element, a same as 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).

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

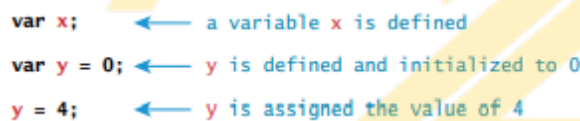
Listing 1.3: External JavaScript example

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.

1.4 Syntax

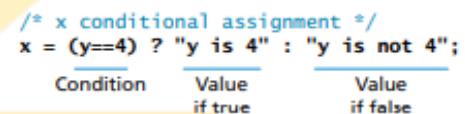
- Since it's a light weight scripting language.
- JavaScript has some features (such as dynamic typing) that are especially helpful to the novice programmer.
- **Features:**
 - Everything is type sensitive, including function, class, and variable names.
 - The scope of variables in blocks is not supported. 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 1.7: Variable declaration and assignment



```

/* x conditional assignment */
x = (y==4) ? "y is 4" : "y is not 4";
  
```

Condition	Value if true	Value if false
<code>(y==4)</code>	<code>"y is 4"</code>	<code>"y is not 4"</code>

Figure 1.8: Conditional assignment

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 variables, we use the var keyword, the name, and a semicolon as shown in Figure 1.7.
- If we specify no value, then (being type less) the default value is undefined.
- **Assignment** can happen at declaration-time by appending the value to the declaration, or at runtime with a simple right-to-left assignment.
- The **conditional assignment** operator, shown in Figure 1.8, can also be used to assign based on condition, although its use is sometimes discouraged.

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 1.1.

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 1.1: Comparison Operators

- These comparison operators are used in conditional, loop, and assignment statements.

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 1.2. Syntactically they are represented with && (and), ||(or), and! (not).

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

AND Truth Table

OR Truth Table

NOT Truth Table

Table 1.2: AND, OR, and NOT Truth Tables

Conditionals

- JavaScript's syntax is almost identical to that of PHP, Java, or C when it comes to conditional structures such as if and ifelse statements.
- In this syntax the condition to test is contained within() brackets with the body contained in{ } blocks.
- Optional elseif statements can follow, with an else ending the branch. Listing 1.4 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 1.4: Conditional statement setting a variable based on the hour of the day

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.

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

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:
- 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.

```
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.
- **Ex:** `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 webpage, and no actions can be done until the pop-up is dismissed.
- Alerts are not used in production code, but area useful tool for debugging and illustration purposes.

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, we can optionally catch these errors preventing disruption of the program using the **try-catch block** as shown in Listing 1.5.

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

Listing 1.5: Try-catch statement

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

Listing 1.6: Throwing a user-defined exception

Throwing Your Own Exceptions

- Although try-catch can be used exclusively to catch built-in JavaScript errors, it can also be used by our programs, to throw our own messages.
- The throw keyword stops normal sequential execution, just like the built-in exceptions as shown in Listing1.6 demonstrates the throwing of a user-defined exception as a string.
- The general consensus in software development is that try-catch and throw statements should be used for *abnormal* or *exceptional* cases in our 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.
- In reality any object can be thrown, although in practice a string usually suffices.

- 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.

1.5 JavaScript Objects

- 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; we can also define our own kind of objects.

Constructors

- Normally to create a new object we use the **new** keyword, the classname, 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. **Ex**, a String object can be defined with the shortcut.

```
var greeting = "Good Morning";  
Instead of the formal definition  
var greeting = new String("Good Morning");
```

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.

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.
- The 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"];
```

Accessing and Traversing an Array

- To access an element in the array we use the familiar square bracket notation from Java and C-style languages, 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().

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 1.7.

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

Listing 1.7: Some constants and functions in the Math object

String

- The **String class** has already been used without us even knowing it.
- 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
```

- 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 as single character using char At(), or
 - searching for one using index Of().
 - Strings allow splitting a string into an array,
 - searching and matching with split(),
 - search(), and match() methods.

Date

- The Date class is yet another helpful included object we should be aware of.
- It allows us 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());
```

Window Object

- The window object in JavaScript corresponds to the browser itself.
- Through it , we 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.

1.6 The Document Object Model(DOM)

- Java Script 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 structure (as shown in below figure 1.9) is formally called the **DOM Tree** with the root, or topmost object called the **Document Root**.

