JavaScript: Client-Side Scripting

Chapter 6

Randy Connolly and Ricardo Hoar

Objectives

1. What is JavaScript
2. JavaScript Design
3. Using JavaScript
4. Syntax
5. JavaScript Objects
6. The DOM
7. JavaScript Events
8. Forms

Section 1 of 8

WHAT IS JAVASCRIPT

What is JavaScript

- JavaScript runs right inside the browser
- JavaScript is dynamically typed
- JavaScript is object oriented in that almost everything in the language is an object
    - the objects in JavaScript are prototype-based rather than class-based, which means that while JavaScript shares some syntactic features of PHP, Java or C#, it is also quite different from those languages

What isn't JavaScript

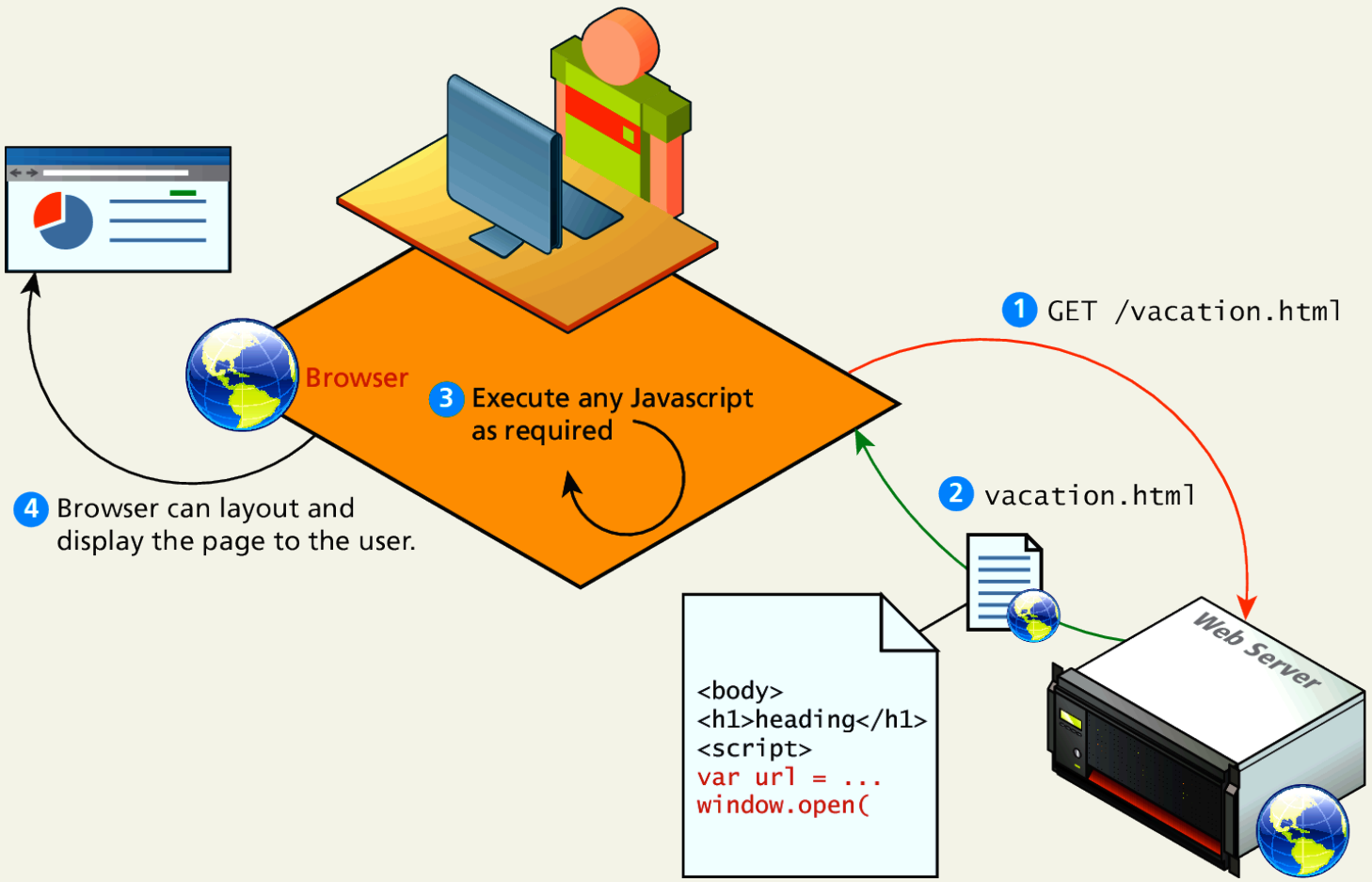
It's not Java

Although it contains the word *Java*, JavaScript and Java are vastly different programming languages with different uses. Java is a full-fledged compiled, object-oriented language, popular for its ability to run on any platform with a JVM installed.

Conversely, JavaScript is one of the world's most popular languages, with fewer of the object-oriented features of Java, and runs directly inside the browser, without the need for the JVM.

Client-Side Scripting

Let the client compute



Client-Side Scripting

It's good

There are many **advantages** of client-side scripting:

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

Client-Side Scripting

There are challenges

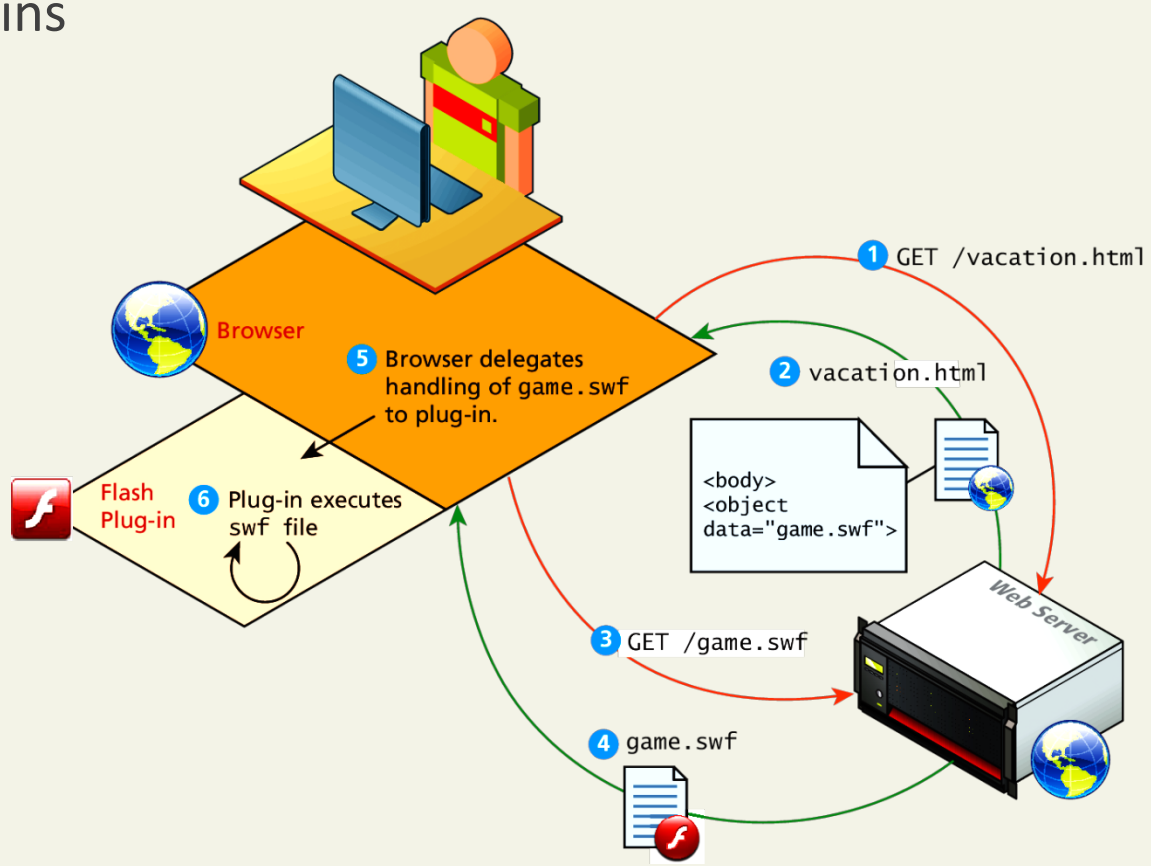
The disadvantages of client-side scripting are mostly related to how programmers use JavaScript in their applications.

