

Chapter1

Hypertext Transport Protocol (HTTP)

- HTTP is based on the request-response communication model:
 - Client sends a request
 - Server sends a response
- HTTP is a stateless protocol:
 - The protocol does not require the server to remember anything about the client between requests

Http:

- The information transmitted using HTTP is often entirely text
- Can use the Internet's Telnet protocol to simulate browser request and view server response

```

Connect    { $ telnet www.example.org 80
            Trying 192.0.34.166...
            Connected to www.example.com
            (192.0.34.166).
            Escape character is '^]'.
Send       {
Request    { GET / HTTP/1.1
            Host: www.example.org
Receive    {
Response   { HTTP/1.1 200 OK
            Date: Thu, 09 Oct 2003 20:30:49 GMT
            ...
  
```

Http request

- Structure of the request:
 - **start line**
 - header field(s)
 - blank line
 - optional body
- **Start line**
 - Example: GET / HTTP/1.1
- Three space-separated parts:
 - HTTP request method
 - Request-URI
 - **HTTP version**
 - We will cover 1.1, in which version part of start line must be exactly as shown
- **Uniform Resource Identifier (URI)**
 - Syntax: *scheme* : *scheme-depend-part*
 - Ex: In <http://www.example.com/> the scheme is http
 - Request-URI is the portion of the requested URI that follows the host name (which is supplied by the required Host header field)
 - Ex: / is Request-URI portion of <http://www.example.com/>

URI's are of two types:

- Uniform Resource Name ([URN](#))
 - Can be used to identify resources with unique names, such as books (which have unique ISBN's)
 - Scheme is urn
- Uniform Resource Locator ([URL](#))
 - Specifies location at which a resource can be found
 - In addition to http, some other URL schemes are https, ftp, mailto, and file

Common request methods:

- GET
 - Used if link is clicked or address typed in browser
 - Nobody in request with GET method
- POST
 - Used when submit button is clicked on a form
 - Form information contained in body of request
- HEAD: Requests that only header fields (no body) be returned in the response

Header field structure:

- *field name : field value*
- Syntax
 - Field name is not case sensitive
 - Field value may continue on multiple lines by starting continuation lines with white space
 - Field values may contain MIME types, quality values, and wildcard characters (*'s)

Multipurpose Internet Mail Extensions (MIME)

- Convention for specifying content type of a message
 - In HTTP, typically used to specify content type of the body of the response
- MIME content type syntax:
 - *top-level type / subtype*
- Examples: text/html, image/jpeg

Http response

- Structure of the response:
 - status line
 - header field(s)
 - blank line
 - optional body
- **Status line**
 - Example: HTTP/1.1 200 OK
- Three space-separated parts:
 - HTTP version
 - status code
 - reason phrase (intended for human use)

- Status code
 - Three-digit number
 - First digit is class of the status code:
 - 1=Informational
 - 2=Success
 - 3=Redirection (alternate URL is supplied)
 - 4=Client Error
 - 5=Server Error
 - Other two digits provide additional information

Client Caching

- A cache is a local copy of information obtained from some other source
- Most web browsers use cache to store requested resources so that subsequent requests to the same resource will not necessarily require an HTTP request/response
 - Ex: icon appearing multiple times in a Web page

Cache advantages

- (Much) faster than HTTP request/response
- Less network traffic
- Less load on server

Cache disadvantage

- Cached copy of resource may be invalid (inconsistent with remote version)

Validating cached resource:

- Send HTTP HEAD request and check Last-Modified or ETag header in response
- Compare current date/time with Expires header sent in response containing resource
- If no Expires header was sent, use heuristic algorithm to estimate value for Expires

Web Servers

- Basic functionality:
 - Receive HTTP request via TCP
 - Map Host header to specific virtual host (one of many host names sharing an IP address)
 - Map Request-URI to specific resource associated with the virtual host
 - File: Return file in HTTP response
 - Program: Run program and return output in HTTP response
 - Map type of resource to appropriate MIME type and use to set Content-Type header in HTTP response
 - Log information about the request and response

Types:-

- httpd: UIUC, primary Web server c. 1995
- Apache: “A patchy” version of httpd, now the most popular server (esp. on Linux platforms)
- IIS: Microsoft Internet Information Server

- Tomcat:
 - Java-based
 - Provides container (Catalina) for running Java servlets (HTML-generating programs) as back-end to Apache or IIS
 - Can run stand-alone using Coyote HTTP front-end

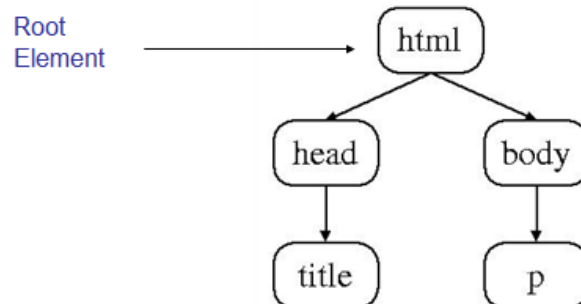
Some Coyote communication parameters:

- Allowed/blocked IP addresses
- Max. simultaneous active TCP connections
- Max. queued TCP connection requests
- “Keep-alive” time for inactive TCP connections
- Modify parameters to tune server performance

Some Catalina container parameters:

- Virtual host names and associated ports
- Logging preferences
- Mapping from Request-URI’s to server resources
- Password protection of resources
- Use of server-side caching

Chapter2



Common HTML Elements

- Headings are produced using h1, h2, ..., h6 elements:
- Should use h1 for highest level, h2 for next highest, etc.

```

<h1>
  Some Common HTML Elements
</h1>
<h2>
  Simple formatting elements
</h2>
  
```

Text can be formatted in various ways:

- Apply style sheet technology (next chapter) to a span element (a styleless wrapper):

```

<span style="font-style:italic">separating line</span>
  
```

- Use a phrase element that specifies semantics of text (not style directly):

```
<strong>hr</strong>
```

- Use a font style element
 - Not recommended, but frequently used

TABLE 2.3: HTML font style elements.