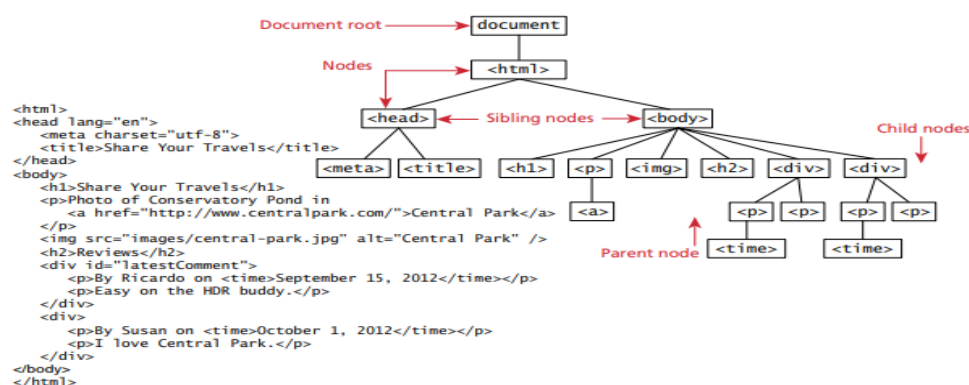


Figure 1.9: DOM tree

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 1.10.
- 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 1.3.

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

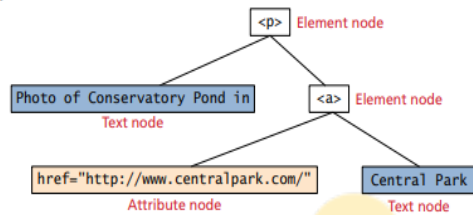


Figure 1.10: DOM nodes

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 1.3: Node Object Properties

Document Object

- The **DOM document object** is the root Java Script 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 doc type and input Encoding.
- Accessing the properties is done through the dot notation.

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

- There are some essential methods (Table 1.4) we will use all the time. They include getElementByTagName() and the indispensable getElementById().

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 1.4: Some Essential Document Object Methods

- While the former method returns an array of DOM nodes (called a Node List) matching the tag, the latter returns a single DOM element that matches the id passed as a parameter as illustrated in Figure 1.11.

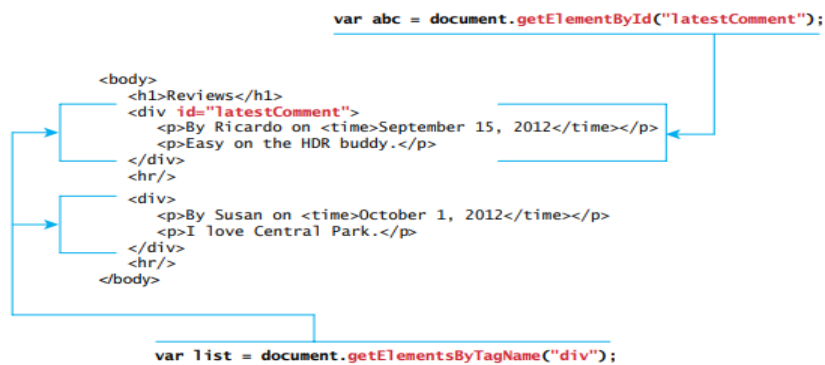


Figure1.11: Relationship between getElementById() and getElementsByTagName()

- 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.

Element Node Object

- The type of object returned by the method document.getElementById() is an element node object.
- This presents an HTML element in the hierarchy, contained between the opening <> and closing </> tags for this element.
- 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 1.3. It also has a variety of additional properties, the most important of which are shown in Table 1.5.

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 <div> 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 1.5: 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 1.6: HTML DOM Element Properties

- While these properties are available for all HTML elements, there are some HTML Elements that have additional properties that can be accessed. Table 1.6 lists some common additional properties and the HTML tags that have these properties.