- There is no guarantee that the client has JavaScript enabled
- 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.

Client-Side Flash

JavaScript is not the only type of client-side scripting.

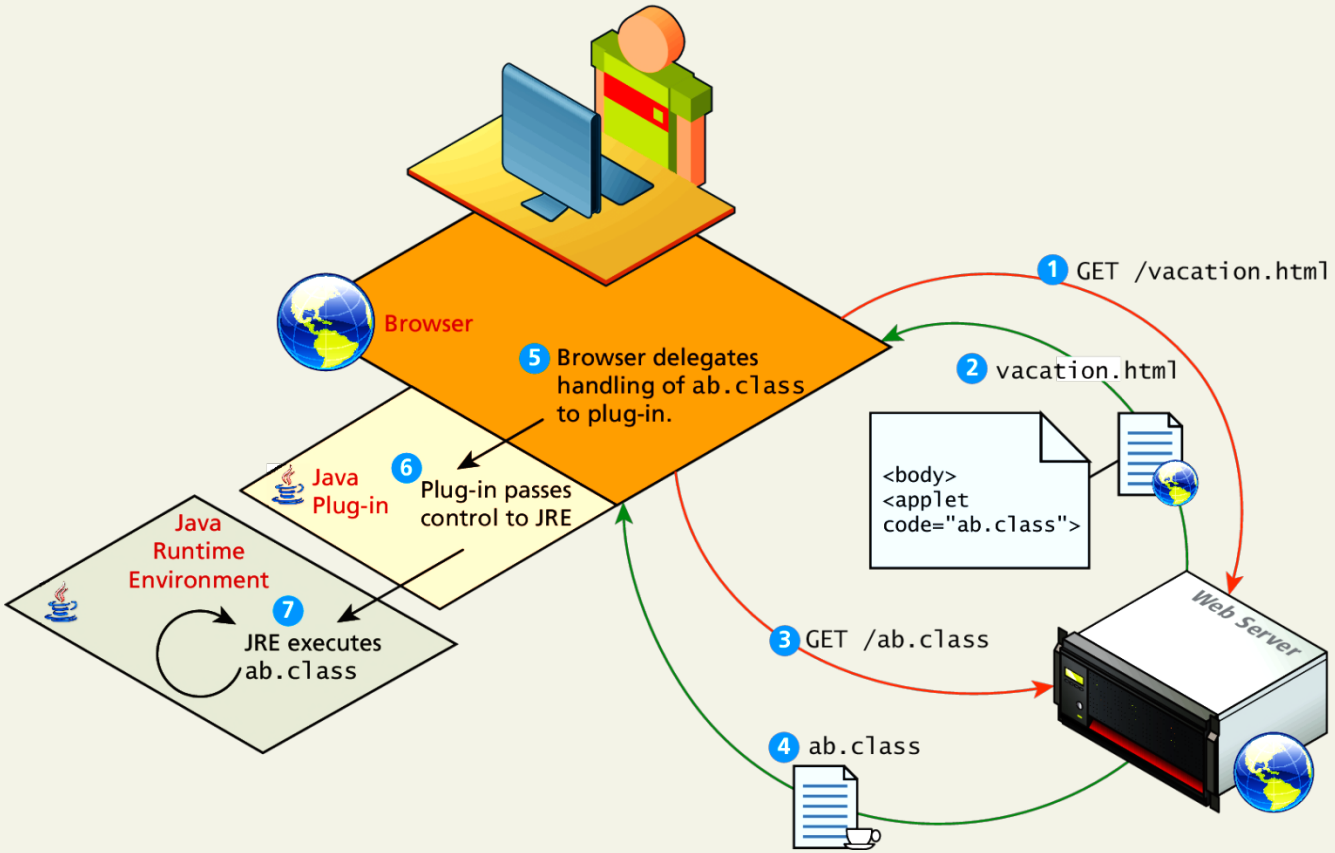
- Browser Plug-ins
  - Flash



Client-Side Applets

Java Applets

Java applets are written in and are separate objects included within an HTML document via the <applet> tag