Element	Font used by content
b	Bold-face
<i>i</i>	Italic
<code>tt</code>	“Teletype” (fixed-width font)
big	Increased font size
<small>small</small>	Decreased font size

Images can be embedded using img element

```

```

- Attributes:
 - src: URL of image file (required). Browser generates a GET request to this URL.
 - alt: text description of image (required)
 - height / width: dimensions of area that image will occupy (recommended)

- Hyperlinks are produced by the anchor element a

See

```
<a href="http://www.w3.org/TR/html4/index/elements.html">the
  W3C HTML 4.01 Element Index</a>
for a complete list of elements.
```

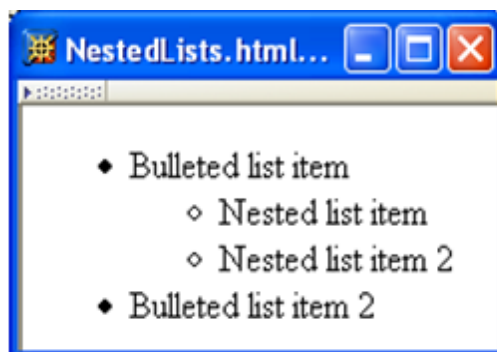
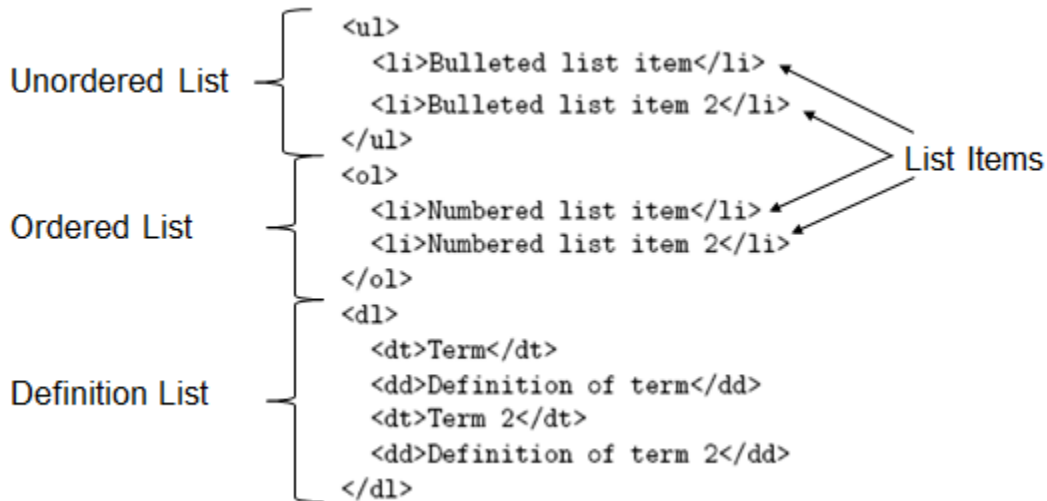
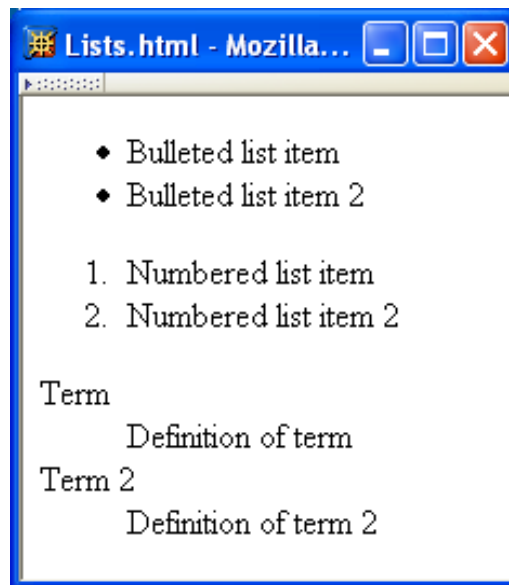
- Anchors can be used as source (previous example) or destination

```
<a id="section1" name="section1"></a>
```

- The fragment portion of a URL is used to reference a destination anchor

```
<a href="http://www.example.org/PageWithAnchor.html#section1">...
```

- Browser scrolls so destination anchor is at (or near) top of client area
- Most HTML elements are either block or inline
 - Block: browser automatically generates line breaks before and after the element content
 - Ex: p
 - Inline: element content is added to the “flow”
 - Ex: span, tt, strong, a

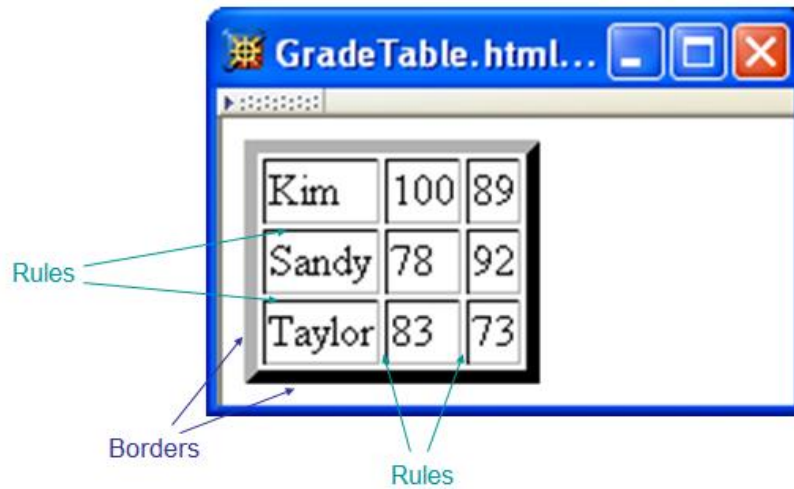
List

```

<ul>
  <li>Bulleted list item
    <ul>
      <li>Nested list item</li>
      <li>Nested list item 2</li>
    </ul>
  </li>
  <li>Bulleted list item 2</li>
</ul>