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.
- Using the DOM document and HTML DOM element objects, we can do exactly that using the inner HTML property as shown in Listing 1.8 (using the HTML shown in Figure 1.11)

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

Listing 1.8: Changing the HTML using innerHTML

A More Verbose Technique

- Although the inner HTML 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 inner HTML could output badly formed HTML.
- DOM functions create-Text Node(), remove Child(), and append Child() allow us to modify an element in a more rigorous way as shown in Listing 1.9.

```
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 1.9: Changing the HTML using createTextNode() and appendChild()

Changing an Element's Style

- We can add or remove any style using the style or class Name property of the Element node, which is something that we 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.backgroundColor = "#FFFF00";  
commentTag.style.borderWidth="3px";
```

- With knowledge of CSS attributes we can easily change any style attribute.
- 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 class Name 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 Java Script 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");
```

Additional Properties

- In addition to the global properties present in all tags, there are additional methods available when dealing with certain tags. Table 1.6 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' />
```

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

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

1.7 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.
- As more powerful frameworks were developed, and website design and best practices were refined, this original mechanism was supplanted by the **listener** approach.

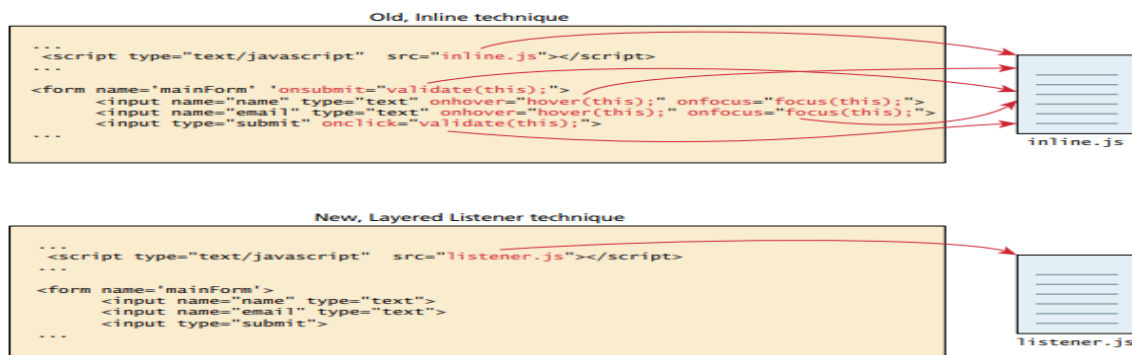


Figure 1.12: Inline hooks versus the Layered Listener technique

- A visual comparison of the old and new technique is shown in Figure 1.12. Note how the old method weaves the Java Script right inside the HTML, while the listener technique has removed JavaScript from the markup, resulting in cleaner, easier to maintain HTML code.

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 Java Script calls are intuitive.
- **Ex**, if we wanted an alert to pop-up, when clicking a <div> we might program:

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

- 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.

Listener Approach

- 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.

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

```
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 1.10: The “old”(registering a listener) **Listing 1.11:** The “new” (registering listeners)

- The approach shown in Listing 1.10 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 on click event to the event handler, which invokes the Java Script alert() method.
- 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.
- The one **limitation** with this approach (and the inline approach) is that only one handler can respond to any given element event.
- The use of add EventListener() shown in Listing 1.11 was introduced in DOM Version2, 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 1.10, And has the additional advantage that multiple handlers can be assigned to a single object’s event.

- The examples in Listing 1.10 and Listing 1.11 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 1.12.

```
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);
```

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

Listing 1.12: Listening to an event with a function **Listing 1.13:** An event with an anonymous function

- An alternative to that shown in Listing 1.12 is to use an anonymous function(that is, One without a name), as shown in Listing 1.13. This approach is especially common when the event handling function will only ever be used as a listener.

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.
 - **Bubbles.**
 - The bubbles property is a Boolean value. If an event's bubble 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.
- **prevent Default.**
 - A cancelable default action for an event can be stopped using the prevent Default() method as shown in Listing 1.14.

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

Listing 1.14: A function that prevents the default event

- This is a common practice when we want to send data asynchronously when a form is submitted, **Ex** 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).

Event Types

- 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 with in 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 inter actions driven by the mouse. These can be further categorized as mouse click and mouse move events.
- Table 1.7 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 on to another in the same moment, triggering on mouse on and on mouse out events as well as the on mouse move 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 1.7: Mouse Events in JavaScript

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 1.8: Keyboard Events in JavaScript

Keyboard Events

- Keyboard events are often overlooked by novice web developers, but are important tools for power users.
- Table 1.8 lists the possible key board events. These events are most useful within input fields.
- **Ex** validate an email address, or send an asynchronous request for a drop down 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 1.14.

