JAVASCRIPT – CLIENT SIDE SCRIPTING

Course Outline

1. Introduction

   What is Javascript (Client side scripting, event driven programming, advantages/disadvantages, what you can do by using Javascript)

2. Document Object Model

   Methods for accessing elements on DOM (getElementbyID, getElementsbyTagname, querySelector, arrays-form example)

   Methods for changing content (innerHTML, textContent, value)

   Methods for changing style (style.color, className)

   Methods for traversing through DOM and adding and removing elements on DOM

3. Unobtrusive Javascript

   Separating JS from HTML and CSS

   Anonymous functions

   Event handlers, binding event handlers to events (by assignment, addEventListener)

4. Advanced Topics (MDN – Intermediate and Advanced Javascript)

   https://developer.mozilla.org/en-US/docs/Web/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.
  - 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*,

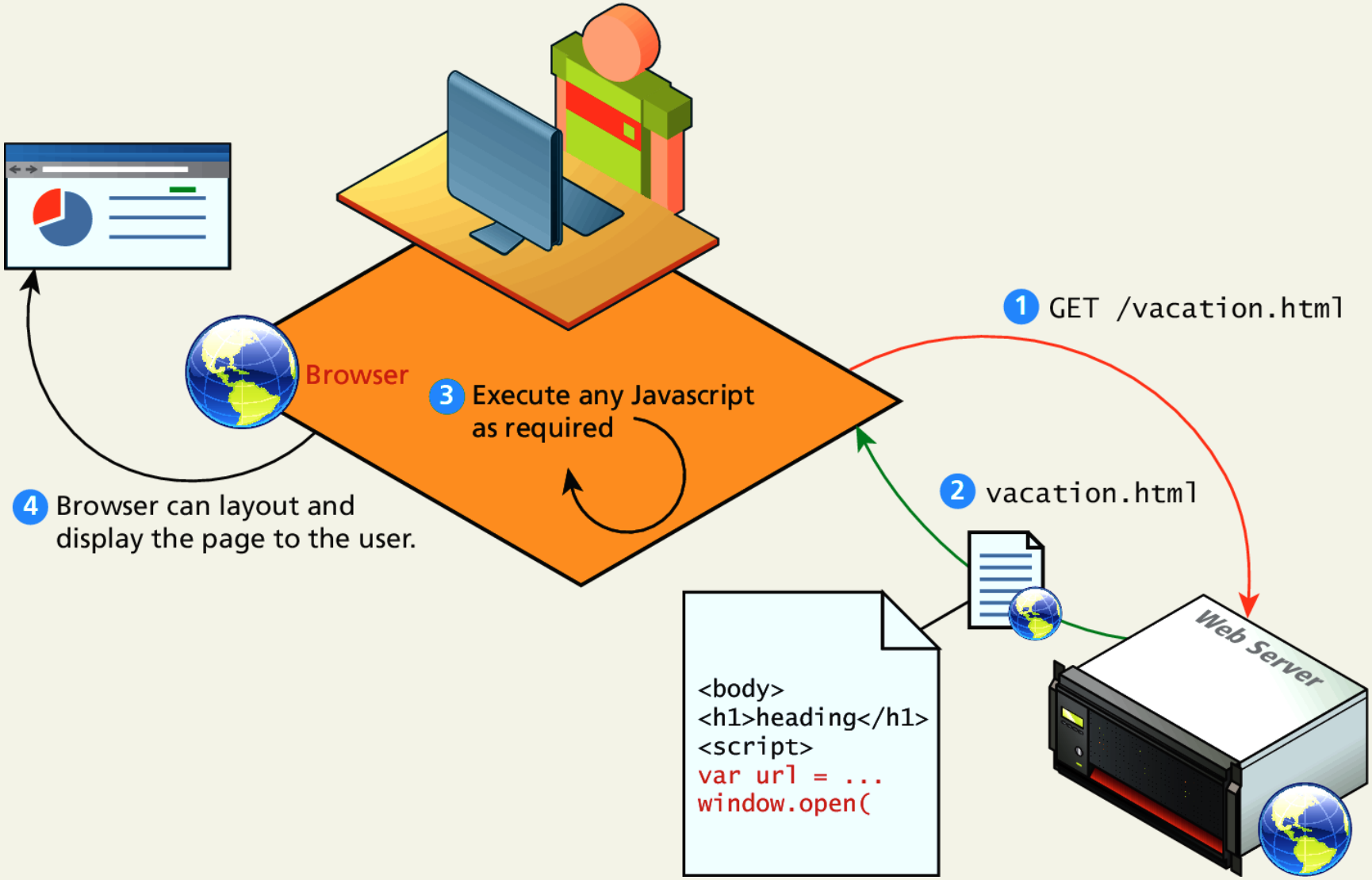
> JavaScript and Java are vastly different programming languages with different uses.

Java is a full-fledged compiled, object-oriented language, popular for its ability to run on any platform with a JVM installed.

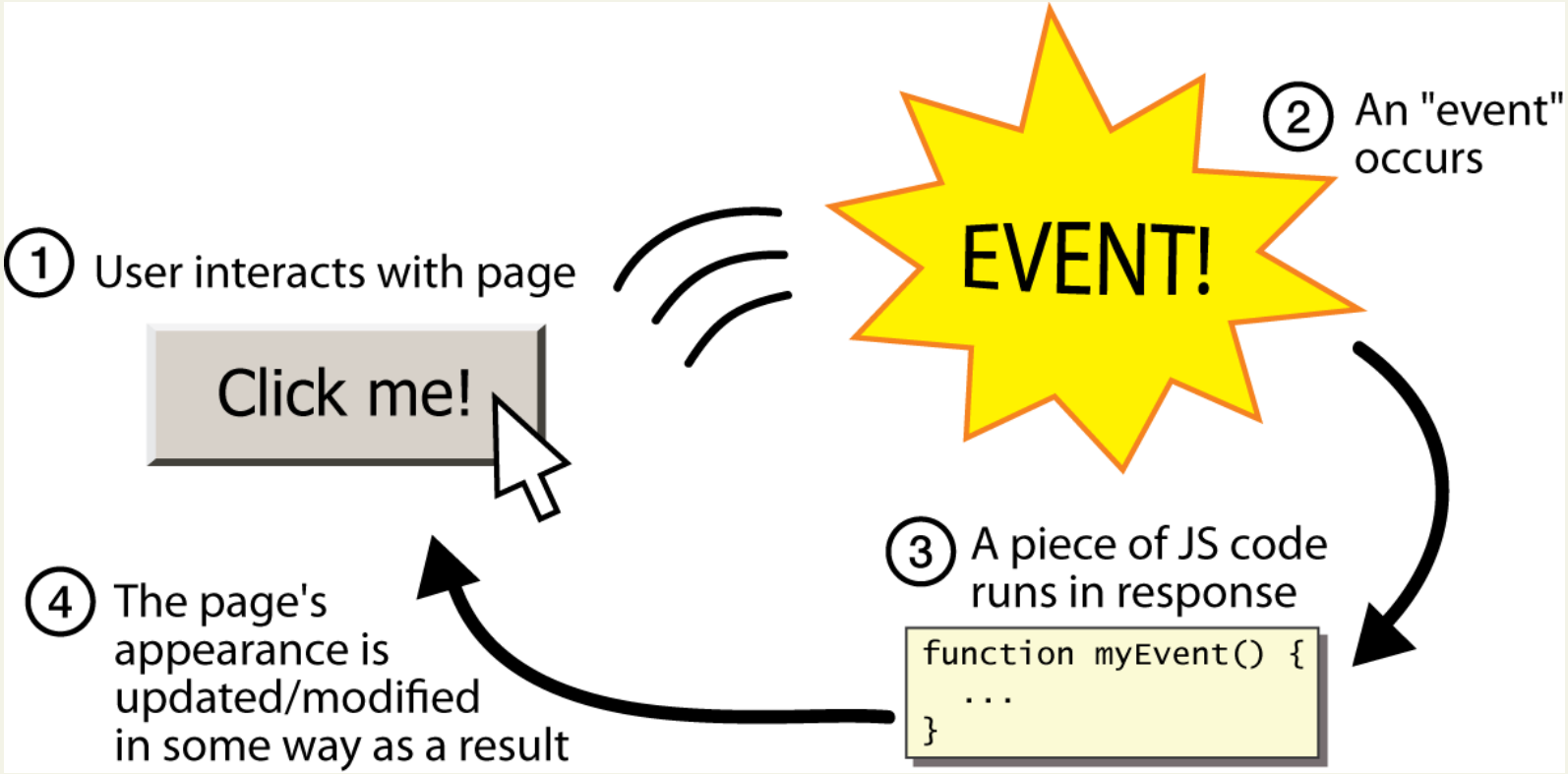
JavaScript is one of the world's most popular languages, with fewer of the object-oriented features of Java, and runs directly inside the browser, without the need for the JVM.