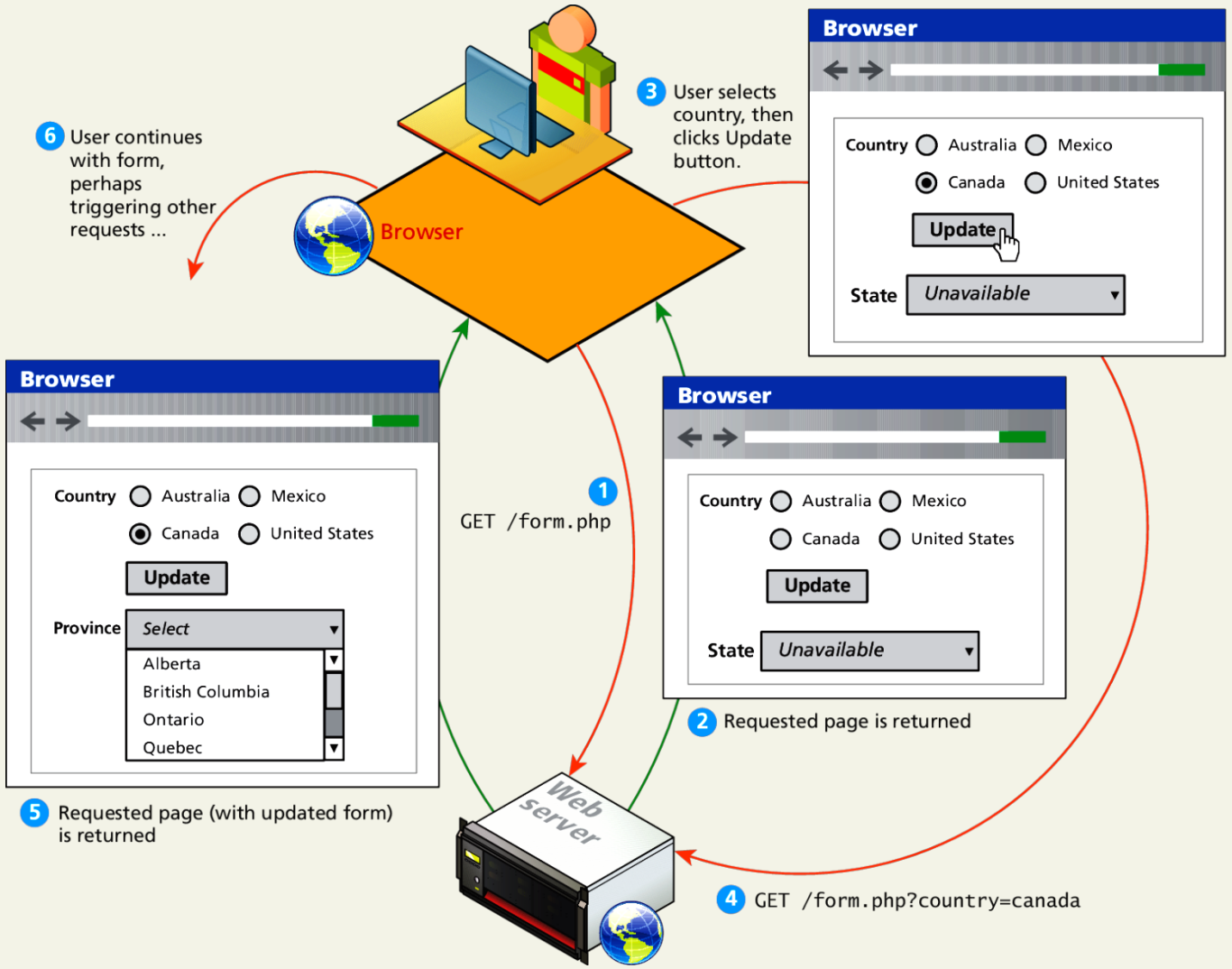


JavaScript History

- JavaScript was introduced by Netscape in their Navigator browser back in 1996.
- JavaScript is in fact an implementation of a standardized scripting language called **ECMAScript**
- JavaScript was only slightly useful, and quite often, very annoying tomany users