```
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 1.14: Listener that hears and alerts key presses

Form Events

- Forms are the main means by which user input is collected and transmitted to the server.
- Table 1.9 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 login Form. If the password field (with id pw) is blank, we prevent submitting to the server using prevent Default()and alert the user.

- Otherwise we do nothing, which allows the default event to happen (submitting the form) as shown in Listing 1.15.

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 1.9: Form Events in JavaScript

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 1.10: Frame Events in JavaScript

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

Listing 1.15: Catching the onsubmit event and validating a password to not be blank

Frame Events

- Frame events (see Table 1.10) are the events related to the browser frame that contains our webpage.
- The most important event is the onload event, which tells us an object is loaded and therefore ready to work with.
- In fact, every non trivial event listener we write requires that the HTML be fully loaded.
- However, a problem can occur if the Java Script tries to reference a particular <div> in the HTML page that has not yet been loaded.
- If the code attempts to set up a listener on this not-yet-loaded<div>, 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.
}
```

1.8 Forms

To illustrate some form-related JavaScript concepts, consider the simple HTML form depicted in Listing 1.16.


```
<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 1.16: A basic HTML form for a login example

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, 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.

Empty Field Validation

- A common application of a client-side validation is to make sure the user entered Something in to a field.
- There's certainly no points ending a request to login if the user name 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 1.17 related JavaScript concepts, consider the simple HTML form depicted in Listing 1.16.

```
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 1.17: Validation to check for empty fields

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

Listing 1.18: A function to test for a numeric value

Number Validation

- Number validation can take many forms. We 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()`.
- 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 1.18.

Note: This function will not parse “European” style numbers with commas (i.e., 12.00 vs.12,00).

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 our 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 we design your applications.

Chapter 2: Introduction to Server-Side Development with PHP

1. What is Server-Side Development
2. A Web Server's Responsibilities
3. Quick Tour of PHP
4. Program Control
5. Functions

2.1 What Is Server-Side Development?

- The basic hosting of our files is achieved through a web server.