Client-Side Scripting

Let the client compute



Event driven programming



Client-Side Scripting

It's good

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

- Processing can be off-loaded from the server to client, thereby reducing the workload 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.

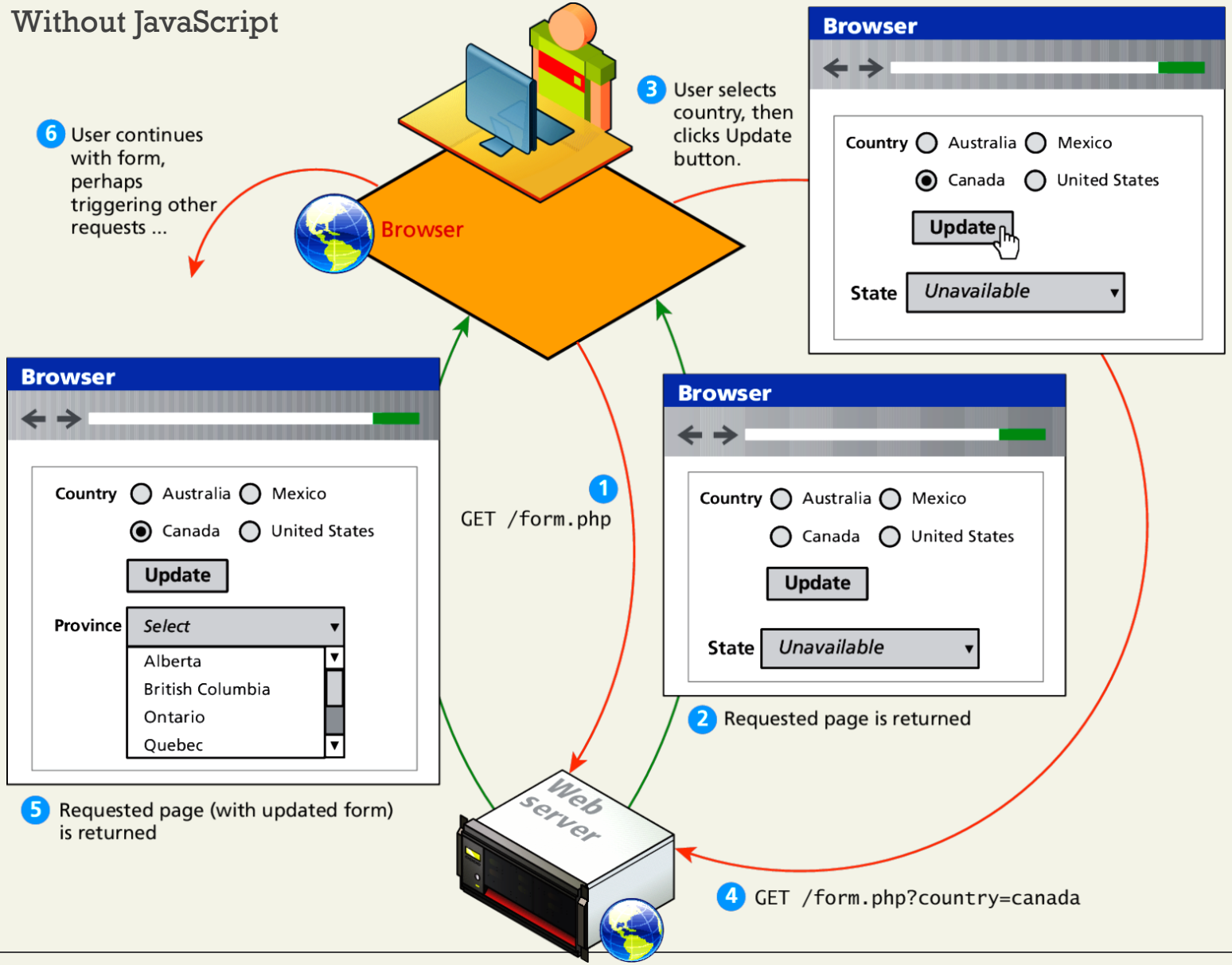
- There is no guarantee that the client has JavaScript enabled
- What works in one browser, may generate an error in another.
- JavaScript-heavy web applications can be complicated to debug and maintain.