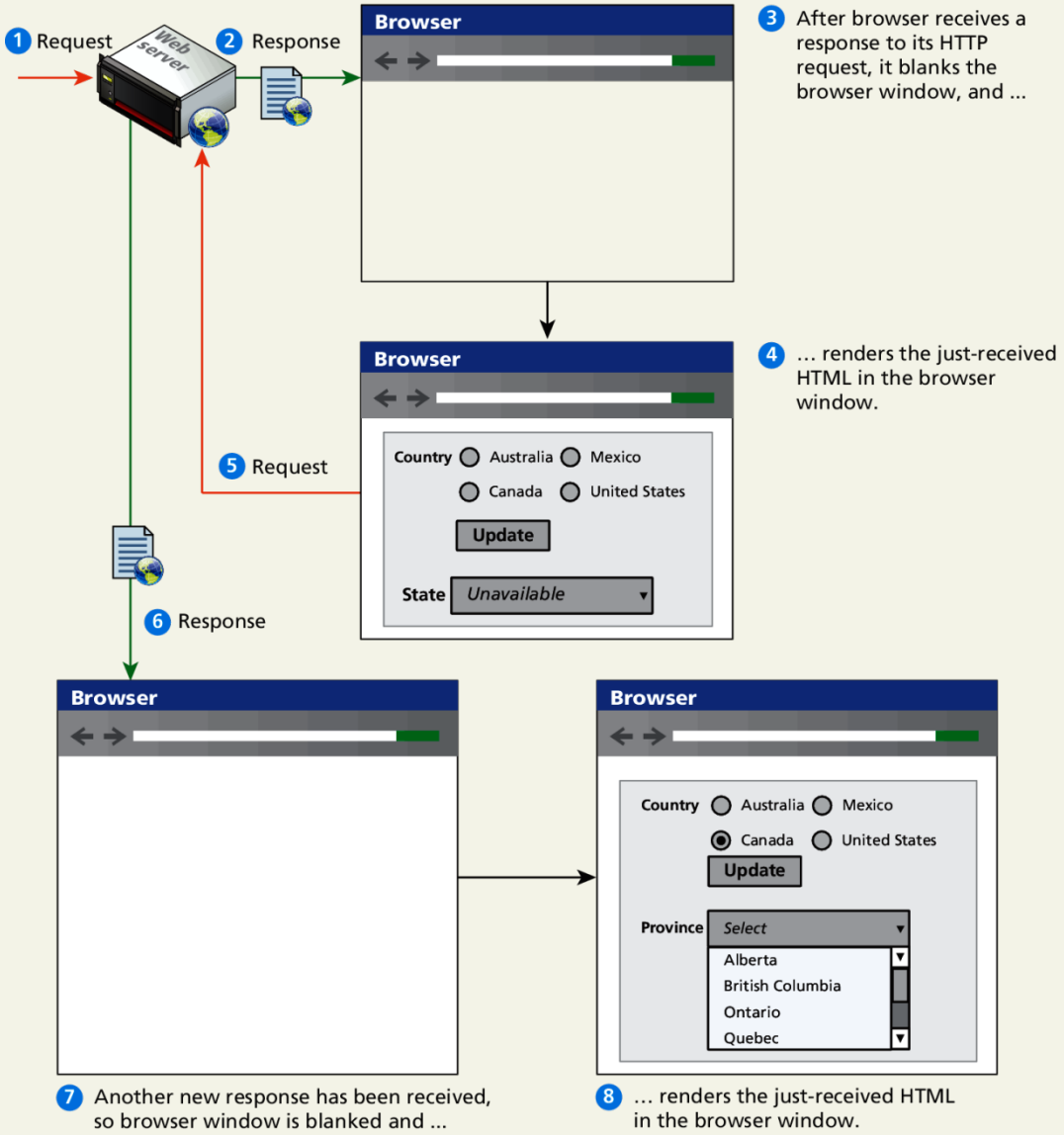
HTTP request-reponse loop

Without JavaScript



HTTP request-reponse loop

Detail



JavaScript in Modern Times

AJAX

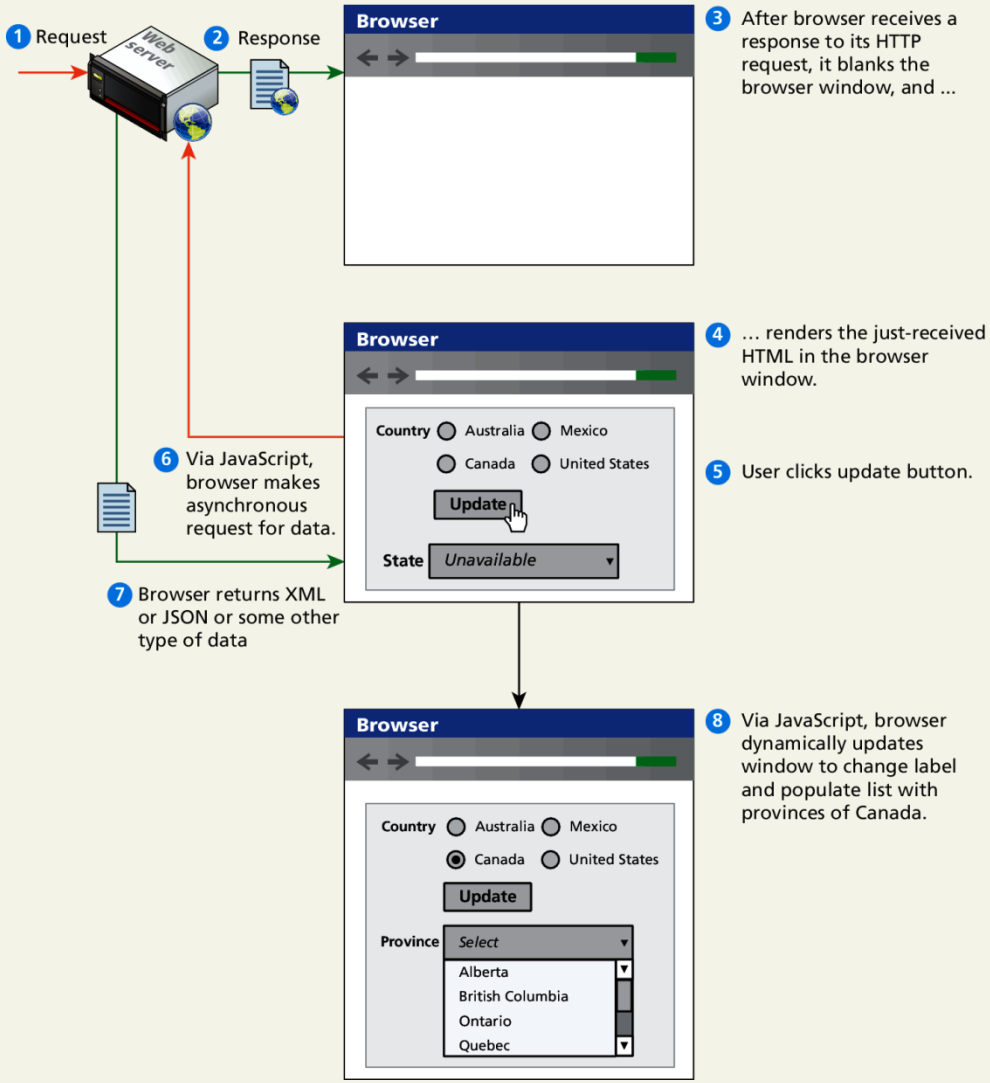
JavaScript became a much more important part of web development in the mid 2000s with **AJAX**.

AJAX is both an acronym as well as a general term.

- As an acronym it means **A**synchronous **J**avaScript **A**nd **X**ML.
- The most important feature of AJAX sites is the asynchronous data requests.

Asynchronous data requests

The better AJAX way



Frameworks

Lots of this is done for you, once you get the basics

Section 2 of 8

JAVASCRIPT DESIGN PRINCIPLES

Layers

They help organize

When designing software to solve a problem, it is often helpful to abstract the solution a little bit to help build a cognitive model in your mind that you can then implement.

Perhaps the most common way of articulating such a cognitive model is via the term **layer**.

In object-oriented programming, a software **layer** is a way of conceptually grouping programming classes that have similar functionality and dependencies.

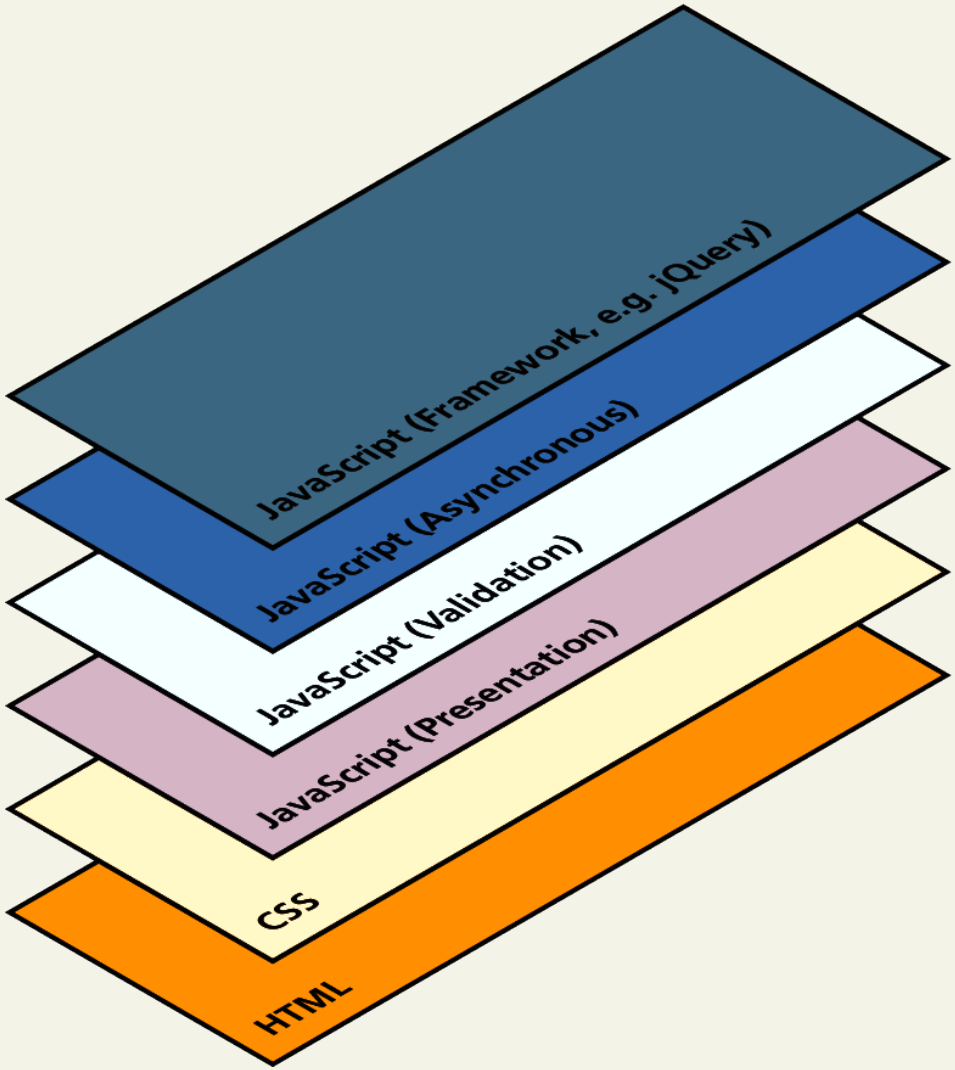
Layers

Common Layers

- **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.