- 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, we 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.

Comparing Client and Server Scripts

- Figure 2.1 illustrates how client and server scripts differ.

<u>Client-side Scripts</u>	<u>Server side Scripts</u>
Client-side scripts are executed on the client browser.	Server-side scripts are executed on the web server.
Java Script code is downloaded to the client and is executed there.	The server sends the JavaScript, but you have no guarantee that the script will even execute.
The clients never get to see the code, just the HTML output from the script.	Server-side source code remains hidden from the client as it is processed on the server.
Client-side scripts cannot make use of the resources.	Server-side scripts can make use of resources available at the server.

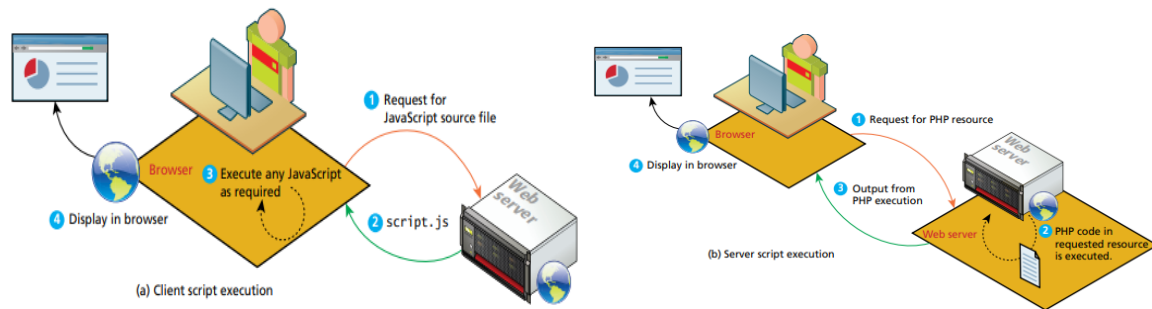


Figure 2.1: Comparison of (a) client script execution and (b) server script execution

Server-Side Script Resources

- A server-side script can access any resources made available to it by the server like data storage resources, web services and software applications.
- **Data Storage** 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.
- **Web services** use the HTTP protocol to return XML or other data formats and are often used to extend the functionality of a website.
- **Additional software** that can be installed on a server or accessed via a network connection. Using other software like server applications can send and receive email, access user authentication services, and use network accessible storage.

Comparing Server-Side Technologies

- **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; 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).

- 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.
- **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.
 - It also uses a JIT compiler for fast execution time and is cross-platform.
- **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.

- Originally, PHP stood for personal home pages, but now it stands for **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.

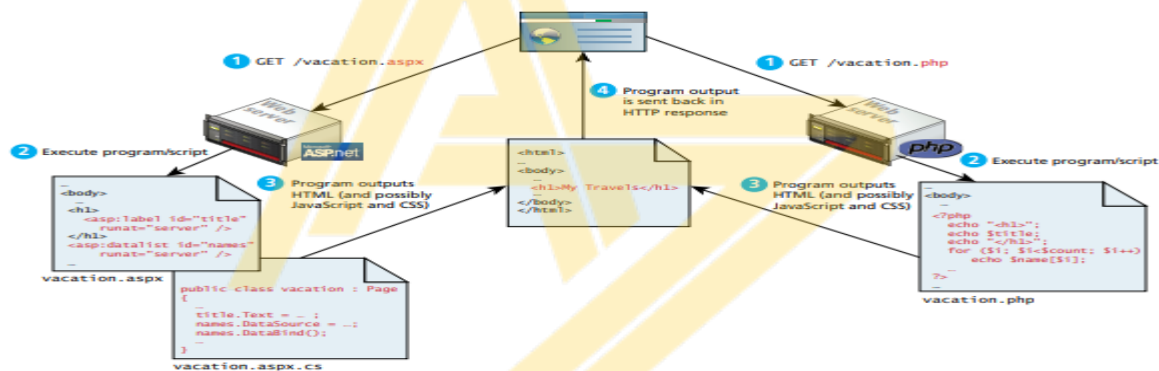


Figure 2.2: Web development technologies

2.2 A Web Server's Responsibilities

- 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
 - Uploading and managing files.

Apache and Linux

- Apache web server is 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.
- 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.

Apache and PHP

- PHP is usually installed as an Apache module and 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.

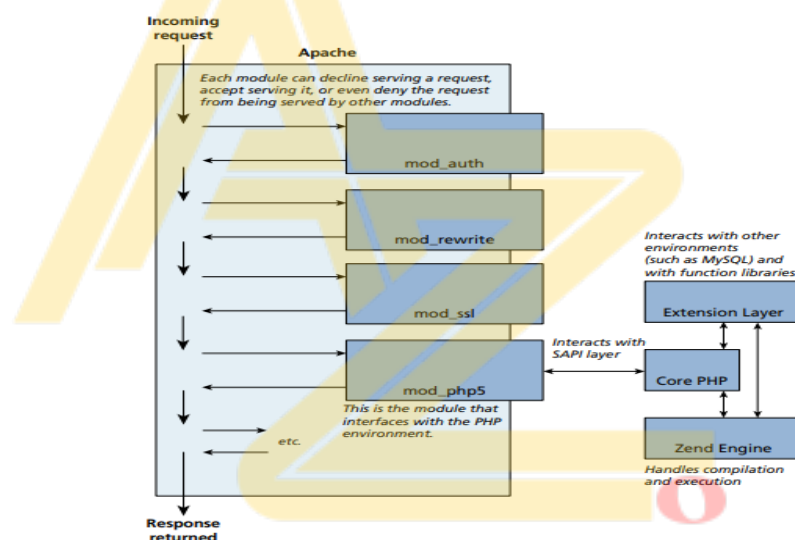


Figure 2.3: Apache modules and PHP

- Apache runs in two possible modes: multi-process (also called preforked) or multi-threaded (also called worker).

Multi-process:

- 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.
- 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.

Multi-threaded:

- 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.