```

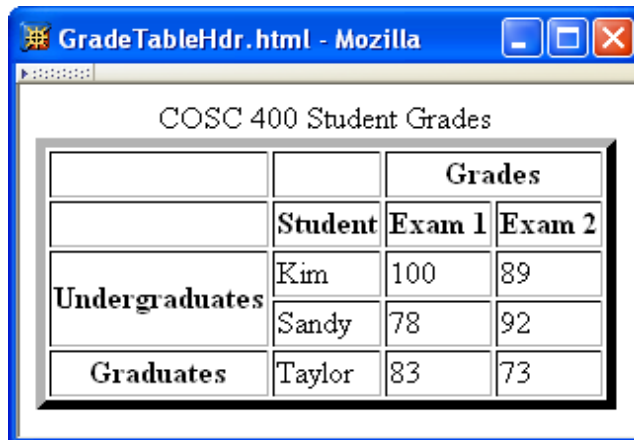
Tables



```

<table border="5">
  <tr>
    <td>Kim</td><td>100</td><td>89</td>
  </tr>
  <tr>
    <td>Sandy</td><td>78</td><td>92</td>
  </tr>
  <tr>
    <td>Taylor</td><td>83</td><td>73</td>
  </tr>
</table>
    
```

Annotations: "Border 5 pixels, rules 1 pixel" points to the `border="5"` attribute. "Table Row" points to the three rows of the table. "Table Data" points to the content within the table cells.



```

<table border="5">
  <caption>
    COSC 400 Student Grades
  </caption>
  <tr>
    <td>&nbsp;</td><td>&nbsp;</td><th colspan="2">Grades</th>
  </tr>
  <tr>
    <td>&nbsp;</td><th>Student</th><th>Exam 1</th><th>Exam 2</th>
  </tr>
  <tr>
    <th rowspan="2">Undergraduates</th><td>Kim</td><td>100</td><td>89</td>
  </tr>
  <tr>
    <td>Sandy</td><td>78</td><td>92</td>
  </tr>
  <tr>
    <th>Graduates</th><td>Taylor</td><td>83</td><td>73</td>
  </tr>
</table>

```

Table Header

cellspacing cellpadding

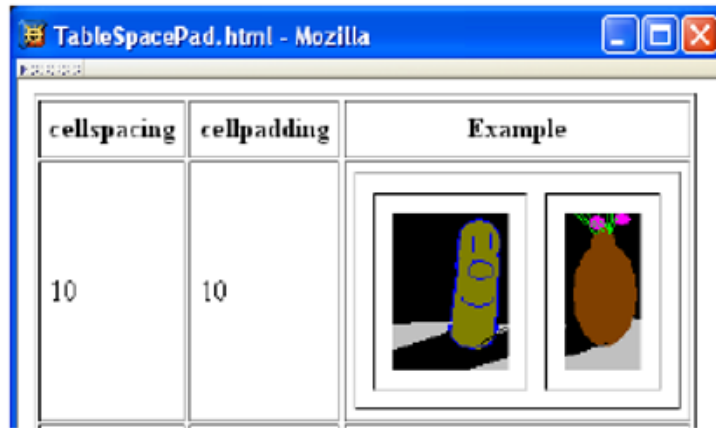
0	10	
---	----	--

cellspacing cellpadding

10	0	
----	---	--

cellspacing cellpadding

0	0	
---	---	--



```
<table border="1" cellspacing="10" cellpadding="10">
```

Forms

action specifies URL where form data is sent in an HTTP request

Each form is content of a form element

```
<form action="http://www.example.org" method="get">
  <div>
    <label>
      Enter your name: <input type="text" name="username" size="40" />
    </label>
    <br />
    <label>
      Give your life's story in 100 words or less:
    <br />
    <textarea name="lifestory" rows="5" cols="60"></textarea>
    </label>
    <br />
  </div>
</form>
```

HTTP request method (lower case)

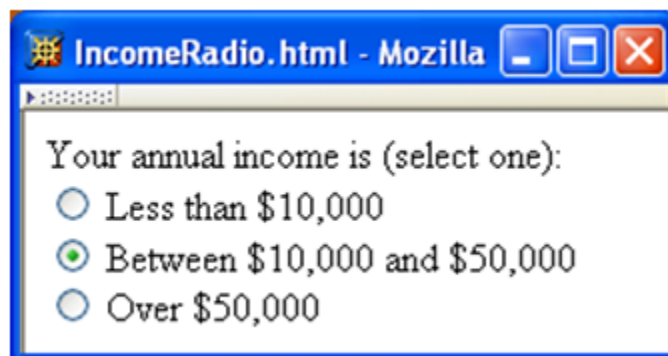
Check all that apply to you: Value sent in HTTP request if box is checked

```

<label>      Checkbox control
<input type="checkbox" name="boxgroup1" value="tall" />tall
</label>
<label>
  <input type="checkbox" name="boxgroup1" value="funny" />funny
</label>
<label>
  <input type="checkbox" name="boxgroup1" value="smart" />smart
</label>
<br /><br />
<input type="submit" name="doit" value="Publish My Life's Story" />
</div>
</form>

```

Radio buttons: at most one can be selected at a time.

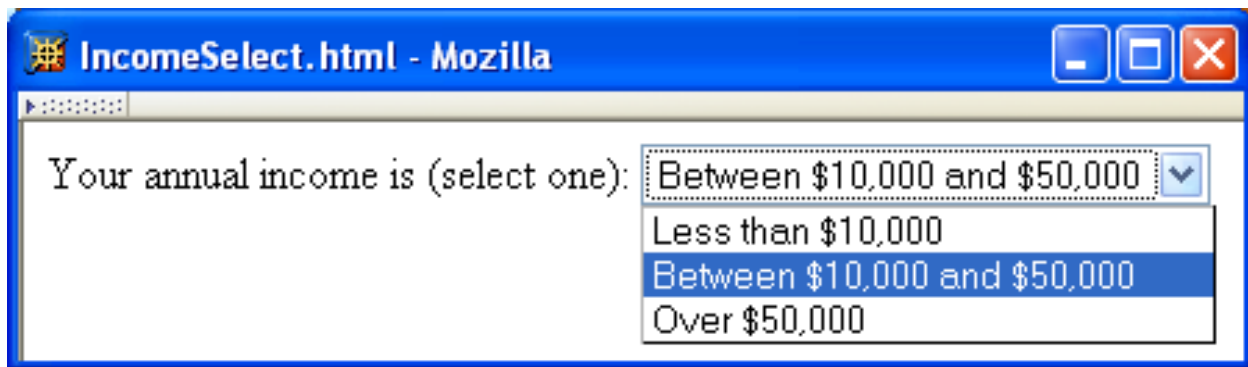


```

Your annual income is (select one):<br />
<label>
  <input type="radio" name="radgroup1" value="0-10" />
  Less than $10,000
</label><br />
<label>
  <input type="radio" name="radgroup1" value="10-50"
  checked="checked" />
  Between $10,000 and $50,000
</label><br />
<label>
  <input type="radio" name="radgroup1" value="&gt;50" />
  Over $50,000
</label>