Layers

Just a conceptual idea



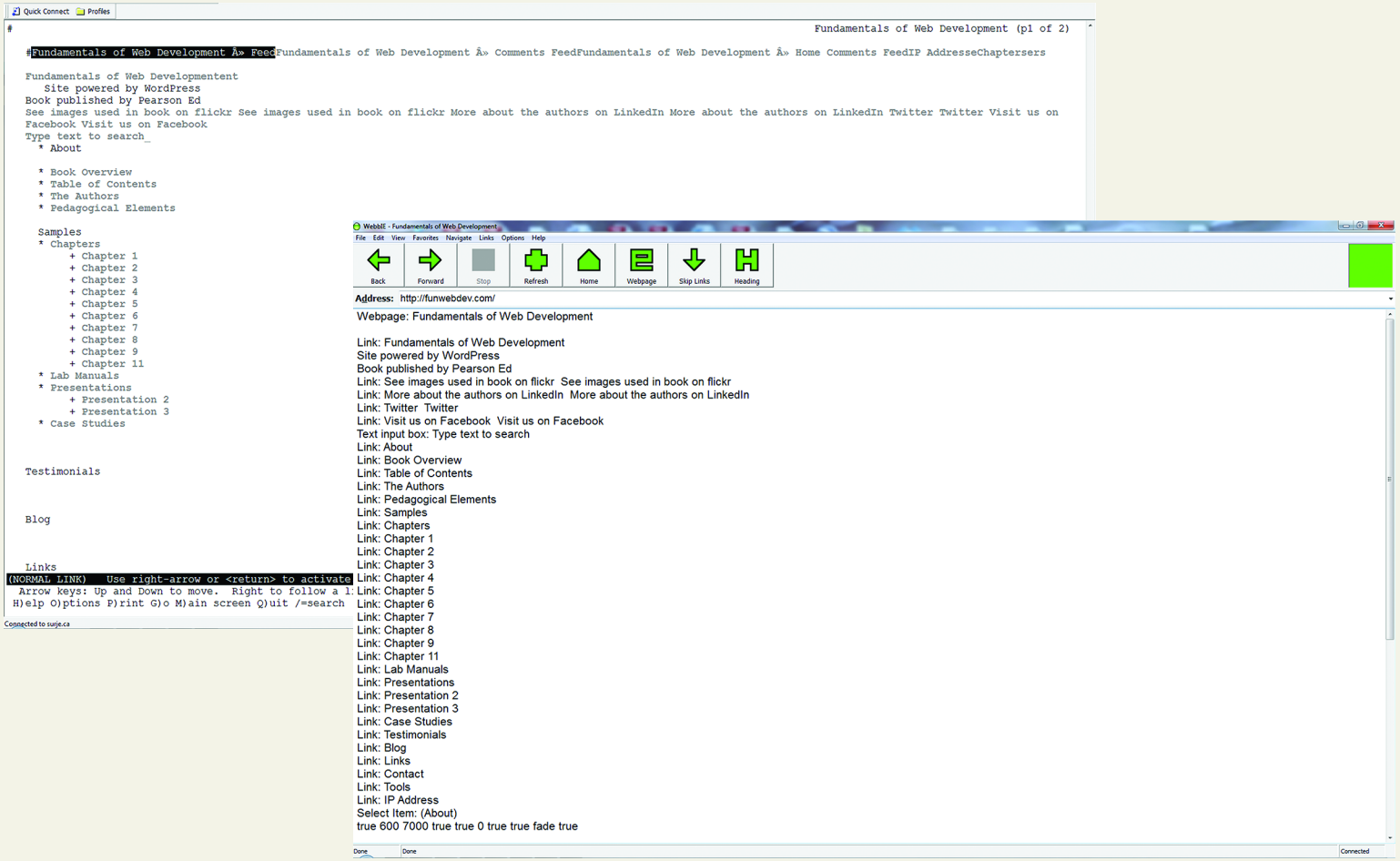
Users Without Javascript

They do exist

- **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.
- **Browser plug-in**. A browser plug-in is a piece of software that works within the browser, that might interfere with JavaScript.
- **Text-based client**. Some clients are using a text-based browser.
- **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.

Users Without Javascript

Lynx, and WebIE



The <noscript> tag

Mechanism to speak to those with JavaScript

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.

Requiring JavaScript (or Flash) for the basic operation of your site will cause problems eventually and should be avoided.

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.

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.

There are two strategies:

- **graceful degradation**
- **progressive enhancement**

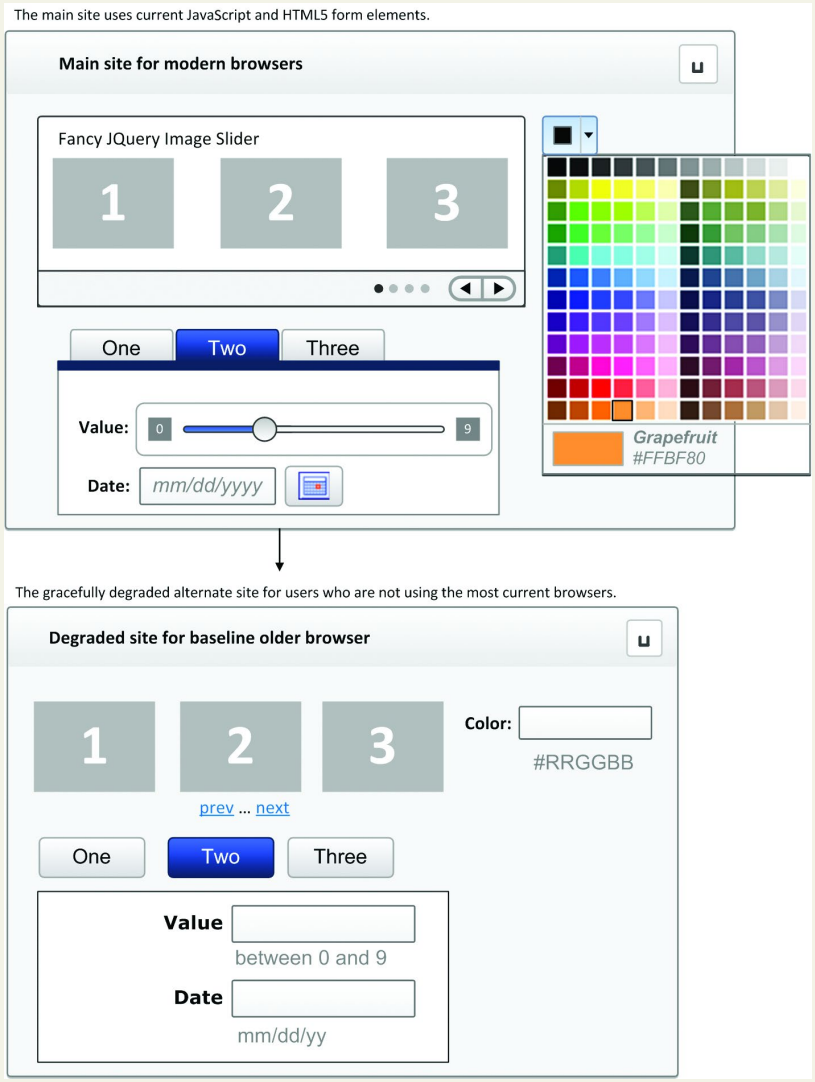
Graceful Degradation

With this strategy you develop your site for the abilities of current browsers.

For those users who are not using current browsers, you might provide an alternate site or pages for those using older browsers that lack the JavaScript (or CSS or HTML5) used on the main site.

The idea here is that the site is “degraded” (i.e., loses capability) “gracefully” (i.e., without pop-up JavaScript error codes or without condescending messages telling users to upgrade their browsers)