- When using this mode, all modules running within Apache have to be thread safe.

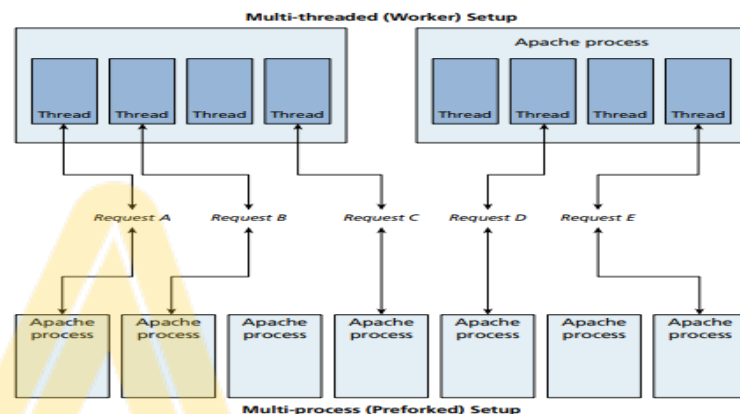


Figure 2.4: Multi-threaded versus multi-process

PHP Internals

1. 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.

2. 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.

3. Zend Engine.

- This module handles the reading in of a requested PHP file, compiling it, and executing it.
- Figure 2.4 illustrates 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.

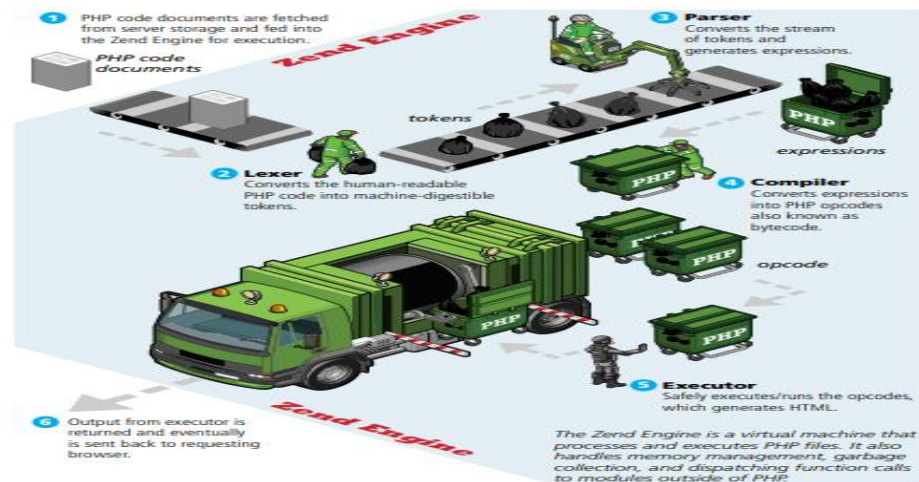


Figure 2.5: Zend Engine

Installing Apache, PHP, and MySQL for Local Development

- For Windows installation package (available at <http://www.apachefriends.org/en/xampp-windows.html>).
- Once the XAMPP package is installed in Windows, we can then run the XAMPP control panel.
- We may need to click the appropriate Start buttons to launch Apache (and later MySQL).
- Once Apache has started, any subsequent PHP requests in our browser will need to use the localhost domain.
- Now we are ready to start creating our own PHP pages. If we used the default XAMPP installation location, our PHP files will have to be saved somewhere within the C:\xampp\htdocs folder.

2.3 Quick Tour of PHP

- It 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 differs in functions, classes, and in how we define variables.

PHP Tags

- PHP 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**.

- 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 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 2.1: PHP tags

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

Listing 2.2: Listing 2.1 in browser

- We can separate the PHP code from HTML code as we have done in CSS and JS to increase the maintainance and accessibility by making use of include statements, which insert the content of the specified file into the current file.



Figure 2.6: Two approaches to PHP coding

PHP Comments

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(/* */)**
 - 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 /*, we are identifying blocks of comment that can later be parsed for inclusion in generated documents.
- **End-of-line comments(//)**
 - 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
# 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 2.3: PHP comments

Variables, Data Types, and Constants

- Variables in PHP are dynamically typed, which means that we as 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 we must preface the variable name with the dollar (\$) symbol.

- Whenever we use that variable, we must also include the \$ symbol with it.
- We 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;
```
- We 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.
- While PHP is loosely typed, it still does have data types, which describe the type of content that a variable can contain.
- Table 2.1 lists the main data types within PHP. We do not declare a data type. Instead the PHP engine determines the data type when the variable is assigned a value.
- 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.