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**

HTTP request-response loop

Without JavaScript



JavaScript in Modern Times

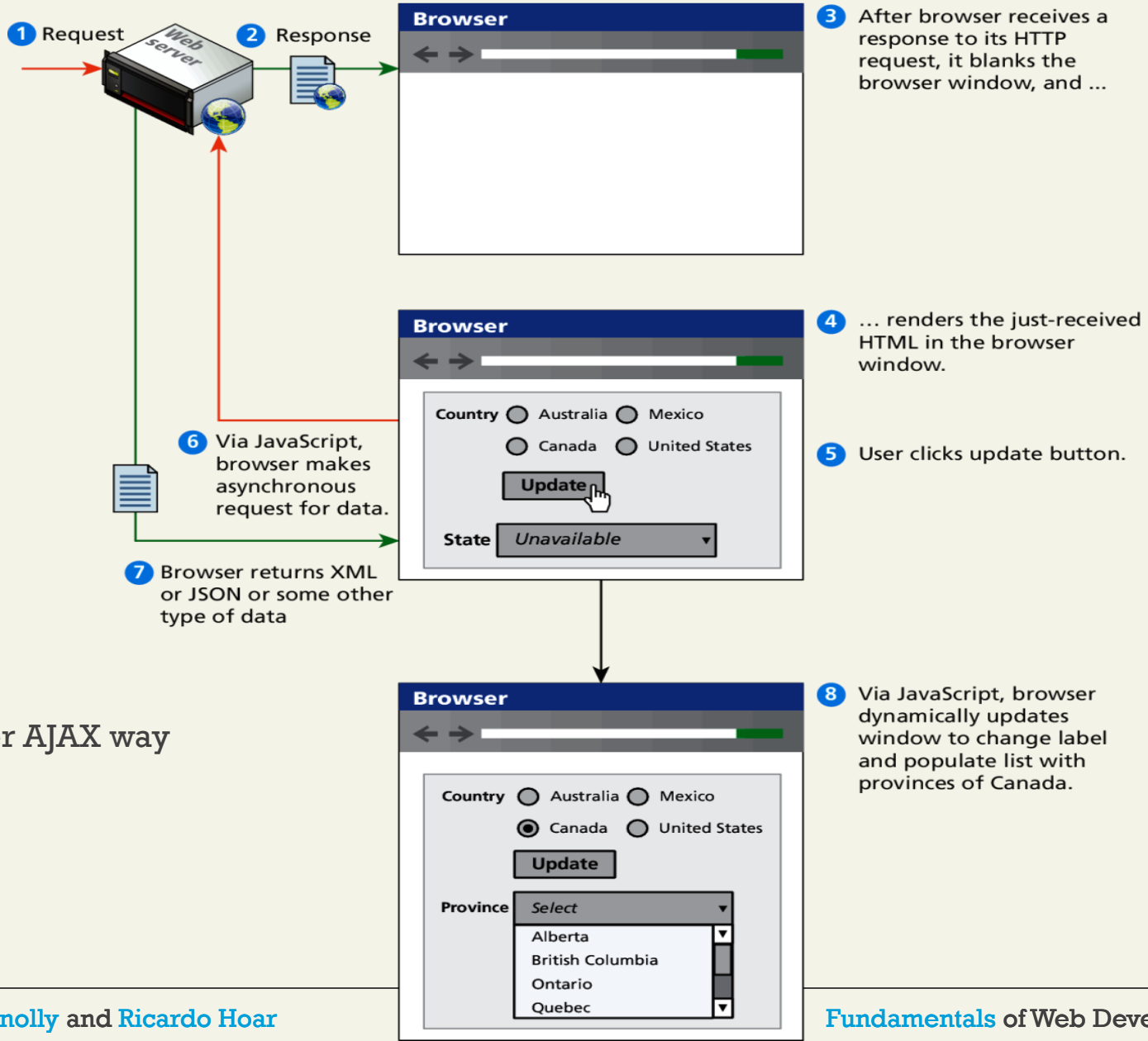
AJAX

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.

- **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

WHAT JAVASCRIPT CAN DO? EXAMPLES

What JavaScript can do?

There are three main things we can do via JavaScript:

- Change the contents of the page (change content of an element, add/remove elements from the page)
- Change the appearance of the page (change style of elements)
- react to user events (mouse clicks, key presses, form submission etc.)

Javascript – example 1

Reference http://www.w3schools.com/jsref/default.asp

```
<!DOCTYPE html>
<html>
<body>
<h1>What Can JavaScript Do?</h1>
<p id="demo">JavaScript can change HTML content.</p>
<button type="button"
        onclick="document.getElementById('demo').innerHTML
         = 'Hello JavaScript!'">
Click Me!</button>
</body>
</html>
```

Javascript – example 1

Reference http://www.w3schools.com/jsref/default.asp


```
<!DOCTYPE html>
<html>
<body>
<h1>What Can JavaScript Do?</h1>
<p id="demo">JavaScript can change HTML content.</p>
<button type="button"
        onclick="document.getElementById('demo').innerHTML
         = 'Hello JavaScript!'">
Click Me!</button>
</body>
</html>
```

Javascript – example 2

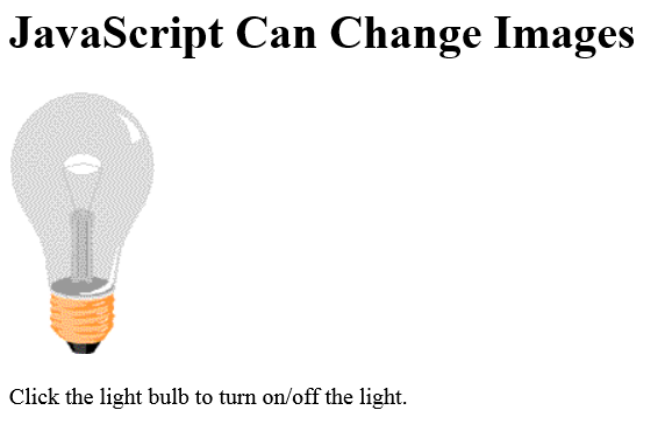
Reference http://www.w3schools.com/jsref/default.asp

```
<!DOCTYPE html>
<html> <body>

<h1>JavaScript Can Change Images</h1>
<img id="myImage" onclick="changeImage()"
          src="pic_bulboff.gif" width="100" height="180">
<p>Click the light bulb to turn on/off the light.</p>
    <script>
    function changeImage() {
       var image = document.getElementById('myImage');
       if (image.src.match("bulbon")) {
          image.src = "pic_bulboff.gif";
       } else {
          image.src = "pic_bulbon.gif";
       }
    }
    </script>
 </body> </html>
```

Javascript – example 2

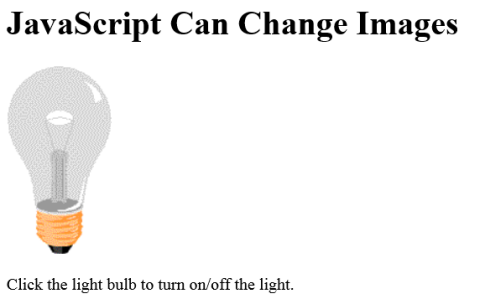
Reference http://www.w3schools.com/jsref/default.asp

```
<!DOCTYPE html>
<html> <body>

<h1>JavaScript Can Change Images</h1>
<img id="myImage" onclick="changeImage()"
          src="pic_bulboff.gif" width="100" height="180">
<p>Click the light bulb to turn on/off the light.</p>
    <script>
    function changeImage() {
       var image = document.getElementById('myImage');
       if (image.src.match("bulbon")) {
          image.src = "pic_bulboff.gif";
       } else {
          image.src = "pic_bulbon.gif";
       }
    }
    </script>
 </body> </html>
```

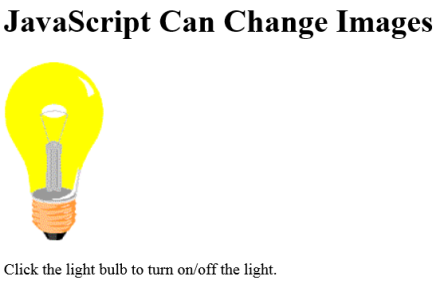


Javascript - example

Reference http://www.w3schools.com/jsref/default.asp




```
<!DOCTYPE html>
<html> <body>

<h1>JavaScript Can Change Images</h1>
<img id="myImage" onclick="changeImage()"
        src="pic_bulboff.gif" width="100" height="180">
<p>Click the light bulb to turn on/off the light.</p>
    <script>
    function changeImage() {
      var image = document.getElementById('myImage');
      if (image.src.match("bulbon")) {
        image.src = "pic_bulboff.gif";
      } else {
        image.src = "pic_bulbon.gif";
      }
    }
    </script>
 </body> </html>
```