```

All radio buttons with the same name form a *button set*



Your annual income is (select one):

```
<select name="income">Each menu item has its own value
  <option value="0-10">Less than $10,000</option>
  <option value="10-50" selected="selected">
    Between $10,000 and $50,000 Item initially displayed in menu
  </option> control
  <option value="&gt;50">Over $50,000</option>
</select>
```

Chapter3

Cascading Style Sheets (CSS)

- A style sheet technology designed to work with HTML and XML documents.
- CSS provides a great deal of control over the presentation of a document,

Style Sheet Languages

- Cascading Style Sheets (CSS)
 - Applies to (X)HTML as well as XML documents in general
- Extensible Stylesheet Language (XSL)
 - Often used to transform one XML document to another form, but can also add style
 - XSL Transformations covered in later chapter

How to use css

- Style sheets referenced by link HTML element are called external style sheets
- Style sheets can be **embedded** directly in HTML document using style element

```
<!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>
      CSSHelloWorld.html
    </title>
    <link rel="stylesheet" type="text/css" href="style1.css"
          title="Style 1" />
    <link rel="alternate stylesheet" type="text/css" href="style2.css"
          title="Style 2" />
  </head>
  <body>
    <p>
```

```
<head>
  <title>InternalStyleSheet.html</title>
  <style type="text/css">
    h1, h2 { background-color:aqua }
  </style>
</head>
```

CSS Syntax:

Selector Strings

- Single element type:

```
p { font-size:smaller; letter-spacing:1em }
```

- Multiple element types:

```
h1,h2,h3,h4,h5,h6 { background-color:purple }
```

- All element types:

Universal selector

```
* { font-weight:bold }
```

- Specific elements by id:

```
#p1, #p3 { background-color:aqua }
```

ID selector

- Elements belonging to a style class:

- Referencing a style class in HTML:

```
#p4, .takeNote { font-style:italic }
```

class selector

- Elements of a certain type and class:

```
span.special { font-size:x-large }
```

this rule applies only to span's belonging to class special

CSS Font Properties

- A font family is a collection of related fonts (typically differ in size, weight, etc.)

```
<p style="font-family:'Jenkins v2.0'">
```

- font-family property can accept a list of families, including generic font families

```
font-family:"Edwardian Script ITC", "French Script MT", cursive
```

first choice font

second choice font

generic

Property	Possible Values
font-style	normal (initial value), italic (more cursive than normal), or oblique (more slanted than normal)
font-weight	bold or normal (initial value) are standard values, although other values can be used with font families having multiple gradations of boldness (see CSS2 [W3C-CSS-2.0] for details)
font-variant	small-caps, which displays lowercase characters using uppercase glyphs (small uppercase glyphs if possible), or normal (initial value)

- **font shortcut property:**

```
{ font: italic bold 12pt "Helvetica", sans-serif }
```

↓

```
{ font-style: italic;
  font-variant: normal;
  font-weight: bold;
  font-size: 12pt;
  line-height: normal;
  font-family: "Helvetica", sans-serif }
```