Graceful Degradation

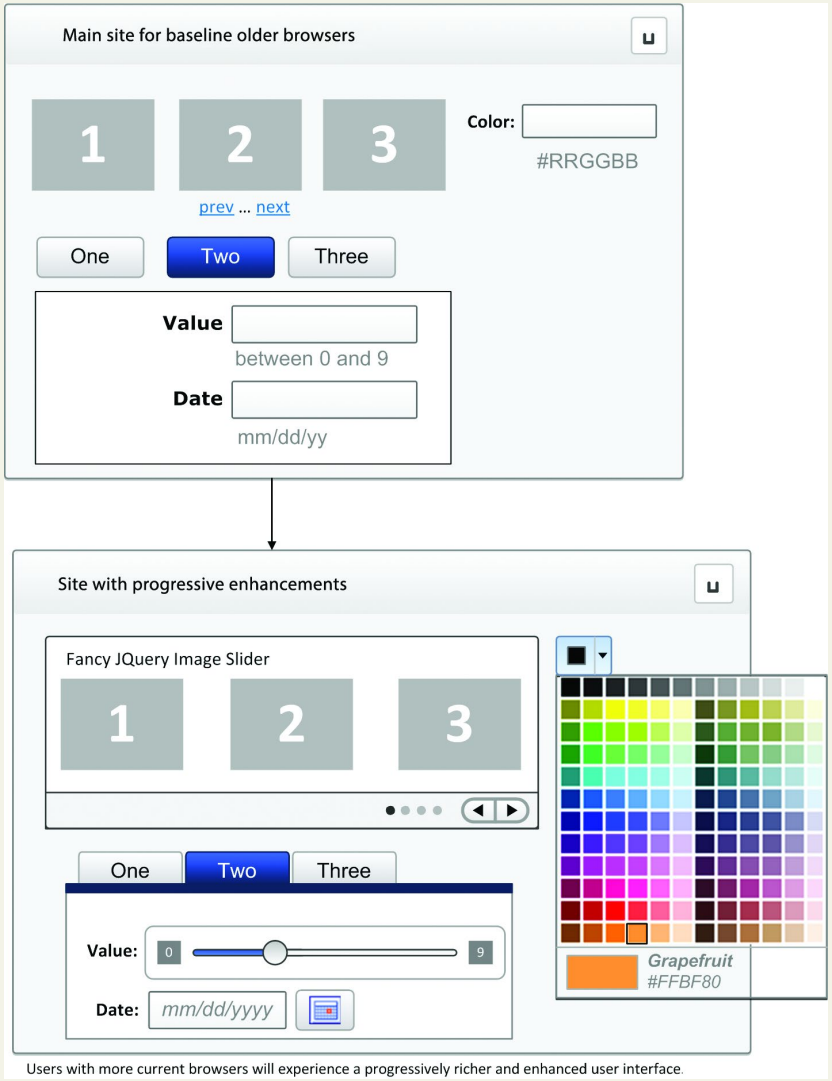


Progressive Enhancement

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.

Progressive Enhancement



Section 3 of 8

WHERE DOES JAVASCRIPT GO?

Where does JavaScript go?

JavaScript can be linked to an HTML page in a number of ways.

- Inline
- Embedded
- External

Inline JavaScript

Mash it in

Inline JavaScript refers to the practice of including JavaScript code directly within certain HTML attributes

Inline JavaScript is a real maintenance nightmare

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

LISTING 6.1 Inline JavaScript example

Embedded JavaScript

Better

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

External JavaScript

Better

JavaScript supports this separation by allowing links to an external file that contains the JavaScript.

By convention, JavaScript external files have the extension .js.

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

LISTING 6.3 External JavaScript example

Advanced Inclusion

In production sites, advanced techniques are used

- Generate embedded styles to reduce requests
  - Code still managed in a external file
- <iframe> loading
- Asynchronus load from another JavaScript file
  - Faster initial load

Section 4 of 8

SYNTAX

JavaScript Syntax

We will briefly cover the fundamental syntax for the most common programming constructs including

- **variables**,
- **assignment**,
- **conditionals**,
- **loops**, and
- **arrays**

before moving on to advanced topics such as **events** and **classes**.

JavaScript's Reputation

Precedes it?

JavaScript's reputation for being quirky not only stems from its strange way of implementing object-oriented principles but also from some odd syntactic *gotchas:*

- 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.

Variables

var

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 we use **var**

Assignment can happen at declaration-time by appending the value to the declaration, or at run time with a simple right-to-left assignment

Variables

Assignment

```
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
```

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

Condition — Value if true — Value if false

Comparison Operators

True or not True

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

Logical Operators

The Boolean operators and, or, and not and their truth tables are listed in Table 6.2. Syntactically they are represented with && (and), || (or), and ! (not).

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

Conditionals

If, else if, …, else

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

```
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

Loops

Round and round we go

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

You will encounter the **while** and **for** loops

While loops normally initialize a **loop control variable** before the loop, use it in the condition, and modify it within the loop.

```
var i=0;  // initialise the Loop Control Variable

while(i < 10){ //test the loop control variable

        i++;  //increment the loop control variable

}
```

For Loops

Counted loops

A **for loop** combines the common components of a loop: initialization, condition, and post-loop operation into one statement.

This statement begins with the **for** keyword and has the components placed between **( )** brackets, semicolon (;) separated as shown

```
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**, which you will learn about later.

They are defined by using the reserved word **function** and then the function name and (optional) parameters.

Since JavaScript is dynamically typed, functions do not require a return type, nor do the parameters require type.

Functions

Example

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

Not really used anymore, console instead

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

alert ("Good Morning");

Using alerts can get tedious fast. When using debugger tools in your browser you can write output to a log with:

console.log("Put Messages Here");

And then use the debugger to access those logs.

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

Throw your own

Exceptions that is.

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

Tips

With Exceptions

Try-catch and throw statements should be used for *abnormal* or *exceptional* cases in your program.

Throwing an exception disrupts the sequential execution of a program. When the exception is thrown all subsequent code is not executed until the catch statement is reached.

This reinforces why try-catch is for exceptional cases.

Section 5 of 8

JAVASCRIPT OBJECTS

JavaScript Objects

Objects not Classes

JavaScript is not a full-fledged object-oriented programming language.

It does not have classes per se, and it does not support many of the patterns you'd expect from an object-oriented language like inheritance and polymorphism.

The language does, however, support objects.

JavaScript Objects

Not full-fledged O.O.

Objects can have **constructors**, **properties**, and **methods** associated with them.

There are objects that are included in the JavaScript language; you can also define your own kind of objects.

Constructors

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

```
var someObject = new ObjectName(p1,p2,..., pn);
```

For some classes, shortcut constructors are defined

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

vs the formal:

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

Properties

Use the dot

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.

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

Methods

Use the dot, with brackets

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 including:

- Array
- Boolean
- Date
- Math
- String
- Dom objects

Arrays

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

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

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

Arrays

Initialize with values

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"];
```

Arrays

Access and Traverse

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

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

One of the most common actions on an array is to traverse through the items sequentially. Using the Array object's **length** property to determine the maximum valid index. We have:

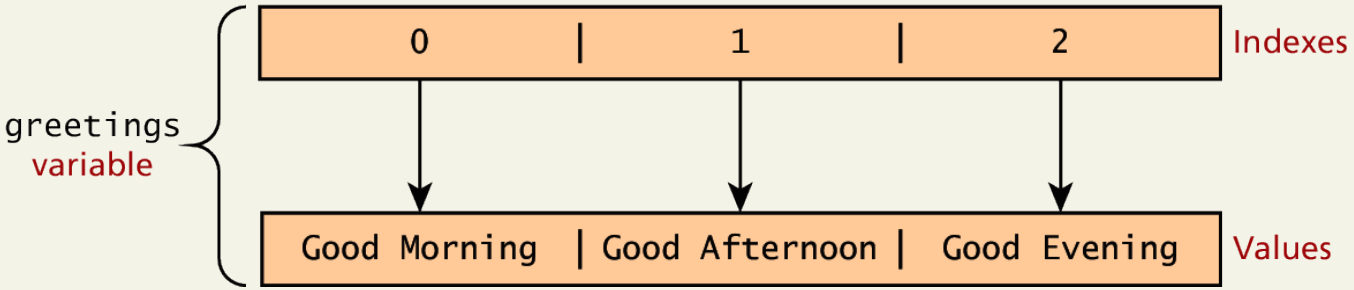
```
for (var i = 0; i < greetings.length; i++){

    alert(greetings[i]);

}
```

Arrays

Index and Value



Arrays

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: 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().

Many mathematical constants are defined such as PI, E, SQRT2, and some others

Math.PI; *// 3.141592657*

Math.sqrt(4); *// square root of 4 is 2.*

Math.random(); *// random number between 0 and 1*

String

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

Constructor usage

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

var greet = "Good"; *// shortcut constructor*

Length of a string

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

String

Concatenation and so much more

```
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()
- searching for one using indexOf().