Javascript – example 3

Reference http://www.w3schools.com/jsref/default.asp

```
<!DOCTYPE html>
<html>
<body>

<h1>What Can JavaScript Do?</h1>

<p id="demo">JavaScript can change the style of an HTML
element.</p>

<script>
function myFunction() {
   var x = document.getElementById("demo");
   x.style.fontSize = "25px";
   x.style.color = "red";
}
</script>
```

Javascript – example 3

Reference http://www.w3schools.com/jsref/default.asp

```
<!DOCTYPE html>
<html>
<body>

<h1>What Can JavaScript Do?</h1>

<p id="demo">JavaScript can change the style of an HTML element.</p>

<script>
function myFunction() {
   var x = document.getElementById("demo");
   x.style.fontSize = "25px";
   x.style.color = "red";
}
</script>
```

Javascript – example 3

Reference http://www.w3schools.com/jsref/default.asp

```
<!DOCTYPE html>
<html>
<body>

<h1>What Can JavaScript Do?</h1>

<p id="demo">JavaScript can change the style of an HTML element.</p>

<script>
function myFunction() {
   var x = document.getElementById("demo");
   x.style.fontSize = "25px";
   x.style.color = "red";
}
</script>
```

Javascript – example 4

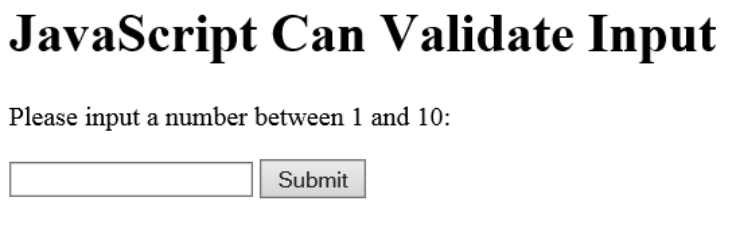
Reference http://www.w3schools.com/jsref/default.asp

```
<!DOCTYPE html> <html> <body>
<h1>JavaScript Can Validate Input</h1>
<p>Please input a number between 1 and 10:</p>
<input id="numb" type="number">
<button type="button" onclick="myFunction()">Submit</button>
<p id="demo"></p>
<script>
function myFunction() {
   var x, text;
   // Get the value of the input field with id="numb"
   x = document.getElementById("numb").value;
   // If x is Not a Number or less than one or greater than 10
   if (isNaN(x) || x < 1 || x > 10) {
      text = "Input not valid";
   } else {
      text = "Input OK";
   }
   document.getElementById("demo").innerHTML = text;
}
</script> </body> </html>
```

Javascript – example 4

Reference http://www.w3schools.com/jsref/default.asp

```
<!DOCTYPE html> <html> <body>
<h1>JavaScript Can Validate Input</h1>
<p>Please input a number between 1 and 10:</p>
<input id="numb" type="number">
<button type="button" onclick="myFunction()">Submit</button>
<p id="demo"></p>
<script>
function myFunction() {
   var x, text;
   // Get the value of the input field with id="numb"
   x = document.getElementById("numb").value;
   // If x is Not a Number or less than one or greater than 10
   if (isNaN(x) || x < 1 || x > 10) {
      text = "Input not valid";
   } else {
      text = "Input OK";
   }
   document.getElementById("demo").innerHTML = text;
}
</script> </body> </html>
```



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

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**
- **Arrays**

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.

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.

Alert

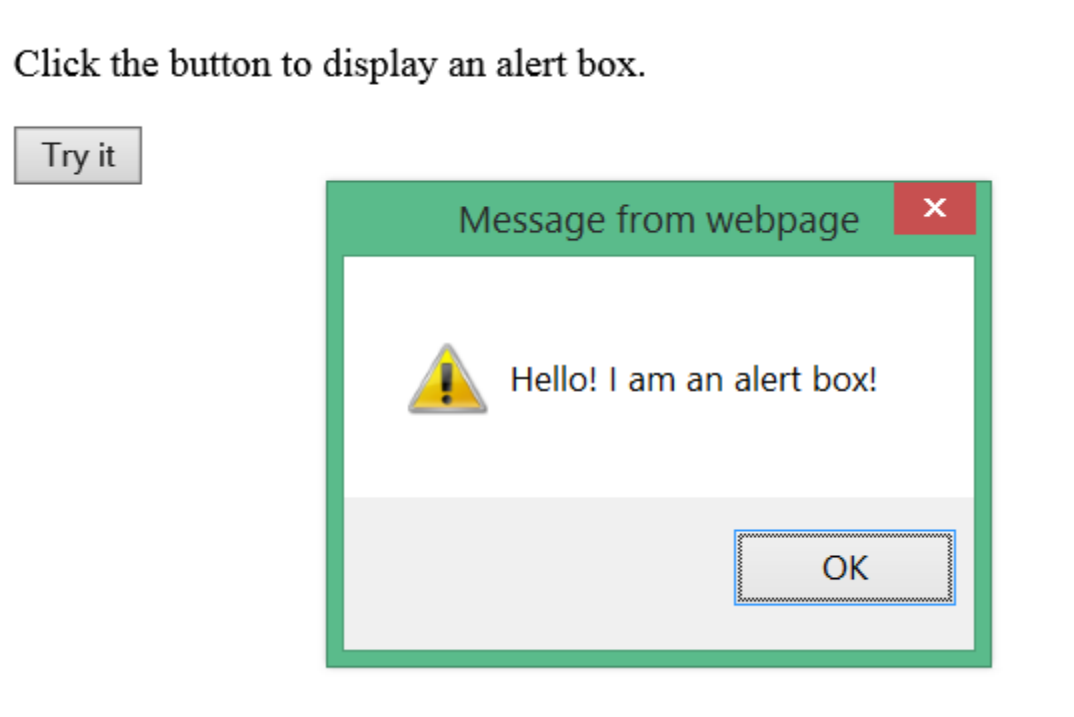
Not really used anymore, console instead

```
<!DOCTYPE html>
<html>
<body>
<p>Click the button to display an alert box.</p>
      <button onclick="myFunction()">Try it</button>
<script>
   function myFunction() {
        alert("Hello! I am an alert box!");
   }
</script>
</body>
</html>
```

Alert

Not really used anymore, console instead

```
<!DOCTYPE html>
<html>
<body>
<p>Click the button to d
  <button onclick="myF
<script>
   function myFunction(
     alert("Hello! I am an
   }
</script>
</body>
</html>
```



Arrays

```
var name = []; // empty array
var name = [value, value, ..., value]; // pre-filled
name[index] = value; // store element
```

JS

```
var ducks = ["Huey", "Dewey", "Louie"];
var stooges = []; // stooges.length is 0
stooges[0] = "Larry"; // stooges.length is 1
stooges[1] = "Moe"; // stooges.length is 2
stooges[4] = "Curly"; // stooges.length is 5
stooges[4] = "Shemp"; // stooges.length is 5
```

JS

Array methods

```
var a = ["Stef", "Jason"]; // Stef, Jason
a.push("Brian"); // Stef, Jason, Brian
a.unshift("Kelly"); // Kelly, Stef, Jason, Brian
a.pop(); // Kelly, Stef, Jason
a.shift(); // Stef, Jason
a.sort(); // Jason, Stef
```