Data Type	Description	Sequence	Description
Boolean	A logical true or false value	\n	Line feed
Integer	Whole numbers	\t	Horizontal tab
Float	Decimal numbers	\\	Backslash
String	Letters	\\$	Dollar sign
Array	A collection of data of any type (covered in the next chapter)	\"	Double quote
Object	Instances of classes		

Table 2.1: PHP Data Types**Table 2.2:** String Escape Sequences

Constant

- 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 2.4.
- 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 2.4: PHP constants**Writing to Output**

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

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

```
<?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 '';
echo "<img src=\"23.jpg\" >";

?>
```

Listing 2.5: PHP quote usage and concatenation approaches

- The first example in above listing illustrates the fact that variable references can appear within string literals (but only if the literal is defined using double quotes).
- As such, it is important to take some time to experiment and evaluate some sample concatenation statements as shown in Listing 2.6.
- The output for the statements in Listing 2.6 is provided in figure 2.7.

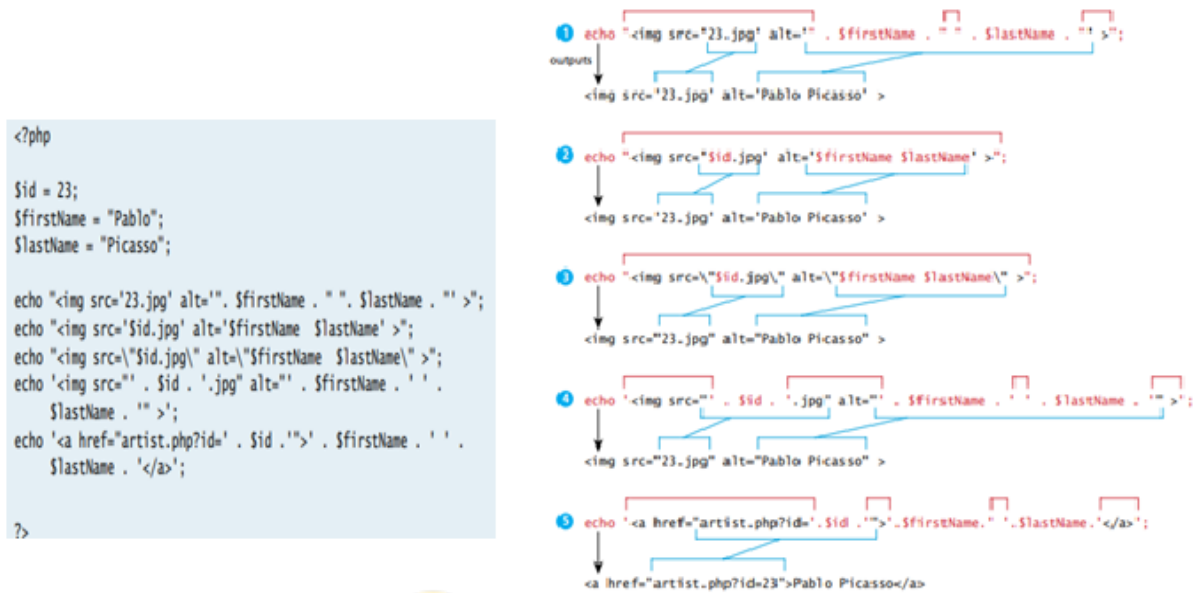


Figure 2.7: More Complicated concatenation examples Explained

Printf

- As the examples discussed above illustrate, while echo is quite simple, more complex output can get confusing.
- As an alternative, We can use the printf() function.
- 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 2.8 illustrates the relationship between the first parameter string, its placeholders and subsequent parameters, precision, and output.

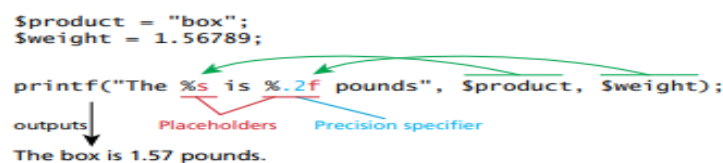


Figure 2.8: Illustration of components in a printf statement and output

- 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.