Strings allow splitting a string into an array, searching and matching with split(), search(), and match() methods.

Date

Not that kind

The Date class is yet another helpful included object you should be aware of. It allows you to quickly calculate the current date or create date objects for particular dates. To display today's date as a string, we would simply create a new object and use the toString() method.

```
var d = new Date();

// This outputs Today is Mon Nov 12 2012 15:40:19 GMT-0700

alert ("Today is "+ d.toString());
```

Window

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.

Section 6 of 8

THE DOCUMENT OBJECT MODEL (DOM)

The DOM

Document Object Model

JavaScript is almost always used to interact with the HTML document in which it is contained.

This is accomplished through a programming interface (API) called the **Document Object Model.**

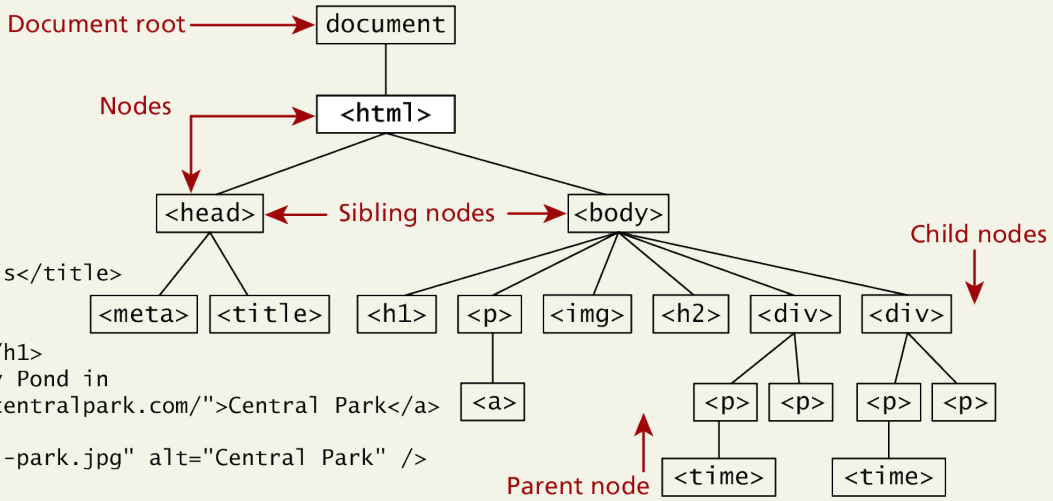
According to the W3C, the DOM is a:

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

The DOM

Seems familiar, because it is!

We already know all about the DOM, but by another name. The tree structure from Chapter 2 (HTML) is formally called the **DOM Tree** with the root, or topmost object called the **Document Root**.




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

DOM 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:

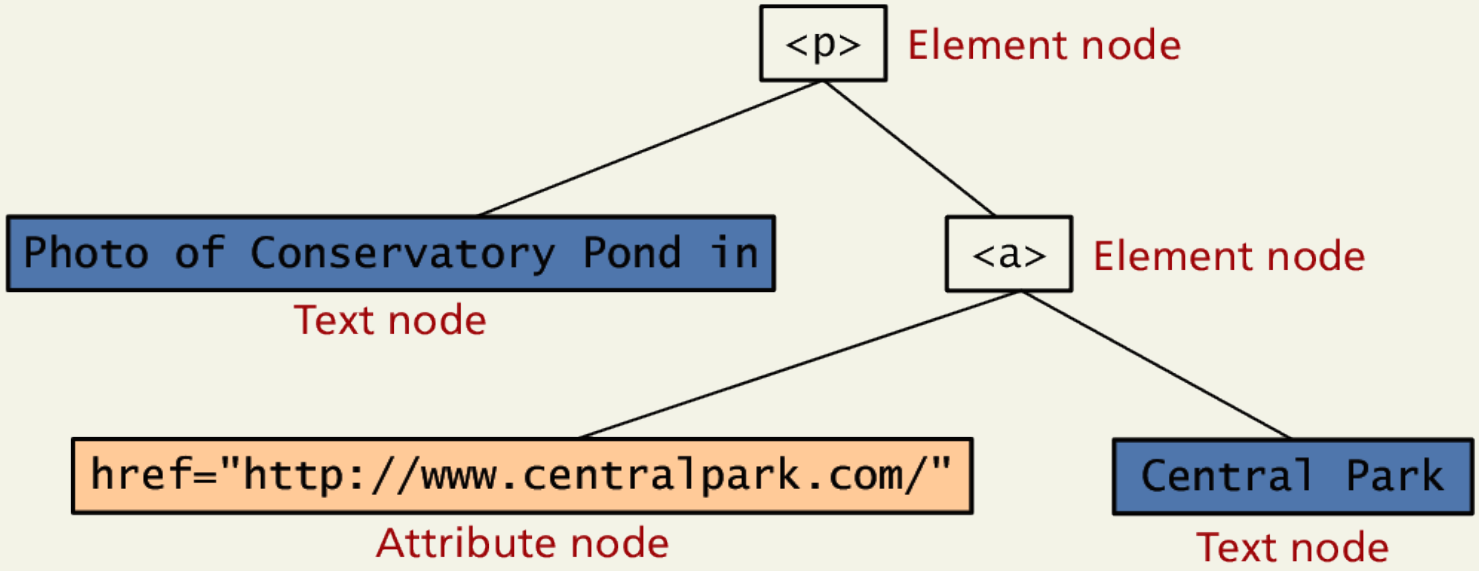
- element nodes,
- text nodes, and
- attribute nodes

All nodes in the DOM share a common set of properties and methods.

DOM Nodes

Element, text and attribute nodes

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



DOM Nodes

Essential Node Object properties

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.