JS

- array serves as many data structures: list, queue, stack, ...
- methods: `concat, join, pop, push, reverse, shift, slice, sort, splice, toString, unshift`
  - push and pop add / remove from back
  - unshift and shift add / remove from front
  - shift and pop return the element that is removed

String type

```
var s = "Connie Client";
var fName = s.substring(0, s.indexOf(" ")); // "Connie"
var len = s.length; // 13
var s2 = 'Melvin Merchant';
```

JS

- methods: charAt, charCodeAt, fromCharCode, indexOf, lastIndexOf, replace, split, substring, toLowerCase, toUpperCase
  - charAt returns a one-letter String (there is no char type)
- length property (not a method as in Java)
- Strings can be specified with "" or ''
- concatenation with + :
  - 1 + 1 is 2, but "1" + 1 is "11"

More about `String`

- escape sequences behave as in Java: \' \" \& \n \t \\
- converting between numbers and Strings:

```
var count = 10;
var s1 = "" + count; // "10"
var s2 = count + " bananas, ah ah ah!"; // "10 bananas, ah ah ah!"
var n1 = parseInt("42 is the answer"); // 42
var n2 = parseFloat("booyah"); // NaN
```
JS

- accessing the letters of a String:

```
var firstLetter = s[0]; // fails in IE
var firstLetter = s.charAt(0); // does work in IE
var lastLetter = s.charAt(s.length - 1);
```
JS

Splitting strings: split and join

```
var s = "the quick brown fox";
var a = s.split(" "); // ["the", "quick", "brown", "fox"]
a.reverse(); // ["fox", "brown", "quick", "the"]
s = a.join("!"); // "fox!brown!quick!the"
```
JS

- split breaks apart a string into an array using a delimiter
  - can also be used with regular expressions (seen later)
- join merges an array into a single string, placing a delimiter between them

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 6 of 8

THE DOCUMENT OBJECT MODEL (DOM)

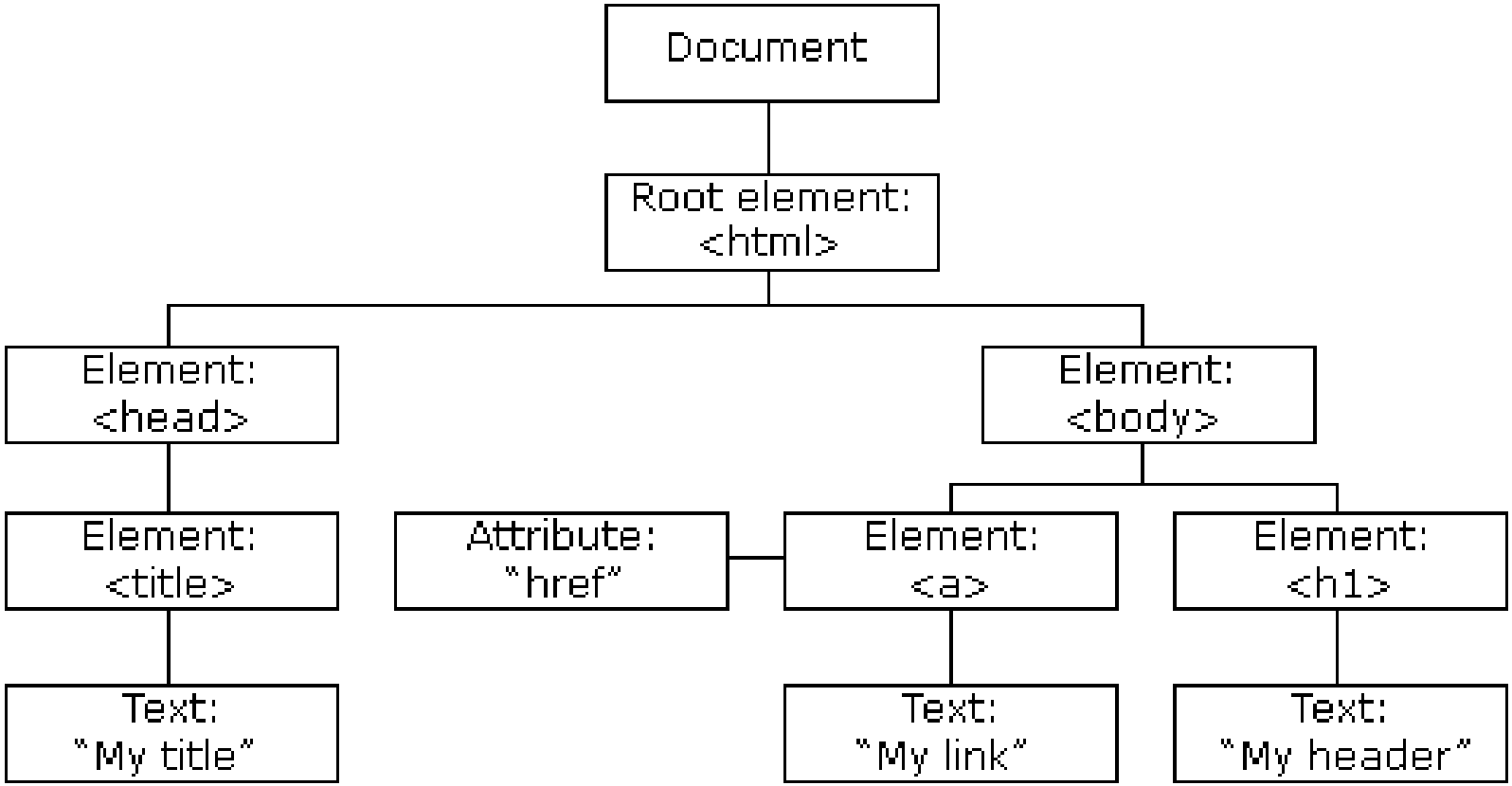
Randy Connolly and Ricardo Hoar

The DOM

Document Object Model

The HTML DOM (Document Object Model)

When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page. The **HTML DOM** model is constructed as a tree of **Objects**:



The DOM

Document Object Model

With the object model, JavaScript gets all the power it needs to create dynamic HTML:

JavaScript can react to all existing HTML events in the page

JavaScript can create new HTML events in the page

JavaScript can add new HTML elements and attributes

JavaScript can change all the HTML elements in the page

JavaScript can change all the HTML attributes in the page

JavaScript can change all the CSS styles in the page

JavaScript can remove existing HTML elements and attributes

The DOM

Document Object Model

With the object model, JavaScript gets all the power it needs to create dynamic HTML.

To know, using Javascript,

- How to change the content of HTML elements (for example):
  - How to change the style (CSS) of HTML elements
  - How to react to HTML DOM events
  - How to add and delete HTML elements

The DOM

Document Object Model

What is the DOM?

The DOM is a W3C (World Wide Web Consortium) standard.

The DOM defines a standard for accessing documents:

"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

The W3C DOM standard is separated into 3 different parts:

Core DOM - standard model for all document types

XML DOM - standard model for XML documents

HTML DOM - standard model for HTML documents

The DOM

Document Object Model

The DOM Programming Interface

The HTML DOM can be accessed with JavaScript (and with other programming languages).