- When programming, we may prefer to use printf() for more complicated formatted output, and use echo for simpler output.

2.4 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 for each loop.

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 as in Listing 2.6.

```
// 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 2.6: if . . . else

```
<?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 2.7: Combining PHP and HTML in the script

- It is also possible to place the body of an if or an else outside of PHP as in Listing 2.7.

switch...case

The switch statement is similar to a series of if...else statements as in Listing 2.8.

```
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 2.8: Conditional statement using switch

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

Listing 2.9: while loops

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 2.9 provides examples of each type of loop.

For

- The for loop in PHP has the same syntax as the for loop in JavaScript that we examined and the for loop contains the same 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 2.10: for loops

```
<?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 2.11: Alternate syntax for control structures

Alternate Syntax for Control Structures

- PHP has an alternative syntax for most of its control structure 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 2.11), the colon (:) replaces the opening curly bracket, while the closing brace is replaced with endif;, endwhile;, endfor;, endforeach;, or endswitch; as in Listing 2.11.

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.

- Include files provide a mechanism for reusing both markup and PHP code, as shown in Figure 2.9

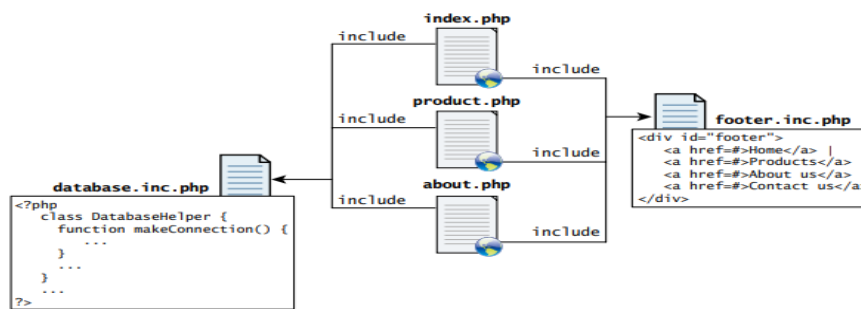


Figure 2.9: Include Files

- 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.

Include

- With `include`, a warning is displayed and then execution continues.
- The `include_once` statements works as `include` but if the requested file has already been included once, then it will not be included again.

Require

- With `require`, an error is displayed and execution stops.
- The `require_once` statements works as `include` but if the requested file has already been included once, then it will not be included again.

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.
- 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 and classes.

2.5 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 our code in the main body of a script makes it hard to reuse, maintain, and understand.
- As an alternative, PHP allows us 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 we 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.

Function Syntax

- To create a new function we must think of a name for it, and functions can return values to the caller, or not return a value.
- They can be setup 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.
- We will notice that the definition requires the use of the function key word followed by the function's name, round() brackets for parameters, and then the body of the function inside curly{ } brackets.
- Listing 8.13 returns a value and Listing 8.14 illustrates a function definition that doesn't return a value but just performs a task.

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

Listing 2.12: function to return value

```
/**
 * 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 2.13: function without a return value

Calling a Function

- We have defined a function, we 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

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

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

```
outputFooterMenu();
```

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.
- As shown in Listing 2.14. 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 2.14: 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
```

Parameter Default Values

- If we could not simply combine the two overloaded functions together into one so that if we call it with no parameter, it uses a default value.
- 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.
- Listing 2.15 illustrate the default values.

```

/**
 * 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 2.15: A function to return the current time with a parameter that includes a default

- If you do include a value in your function call, the default will be overridden by whatever that value was.
- Either way we can 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 2.16 illustrates a simple example of passing by value.

```

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 2.16: Passing a parameter by value

```

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 2.17: Passing a parameter by reference

- 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 2.17 illustrates an example of passing by reference.
- Figure 2.10 illustrates visually the memory differences between pass-by value and pass-by reference.
- By and large, we will likely find that most of the time we will use pass-by value in the majority of our functions.

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.

```
$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.
- 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.
- But PHP does allow variables with global scope to be accessed within a function using the global keyword, as shown in Listing 2.18.

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

Listing 2.18: 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.