Initial values used if no value specified in font property list

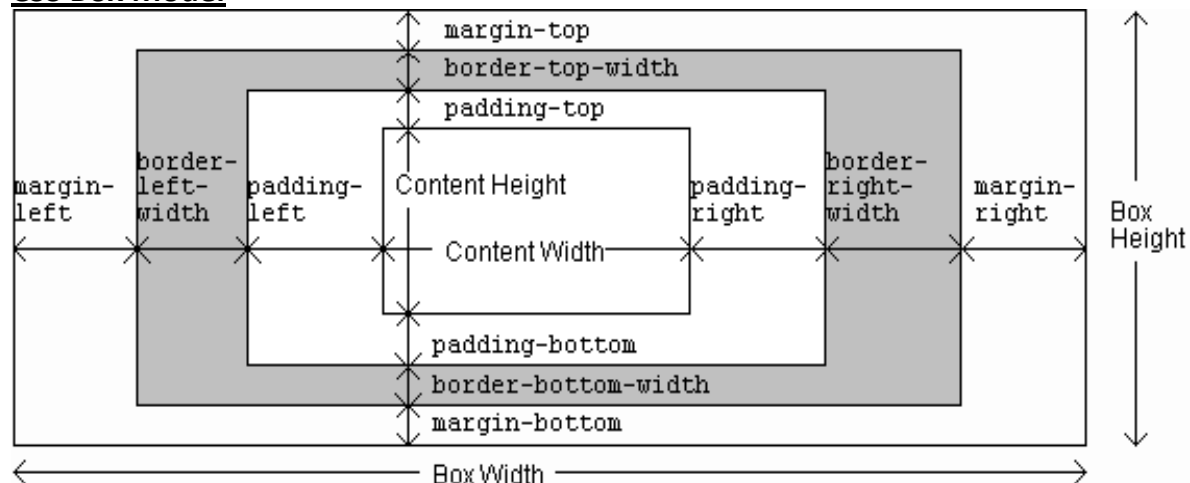
CSS Text Color

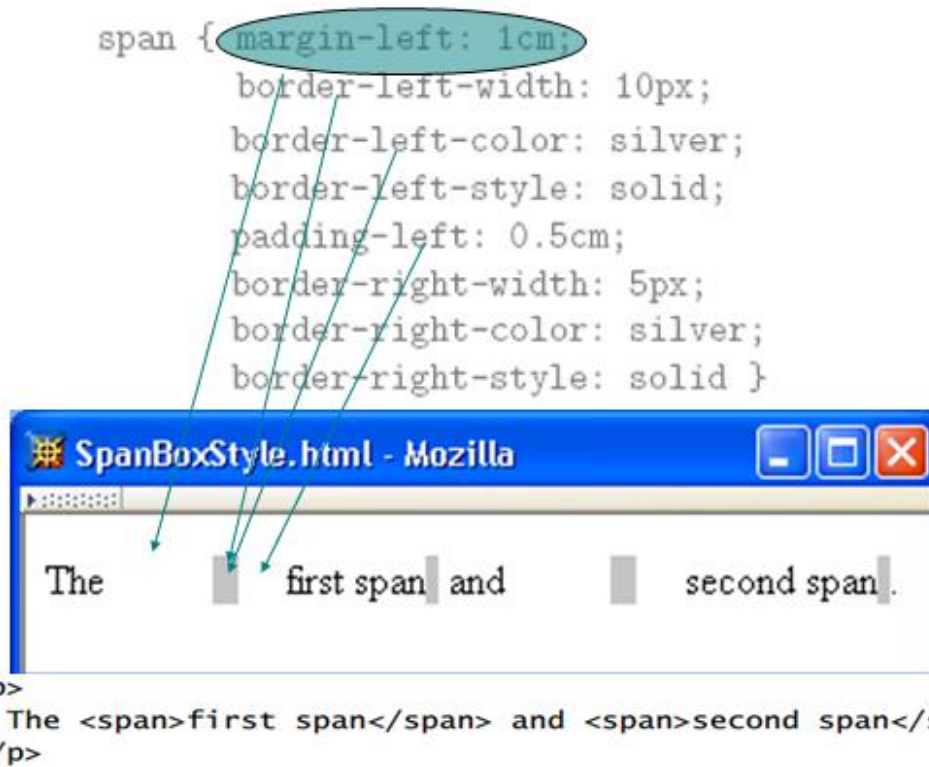
- Font color specified by color property
- Two primary ways of specifying colors:
 - Color name: black, gray, silver, white, red, lime, blue, yellow, aqua, fuchsia, maroon, green, navy, olive, teal, purple, full list at <http://www.w3.org/TR/SVG11/types.html#ColorKeywords>
 - red/green/blue (RGB) values

TABLE 3.7 Alternative Formats for Specifying Numeric Color Values

Format	Example	Meaning
Functional, integer arguments	rgb(255, 170, 0)	Use arguments as RGB values.
Functional, percentage arguments	rgb(100%, 66.7%, 0%)	Multiply arguments by 255 and round to obtain RGB values (at most one decimal place allowed in arguments).
Hexadecimal	#ffaa00	The first pair of hexadecimal digits represents the red intensity; the second and third represent green and blue, respectively.
Abbreviated hexadecimal	#fa0	Duplicate the first hexadecimal digit to obtain red intensity; duplicate the second and third to obtain green and blue, respectively.

CSS Box Model



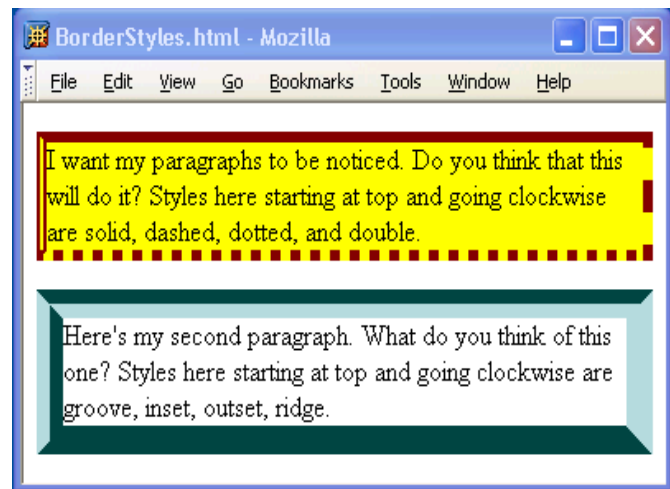
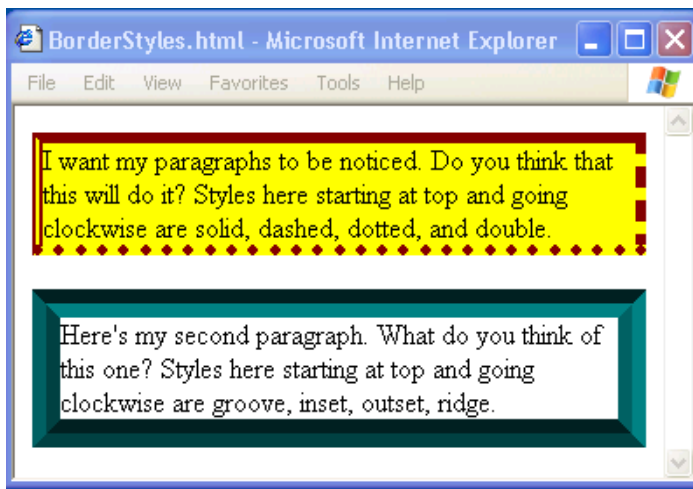
**TABLE 3.9** Basic CSS Style Properties Associated with the Box Model

Property	Values
padding-{top, right, bottom, left}	CSS length (Section 3.6.2).
padding	One to four length values (see text).

TABLE 3.10 Meaning of Values for Certain Shorthand Properties that Take One to Four Values

Number of Values	Meaning
One	Assign this value to all four associated properties (top, right, bottom, and left).
Two	Assign first value to associated top and bottom properties, second value to associated right and left properties.
Three	Assign first value to associated top property, second value to right and left, and third value to bottom.
Four	Assign first value to associated top property, second to right, third to bottom, and fourth to left.

border-{top, right, bottom, left}-width	thin, medium (initial value), thick, or a length.
border-width	One to four border-*-width values.
border-{top, right, bottom, left}-color	Color value. Initial value is value of element's color property.
border-color	transparent or one to four border-*-color values.
border-{top, right, bottom, left}-style	none (initial value), hidden, dotted, dashed, solid, double, groove, ridge, inset, outset.
border-style	One to four border-*-style values.