In the DOM, all HTML elements are defined as **objects**.

The programming interface is the properties and methods of each object.

A **property** is a value that you can get or set (like changing the content of an HTML element).

A **method** is an action you can do (like add or deleting an HTML element).

The DOM

Document Object Model

The following example changes the content (the innerHTML) of the <p> element with id="demo":

```
<html>
<body>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script>

</body>
</html>
```

// In the example above, getElementById is a **method**, while innerHTML is a **property**.

The DOM

Document Object Model

The getElementById Method

The most common way to access an HTML element is to use the id of the element.

In the example above the **getElementById** method used id="demo" to find the element.

```
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script>
```

The DOM

Document Object Model

The innerHTML Property

The easiest way to get the content of an element is by using the **innerHTML** property.

The innerHTML property is useful for getting or replacing the content of HTML elements.

The innerHTML property can be used to get or change any HTML element, including <html> and <body>.

The DOM

Document Object Model

The HTML DOM Document

In the HTML DOM object model, the **document** object represents your web page.

The **document** object is the owner of all other objects in your web page.

If you want to access objects in an HTML page, you always start with accessing the **document** object.

How can you use the **document** object to access and manipulate HTML via document methods?

The DOM

Document Object Model

Finding HTML Elements

Method	Description
document.getElementById()	Find an element by element id
document.getElementsByTagName()	Find elements by tag name
document.getElementsByClassName()	Find elements by class name

The DOM

Document Object Model

Changing HTML Elements

Changing HTML Elements

Method	Description
element.innerHTML=	Change the inner HTML of an element
element.*attribute*=	Change the attribute of an HTML element
element.setAttribute(*attribute*,*value*)	Change the attribute of an HTML element
element.style.*property*=	Change the style of an HTML element

The DOM

Document Object Model

Adding and Deleting Elements

Method	Description
document.createElement()	Create an HTML element
document.removeChild()	Remove an HTML element
document.appendChild()	Add an HTML element
document.replaceChild()	Replace an HTML element
document.write(*text*)	Write into the HTML output stream

The DOM

Document Object Model

Adding Events Handlers

Method	Description
document.getElementById (*id*).onclick=function(){*code*}	Adding event handler code to an onclick event

The DOM

Document Object Model

Finding HTML Objects - Elements

Often, with JavaScript, you want to manipulate HTML elements.

To do so, you have to find the elements first. There are a couple of ways to do this:

Finding HTML elements by id

Finding HTML elements by tag name

Finding HTML elements by class name

Finding HTML elements by CSS selectors

Finding HTML elements by HTML object collections

The DOM

Document Object Model

Finding HTML Objects - Elements

Often, with JavaScript, you want to manipulate HTML elements.

To do so, you have to find the elements first. There are a couple of ways to do this:

Finding HTML elements by id

Finding HTML elements by tag name

Finding HTML elements by class name

Finding HTML elements by CSS selectors

Finding HTML elements by HTML object collections

The DOM

Document Object Model

Finding HTML Element by Id

The easiest way to find an HTML element in the DOM, is by using the element id.

This example finds the element with id="intro":

```
var x = document.getElementById("intro");
```

If the element is found, the method will return the element as an object (in x).

If the element is not found, x will contain null.

The DOM

Document Object Model

Finding HTML Element by Id

The easiest way to find an HTML element in the DOM, is by using the element id.

This example finds the element with id="intro":

```
var x = document.getElementById("intro");
```

If the element is found, the method will return the element as an object (in x).

If the element is not found, x will contain null.

The DOM

Document Object Model

Finding HTML Elements by Tag Name

This example finds all <p> elements:

```
var x = document.getElementsByTagName("p");
```

This example finds the element with id="main", and then finds all <p> elements inside "main":

```
var x = document.getElementById("main");
var y = x.getElementsByTagName("p");
```

The DOM

Document Object Model

Finding HTML Elements by Class Name

If you want to find all HTML elements with the same class name, use getElementsByClassName().

This example returns a list of all elements with class="intro".

```
var x = document.getElementsByClassName("intro");
```

The DOM

Document Object Model

Finding HTML Elements by CSS Selectors

If you want to find all HTML elements that matches a specified CSS selector (id, class names, types, attributes, values of attributes, etc), use the querySelectorAll() method.

This example returns a list of all <p> elements with class="intro".

```
var x = document.querySelectorAll("p.intro");
```

The querySelectorAll() method does not work in Internet Explorer 8 and earlier versions.

The DOM

Document Object Model

Finding HTML Elements by HTML Object Collections

This example finds the form element with id="frm1", in the forms collection, and displays all element values:

```
var x = document.forms["frm1"];
var text = "";
var i;

for (i = 0; i < x.length; i++) {
   text += x.elements[i].value + "<br>";
}
document.getElementById("demo").innerHTML = text;
```

The DOM

Document Object Model

The following HTML objects (and object collections) are also accessible:

- document.anchors
- document.body
- document.documentElement
- document.embeds
- document.forms
- document.head
- document.images
- document.links
- document.scripts
- document.title

The DOM

Document Object Model

Changing the HTML Output Stream

JavaScript can create dynamic HTML content:

Date: Thu Jun 18 2015 10:12:25 GMT-0500 (Central Daylight Time)

In JavaScript, document.write() can be used to write directly to the HTML output stream:

```
<!DOCTYPE html>
<html>
<body>

<script>
document.write(Date());
</script>

</body>
</html>
```

The DOM

Document Object Model

Changing HTML Content

The easiest way to modify the content of an HTML element is by using the **innerHTML** property. To change the content of an HTML element, use this syntax:

document.getElementById(*id*).innerHTML = *new HTML*

This example changes the content of a <p> element:

```
<html>
<body>

<p id="p1">Hello World!</p>

<script>
document.getElementById("p1").innerHTML = "New text!";
</script>

</body>
</html>
```

The DOM

Document Object Model

Changing HTML Content

```
<!DOCTYPE html>
<html>
<body>
<p id="p1">Hello World!</p>
<script>
document.getElementById("p1").innerHTML = "New text!";
</script>
<p>The paragraph above was changed by a script.</p>
</body>
</html>
```

New text!

The paragraph above was changed by a script.

The DOM

Document Object Model

Changing HTML Content

This example changes the content of an <h1> element:

```
<!DOCTYPE html>
<html>
<body>
<h1 id="header">Old Header</h1>
<script>
var element = document.getElementById("header");
element.innerHTML = "New Header";
</script>
</body>
</html>
```

Example explained:

The HTML document above contains an <h1> element with id="header"

We use the HTML DOM to get the element with id="header"

A JavaScript changes the content (innerHTML) of that element

The DOM

Document Object Model

Changing the Value of an Attribute

To change the value of an HTML attribute, use this syntax:

document.getElementById(*id*).*attribute=new value*

This example changes the value of the src attribute of an <img> element:

```
<!DOCTYPE html>
<html>
<body>
<img id="myImage" src="smiley.gif">
<script>
        document.getElementById("myImage").src = "landscape.jpg";
</script>
</body>
</html>
```

The DOM

Document Object Model

Changing HTML Style

To change the style of an HTML element, use this syntax:

document.getElementById(*id*).style.*property=new style*

The following example changes the style of a <p> element:

```
<html>
<body>
<p id="p2">Hello World!</p>

<script>
        document.getElementById("p2").style.color = "blue";
</script>

<p>The paragraph above was changed by a script.</p>
</body>
</html>
```

The DOM

Document Object Model

Style object

The Style object represents an individual style statement.