Document Object

One root to ground them all

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**.

```
// specify the doctype, for example html

var a = document.doctype.name;

// specify the page encoding, for example ISO-8859-1

var b = document.inputEncoding;
```

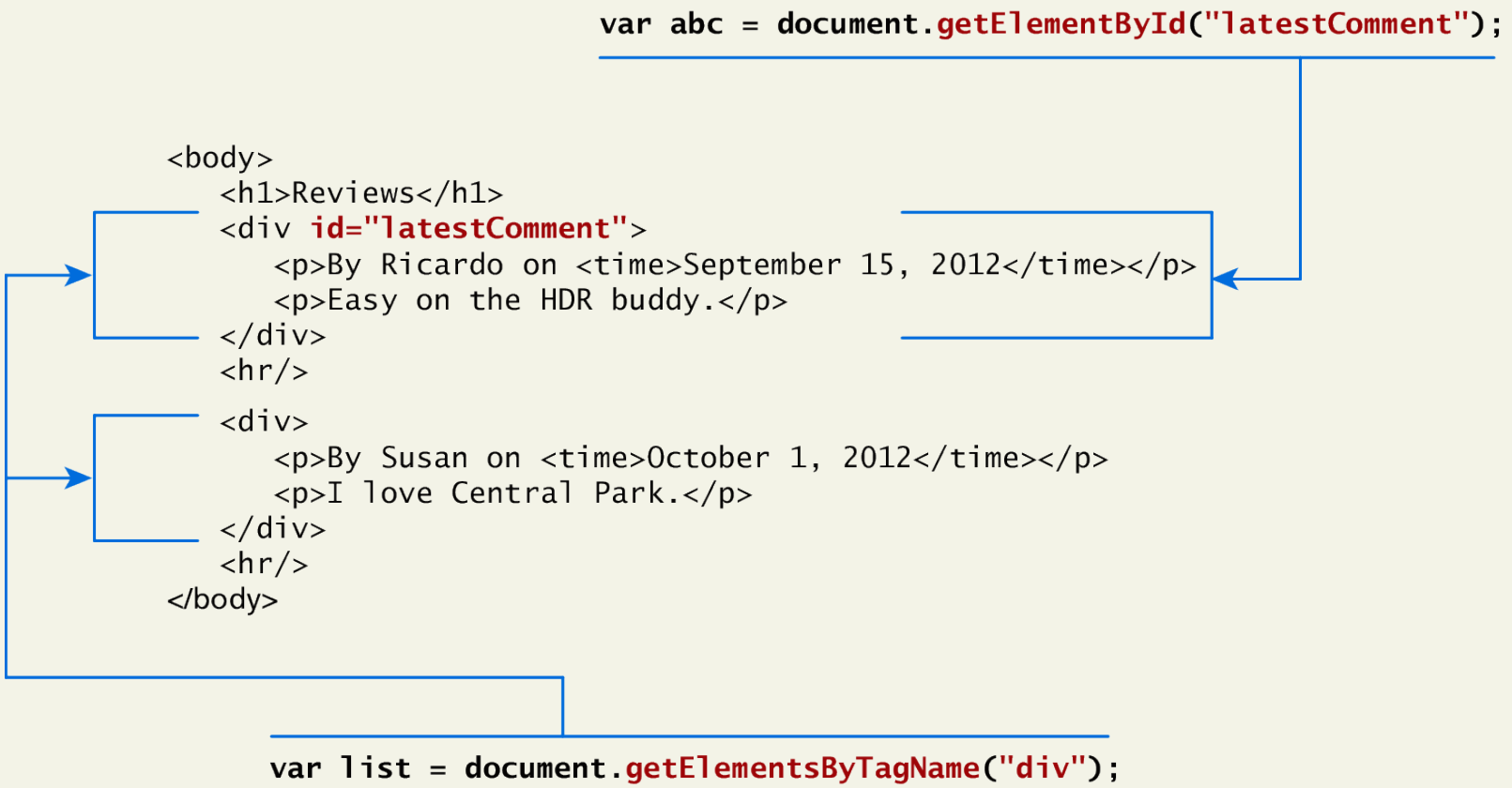
Document Object

Document Object Methods

Method	Description
createAttribute()	Creates an attribute node
createElement()	Creates an element node
createTextNode()	Create 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.

Accessing nodes

getElementById(), getElementsByTagName()



Element node Object

The type of object returned by the method document.getElementById() described in the previous section is an **element node** object.

This represents an HTML element in the hierarchy, contained between the opening <> and closing </> tags for this element.

- can itself contain more elements

Element node Object

Essential Element Node Properties

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 which we update particular div's using JS.
style	The style attribute of an element. We can read and modify this property.
tagName	The tag name for the element.

Modifying a DOM element

The document.write() method is used to create output to the HTML page from JavaScript. 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

Modifying a DOM element

More verbosely, and validated

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

DOM functions createTextNode(), 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 add or remove any style using the **style** or **className** property of the Element node.

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";
```

Changing an element's style

With class

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.

```
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.

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

More Properties

Some Specific HTML DOM Element Properties for Certain Tag Types

Property	Description	Tags
href	The href attribute used in a tags 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 tot he 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

Section 7 of 8

JAVASCRIPT EVENTS

JavaScript Events

A JavaScript **event** is an action that can be detected by JavaScript.

We say then that an event is *triggered* and then it can be *caught* by JavaScript functions, which then do something in response.

JavaScript Events

A brave new world

In the original JavaScript world, events could be specified right in the HTML markup with *hooks* to the JavaScript code (and still can).

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

JavaScript Events

Two approaches

Old, Inline technique

```
...
 <script type="text/javascript"  src="inline.js"></script>
...

<form name='mainForm' 'onsubmit="validate(this);">
      <input name="name" type="text" onhover="hover(this);" onfocus="focus(this);">
      <input name="email" type="text" onhover="hover(this);" onfocus="focus(this);">
      <input type="submit" onclick="validate(this);">
...
```

New, Layered Listener technique

```
...
 <script type="text/javascript"  src="listener.js"></script>
...

<form name='mainForm'>
      <input name="name" type="text">
      <input name="email" type="text">
      <input type="submit">
...
```

Inline Event Handler Approach

For example, if you wanted an alert to pop-up when clicking a you might program:

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

The problem with this type of programming is that the HTML markup and the corresponding JavaScript logic are woven together. It does not make use of layers; that is, it does not separate content from behavior.

Listener Approach

Two ways to set up listeners

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

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

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

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

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

Listener Approach

Using functions

What if we wanted to do something more elaborate when an event is triggered? In such a case, the behavior would have to be encapsulated within a function, as shown in Listing 6.12.

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

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

LISTING 6.12 Listening to an event with a function

Listener Approach

Anonymous functions

An alternative to that shown in Listing 6.12 is to use an anonymous function (that is, one without a name), as shown in Listing 6.13.

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

LISTING 6.13 Listening to an event with an anonymous function

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.
}
```

Event Object

Several Options

- **Bubbles**. 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.
- **Cancelable**. The Cancelable property is also a Boolean value that indicates whether or not the event can be cancelled.
- **preventDefault**. A cancelable default action for an event can be stopped using the preventDefault() method in the next slide

Event Object

Prevent the default behaviour

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

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

Event Types

There are several classes of event, with several types of event within each class specified by the W3C:

- mouse events
- keyboard events
- form events
- frame events

Mouse events

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

Keyboard events

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)

Keyboard events

Example

<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

Form Events

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 pre-validation when the user submits the form in JavaScript before sending the data on to the server.

Form Events

Example

```
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 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. If the code attempts to set up a listener on this not-yet-loaded , then an error will be triggered.

```
window.onload= function(){

        //all JavaScript initialization here.

}
```

Frame Events

Table of frame events

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

Section 8 of 8

FORMS

Validating Forms

You mean pre-validating right?

Writing code to prevalidate forms on the client side will reduce the number of incorrect submissions, thereby reducing server load.

There are a number of common validation activities including email validation, number validation, and data validation.

Validating Forms

Empty field

```
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

Validating Forms

Empty field

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

}
```

Validating Forms

Number Validation

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

LISTING 6.19 A function to test for a numeric value

Validating Forms

More to come in Chapter 12

Form validation uses regular expressions, covered in Chapter 12

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.

What you Learned

1. What is JavaScript
2. JavaScript Design
3. Using JavaScript
4. Syntax
5. JavaScript Objects
6. The DOM
7. JavaScript Events
8. Forms