`border-{top,right,bottom,left}`

One to three values (in any order) for `border-*-width`, `border-*-color`, and `border-*-style`. Initial values are used for any unspecified values.

`border`

One to three values; equivalent to specifying given values for each of `border-top`, `border-right`, `border-bottom`, and `border-left`.

`margin-{top,right,bottom,left}`

`auto` (see text) or length.

`margin`

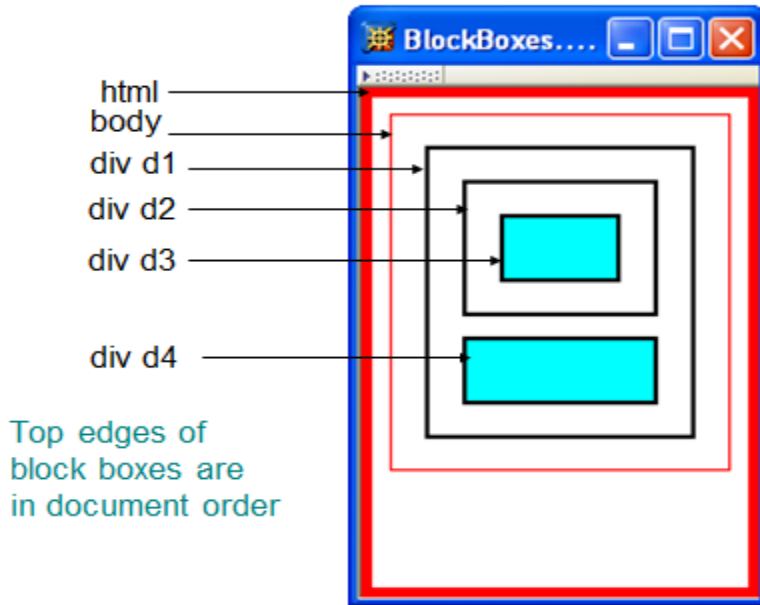
One to four `margin-*` values.

Normal Flow Layout

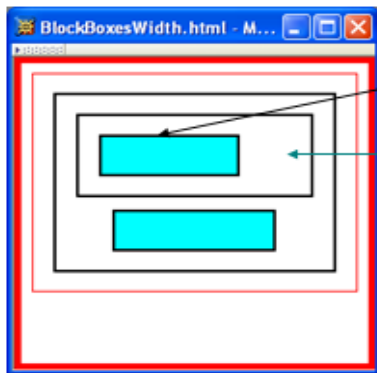
- In normal flow processing, each displayed element has a corresponding box
 - html element box is called initial containing block and corresponds to entire document
 - Boxes of child elements are contained in boxes of parent
 - Sibling block elements are laid out one on top of the other
 - Sibling inline elements are one after the other

```
html, body { border:solid red thin }
html { border-width:thick }
body { padding:15px }
div { margin:0px; padding:15px; border:solid black 2px }
.shade { background-color:aqua }
.topMargin { margin-top:10px }
```

```
<body>
  <div id="d1">
    <div id="d2">
      <div id="d3" class="shade"></div>
    </div>
    <div id="d4" class="shade topMargin"></div>
  </div>
</body>
```



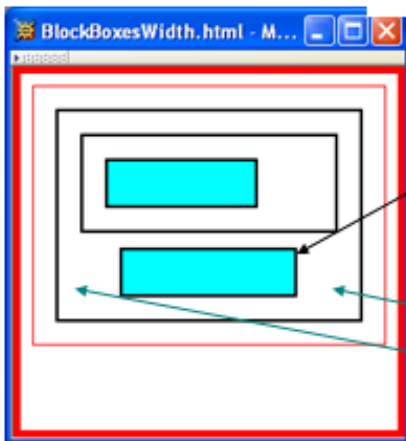
- Can also specify CSS length or percentage (of parent’s content width) for width property



```
#d3 { width:50% }
```