Access a Style Object

The Style object can be accessed from the head section of the document, or from specific HTML element(s).

Accessing style object(s) from the head section of the document:

```
var x = document.getElementsByTagName("STYLE");
```

Accessing a specified element's style object:

```
var x = document.getElementById("myH1").style;
```

The DOM

Document Object Model

Create a Style Object

You can create a <style> element by using the document.createElement() method:

```
var x = document.createElement("STYLE");
```

You can also set the style properties of an existing element:

```
document.getElementById("myH1").style.color = "red";
```

The DOM

Document Object Model

HTML DOM allows JavaScript to react to HTML events.

Reacting to Events

A JavaScript can be executed when an event occurs, like when a user clicks on an HTML element.

To execute code when a user clicks on an element, add JavaScript code to an HTML event attribute:

onclick=*JavaScript*

The DOM

Document Object Model

Examples of HTML events:

- When a user clicks the mouse
- When a web page has loaded
- When an image has been loaded
- When the mouse moves over an element
- When an input field is changed
- When an HTML form is submitted
- When a user strokes a key

The DOM

Document Object Model

In this example, the content of the <h1> element is changed when a user clicks on it:

```
<!DOCTYPE html>
<html>
<body>

<h1 onclick="this.innerHTML='Ooops!'">Click on this text!</h1>

</body>
</html>
```

Click on this text!

Ooops!

The DOM

Document Object Model

HTML Event Attributes

To assign events to HTML elements you can use event attributes.

Example

Assign an onclick event to a button element:

```
<button onclick="displayDate()">Try it</button>
```

Click the button to display the date.

The time is?

Click the button to display the date.

The time is?

Thu Jun 18 2015 11:12:40 GMT-0500 (Central Daylight Time)

The DOM

Document Object Model

Assign Events Using the HTML DOM

The HTML DOM allows you to assign events to HTML elements using JavaScript:

Example

Assign an onclick event to a button element:

```
<script>
document.getElementById("myBtn").onclick = displayDate;
</script>
```

The DOM

Document Object Model

Click "Try it" to execute the displayDate() function.

Try it

```
<!DOCTYPE html> <html> <head></head>
<body>
<p>Click "Try it" to execute the displayDate() function.</p>
<button id="myBtn">Try it</button>
<p id="demo"></p>
<script>
document.getElementById("myBtn").onclick = displayDate;

function displayDate() {
   document.getElementById("demo").innerHTML = Date();
}
</script>
</body></html>
```

Click "Try it" to execute the displayDate() function.

Try it

Thu Jun 18 2015 11:18:19 GMT-0500 (Central Daylight Time)

The DOM

Document Object Model

The addEventListener() method

Example

Add an event listener that fires when a user clicks a button:

```
document.getElementById("myBtn").addEventListener("click",
displayDate);
```

The DOM

Document Object Model

The addEventListener() method attaches an event handler to the specified element.

The addEventListener() method attaches an event handler to an element without overwriting existing event handlers.

You can add many event handlers to one element.

You can add many event handlers of the same type to one element, i.e two "click" events.

You can add event listeners to any DOM object not only HTML elements. i.e the window object.

The DOM

Document Object Model

Add an Event Handler to an Element

Example

Alert "Hello World!" when the user clicks on an element:

```
element.addEventListener("click",
        function(){ alert("Hello World!"); });
```

You can also refer to an external "named" function:

Alert "Hello World!" when the user clicks on an element:

```
element.addEventListener("click", myFunction);

function myFunction() {    alert ("Hello World!");  }
```

The DOM

Document Object Model

Add Many Event Handlers to the Same Element

The addEventListener() method allows you to add many events to the same element, without overwriting existing events:

```
element.addEventListener("click", myFunction);
element.addEventListener("click", mySecondFunction);
```

You can add events of different types to the same element:

```
element.addEventListener("mouseover", myFunction);
element.addEventListener("click", mySecondFunction);
element.addEventListener("mouseout", myThirdFunction);
```

The DOM

Document Object Model

Add an Event Handler to the Window Object

The addEventListener() method allows you to add event listeners on any HTML DOM object such as HTML elements, the HTML document, the window object, or other objects that supports events, like the xmlHttpRequest object.

Example

Add an event listener that fires when a user resizes the window:

```
window.addEventListener("resize", function(){
   document.getElementById("demo").innerHTML =
sometext;
});
```

The DOM

Document Object Model

Passing Parameters

When passing parameter values, use an "anonymous function" that calls the specified function with the parameters:

```
element.addEventListener("click",
        function(){ myFunction(p1, p2); });
```

The DOM

Document Object Model

Event Bubbling or Event Capturing?

There are two ways of event propagation in the HTML DOM, bubbling and capturing.

Event propagation is a way of defining the element order when an event occurs. If you have a <p> element inside a element, and the user clicks on the <p> element, which element's "click" event should be handled first?

The DOM

Document Object Model

In *bubbling* the inner most element's event is handled first and then the outer: the <p> element's click event is handled first, then the element's click event.

In *capturing* the outer most element's event is handled first and then the inner: the element's click event will be handled first, then the <p> element's click event.

With the addEventListener() method you can specify the propagation type by using the "useCapture" parameter:

addEventListener(*event*, *function*, *useCapture*);

The DOM

Document Object Model

With the addEventListener() method you can specify the propagation type by using the "useCapture" parameter:

addEventListener(*event*, *function*, *useCapture*);

The default value is false, which will use the bubbling propagation

When the value is set to true, the event uses the capturing propagation.

```
document.getElementById("myP").addEventListener("click",
myFunction, true);
document.getElementById("myDiv").addEventListener("click",
myFunction, true);
```

The DOM

Document Object Model

Using Events

The HTML DOM allows you to execute code when an event occurs.

Events are generated by the browser when "things happen" to HTML elements:

An element is clicked on

The page has loaded

Input fields are changed

This example changes the style of the HTML element with id="id1", when the user clicks a button:

The DOM

Document Object Model

Using Events

This example changes the style of the HTML element with id="id1", when the user clicks a button:

```
<!DOCTYPE html>
<html>
<body>
<h1 id="id1">My Heading 1</h1>


<button type="button"
        onclick="document.getElementById('id1').style.color = 'red'">
            Click Me!</button>

</body>
</html>
```

The DOM

Document Object Model

This is a text. This is a text. This is a text. This is a text.

Hide text Show text

Using Events – Visibility

Hide text Show text

```
<!DOCTYPE html>
<html> <body>
<p id="p1">
This is a text. This is a text. This is a text. This is a text.
</p>
<input type="button" value="Hide text"
        onclick="document.getElementById('p1').style.visibility='hidden'">
<input type="button" value="Show text"
        onclick="document.getElementById('p1').style.visibility='visible'">
</body></html>
```

The DOM

Document Object Model

Using Events

This example changes the style of the HTML element with id="id1", when the user clicks a button:

```
<!DOCTYPE html>
<html>
<body>
<h1 id="id1">My Heading 1</h1>


<button type="button"
        onclick="document.getElementById('id1').style.color = 'red'">
            Click Me!</button>

</body>
</html>
```

Unobtrusive JavaScript

- allows separation of web site into 3 major categories:
  - content (HTML) - what is it?
  - presentation (CSS) - how does it look?
  - behavior (JavaScript) - how does it respond to user interaction?