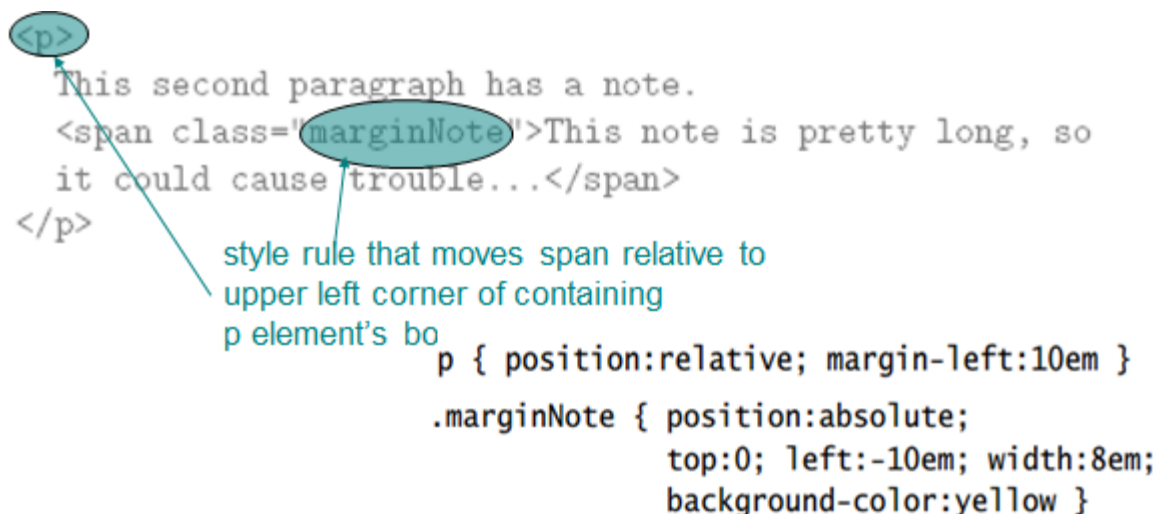
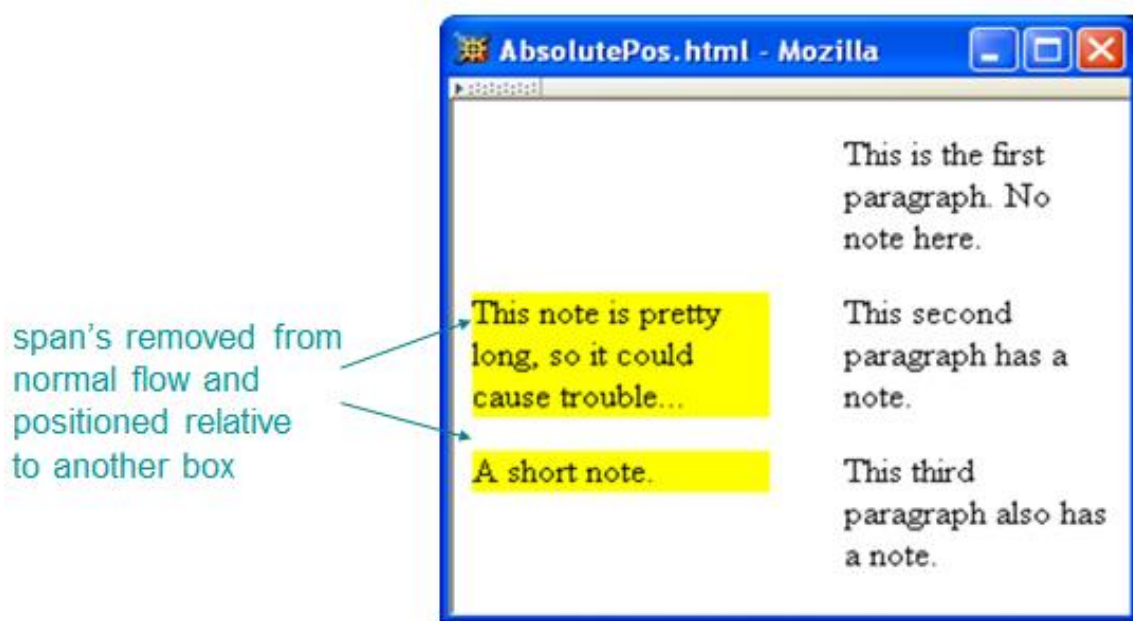
By default, width of right margin is adjusted to accommodate a change to width

```
#d4 { width:50%;| margin-left:auto; margin-right:auto }
```



Centering can be achieved by setting both margins to auto

- CSS allows for boxes to be positioned outside the normal flow:
 - **Absolute positioning**



Properties used to specify positioning:

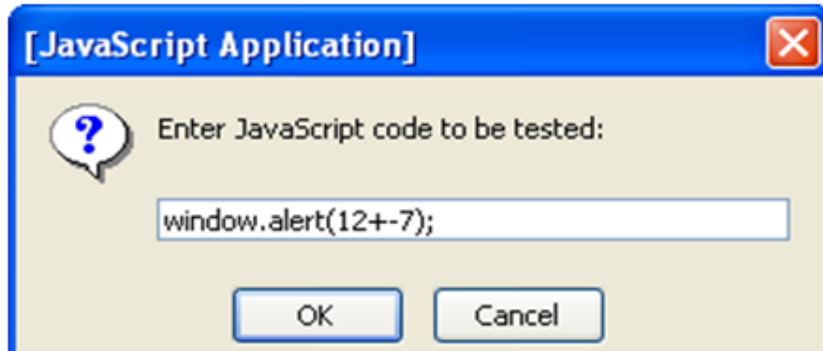
- position: static (initial value), relative, or absolute
 - Element is positioned if this property not static
 - Properties left, right, top, bottom apply to positioned elements
 - Primary values are auto (initial value) or CSS length
- float: none, left, or right
 - Applies to elements with static and relative positioning only

Chapter4

JavaScript is a scripting language: designed to be executed within a larger software environment

Prompt window example:

```
var inString = window.prompt("Enter JavaScript code to be tested:",
                             "");
```

**Variables and Data Types**

- Type of a variable is dynamic: depends on the type of data it contains
- JavaScript has six data types:
 - Number
 - String
 - Boolean (values true and false)
 - Object
 - Null (only value of this type is null)
 - Undefined (value of newly created variable)
- Primitive data types: all but Object
- typeof operator returns string related to data type
 - Syntax: `typeof expression`

Example:

```
// TypeOf.js
var i;
var j;
j = "Not a number";
alert("i is " + (typeof i) + "\n" +
      "j is " + (typeof j));
```

**Syntax rules for names (identifiers):**

- Must begin with letter or underscore (_)
- Must contain only letters, underscores, and digits (or certain other characters)
- Must not be a reserved word

JavaScript Operators

- Operators are used to create compound expressions from simpler expressions
- Operators can be classified according to the number of operands involved:
 - Unary: one operand (e.g., `typeof i`)
 - Prefix or postfix (e.g., `++i` or `i++`)
 - Binary: two operands (e.g., `x + y`)
 - Ternary: three operands (conditional operator)

Automatic Type Conversion

- Binary operators +, -, *, /, % convert both operands to Number
 - Exception: If one of operands of + is String then the other is converted to String
- Relational operators <, >, <=, >= convert both operands to Number
 - Exception: If both operands are String, no conversion is performed and lexicographic string comparison is performed
- Operators ==, != convert both operands to Number
 - Exception: If both operands are String, no conversion is performed (lex. comparison)
 - Exception: values of Undefined and Null are equal(!)
 - Exception: instance of Date built-in “class” is converted to String (and host object conversion is implementation dependent)
 - Exception: two Objects are equal only if they are references to the same object
- Operators ===, !== are strict:
 - Two operands are === only if they are of the same type and have the same value
 - “Same value” for objects means that the operands are references to the same object
- Unary +, - convert their operand to Number
- Logical &&, ||, ! convert their operands to Boolean (normally)

JavaScript Functions

```
function oneTo(high) {
  return Math.ceil(Math.random()*high);
}
thinkingOf = oneTo(100);
```

Argument value(s) associated with corresponding formal parameters

Expression(s) in body evaluated as if formal parameters are variables initialized by argument values