Obtrusive event handlers (bad)

```
<button id="ok" onclick="okayClick();">OK</button>
```
HTML

```
// called when OK button is clicked
function okayClick() {
      alert("booyah");
}
```
JS

- this is bad style (HTML is cluttered with JS code)
- goal: remove all JavaScript code from the HTML body

Attaching an event handler in JavaScript code

```
// where element is a DOM element object
element.event = function;
```
JS

```
$("ok").onclick = okayClick;
```
JS

- it is legal to attach event handlers to elements' DOM objects in your JavaScript code
  - notice that you do not put parentheses after the function's name
- this is better style than attaching them in the HTML
- Where should we put the above code?

When does my code run?

```
<head>
<script src="myfile.js" type="text/javascript"></script>
</head>
<body> ... </body>
```
HTML

```
// global code
var x = 3;
function f(n) { return n + 1; }
function g(n) { return n - 1; }
x = f(x);
```
JS

- your file's JS code runs the moment the browser loads the script tag
  - any variables are declared immediately
  - any functions are declared but not called, unless your global code explicitly calls them

When does my code run?

```
<head>
<script src="myfile.js" type="text/javascript"></script>
</head>
<body> ... </body>
```
HTML

```
// global code
var x = 3;
function f(n) { return n + 1; }
function g(n) { return n - 1; }
x = f(x);
```
JS

- at this point in time, the browser has not yet read your page's body
  - none of the DOM objects for tags on the page have been created

A failed attempt at being unobtrusive

```
<head>
<script src="myfile.js" type="text/javascript"></script>
</head>
<body>
<div><button id="ok">OK</button></div>
```
HTML

```
// global code
$("ok").onclick = okayClick; // error: $("ok") is null
```
JS

- problem: global JS code runs the moment the script is loaded
- script in head is processed before page's body has loaded
  - no elements are available yet or can be accessed yet via the DOM

The `window.onload` event

```
// this will run once the page has finished loading
function functionName() {
      element.event = functionName;
      element.event = functionName;
...
}
window.onload = functionName; // global code
```

JS

- we want to attach our event handlers right after the page is done loading
  - there is a global event called `window.onload` event that occurs at that moment
- in `window.onload` handler we attach all the other handlers to run when events occur

An unobtrusive event handler

```
<!-- look Ma, no JavaScript! -->
<button id="ok">OK</button>
```

HTML

```
// called when page loads; sets up event handlers
function pageLoad() {
      $("ok").onclick = okayClick;
}
function okayClick() {
      alert("booyah");
}
window.onload = pageLoad; // global code
```

JS

Common unobtrusive JS errors

```
~~window.onload = pageLoad();~~
window.onload = pageLoad;
~~okButton.onclick = okayClick();~~
okButton.onclick = okayClick;
```
JS

- event names are all lowercase, not capitalized like most variables

```
~~window.onLoad = pageLoad;~~
window.onload = pageLoad;
```
JS

Anonymous functions

```
function(parameters) {
        statements;
}
```
JS

- JavaScript allows you to declare anonymous functions
- quickly creates a function without giving it a name
- can be stored as a variable, attached as an event handler, etc.

Anonymous function example

```
window.onload = function() {
      var okButton = document.getElementById("ok");
      okButton.onclick = okayClick;
};
function okayClick() {
      alert("booyah");
}
```

JS

The keyword `this`

```
this.fieldName // access field
this.fieldName = value; // modify field
this.methodName(parameters); // call method
```

JS

- all JavaScript code actually runs inside of an object
- by default, code runs inside the global window object
  - all global variables and functions you declare become part of window
- the this keyword refers to the current object

The keyword `this`

```
function pageLoad() {
      $("ok").onclick = okayClick; // bound to okButton
here
}
function okayClick() { // okayClick knows what DOM object
      this.innerHTML = "booyah"; // it was called on
}
window.onload = pageLoad;
```

JS

- event handlers attached unobtrusively are **bound** to the element
- inside the handler, that element becomes this (rather than the window)

Fixing redundant code with `this`

```
<fieldset>
	<label><input type="radio" name="ducks"
value="Huey" /> Huey</label>
	<label><input type="radio" name="ducks"
value="Dewey" /> Dewey</label>
	<label><input type="radio" name="ducks"
value="Louie" /> Louie</label>
</fieldset>
```

HTML

```
function processDucks() {
	if ($("huey").checked) {
		alert("Huey is checked!");
	} else if ($("dewey").checked) {
		alert("Dewey is checked!");
	} else {
		alert("Louie is checked!");
	}
	alert(this.value + " is checked!");
}
```

JS

Example: Tip Calculator

```
<h1>Tip Calculator</h1>
<div>
        $<input id="subtotal" type="text" size= "5" /> subtotal
<br />
        <button id="tenpercent">10%</button>
        <button id="fifteenpercent"> 15%</button>
        <button id="eighteenpercent"> 18%</button>

        <span id="total"></span>
</div>
```

HTML

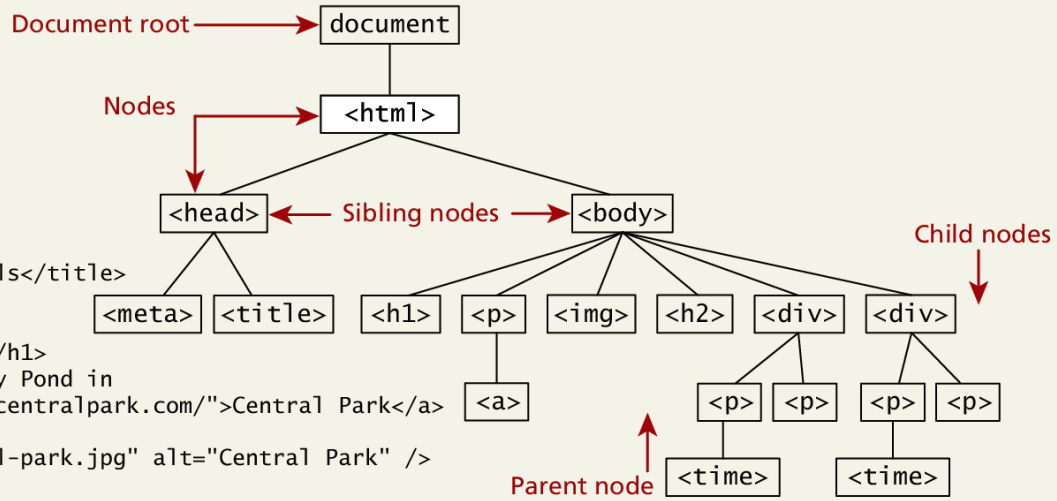
```
window.onload = function() {
        $("tenpercent").onclick = computeTip;
        }
function computeTip{
        var subtotal = parseFloat($("subtotal").value);
        var tipAmount = subtotal*0.1;//Add this code
        $("total").innerHTML = "Tip: $" + tipAmount;
}
```

JS

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>
```

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);">
...
```

inline.js

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">
...
```

listener.js

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.