- **Function call semantics details:**
 - Arguments:
 - May be expressions:
 - Object’s effectively passed by reference (more later)
 - Formal parameters:
 - May be assigned values, argument is not affected
 - Return value:
 - If last statement executed is not return-value, then returned value is of type Undefined
- Number mismatch between argument list and formal parameter list:
 - More arguments: excess ignored
 - Fewer arguments: remaining parameters are Undefined

Local vs. global variables

Global variable: declared outside any function

Local variable declared within a function

```

var j=6; // global variable declaration and initialization
function test()
{
  var j; // local variable declaration
  j=7; // Which variable(s) does this change?
  return;
}
test();
window.alert(j);

```

```

var j=6; // global variable declaration and initialization
function test()
{
  var j; // local variable declaration
  j=7; // Which variable(s) does this change?
  return;
}
test();
window.alert(j);

```

Output is 7

In browsers, global variables (and functions) are stored as properties of the window built-in object.

```
window.j = 7;
```

- Explicit type conversion supplied by built-in functions
 - Boolean(), String(), Number()
 - Each takes a single argument, returns value representing argument converted according to type-conversion rules given earlier

Object Introduction

- An object is a set of properties
- A property consists of a unique (within an object) name with an associated value
- The type of a property depends on the type of its value and can vary dynamically

```

o.prop = true;      prop is Boolean
o.prop = "true";   prop is now String
o.prop = 1;        prop is now Number

```

Object Creation

- Objects are created using new expression

```
new Object() Constructor and argument list
```

- A constructor is a function
 - When called via new expression, a new empty Object is created and passed to the constructor along with the argument values
 - Constructor performs initialization on object
 - Can add properties and methods to object
 - Can add object to an inheritance hierarchy

Property Creation

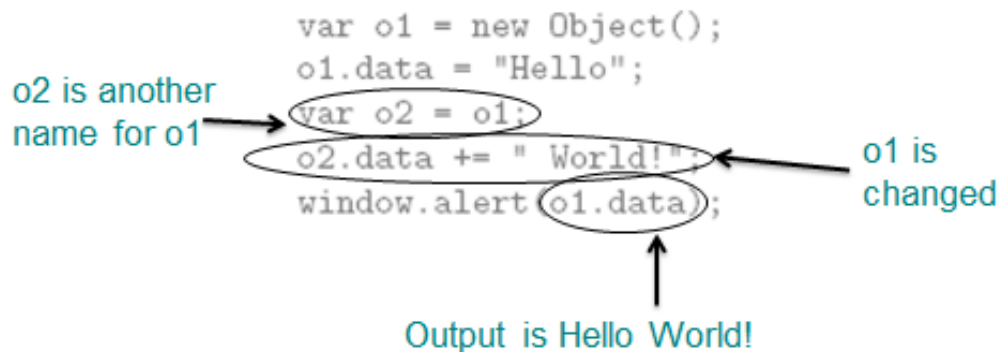
- Assignment to a non-existent (even if inherited) property name creates the property:

```
o1.testing = "This is a test";
```

- Object initializer notation can be used to create an object (using Object() constructor) and one or more properties in a single statement:

```
var o2 = { p1:5+9, p2:null, testing:"This is a test" };
```

Object argument values are references



```

function objArgs(param1, param2) {
  // Change the data in param1 and its argument
  param1.data = "changed";
  // Change the object referenced by param2, but not its argument
  param2 = param1;

  window.alert("param1 is " + param1.data + "\n" +
    "param2 is " + param2.data);
  return;
}

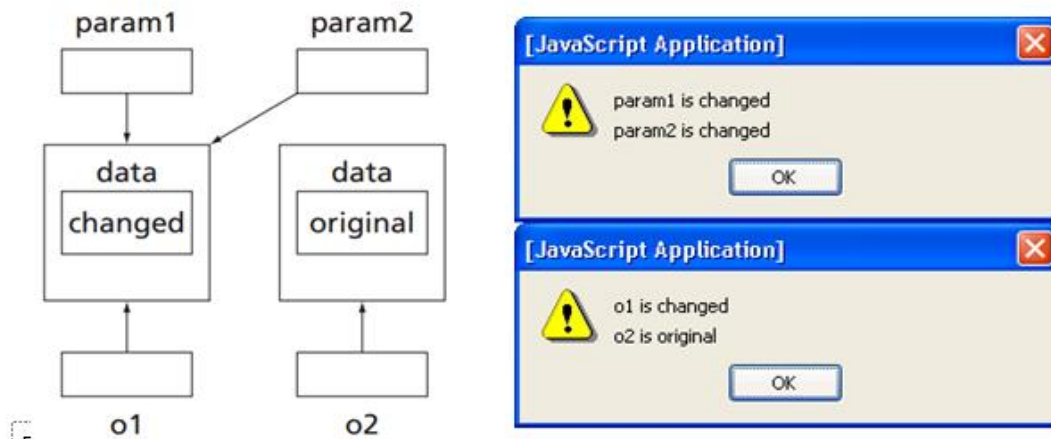
```

```

// Create two different objects with identical data
var o1 = new Object();
o1.data = "original";
var o2 = new Object();
o2.data = "original";

```

```
// Call the function on these objects and display the results
objArgs(o1, o2);
window.alert("o1 is " + o1.data + "\n" +
            "o2 is " + o2.data);
```



JavaScript Regular Expressions

A regular expression is a particular representation of a set of strings

Regular expression as String (must escape \)

```
var acTest = new RegExp("\\d\\d\\d$");
if (!acTest.test(areaCode)) {
    window.alert(areaCode + " is not a valid area code.");
}
```

Variable containing string to be tested

Method inherited by RegExp instances:
returns true if the argument *contains* a substring in the set of strings represented by the regular expression

Represents beginning of string

Represents end of string

```
var acTest = new RegExp("^\\d\\d\\d$");
if (!acTest.test(areaCode)) {
    window.alert(areaCode + " is not a valid area code.");
}
```

This expression matches only strings with exactly three digits (no other characters, even white space)


```
var acTest = new RegExp("^\\d\\d\\d");
```

- if we removed the dollar sign → the set of all strings that begin with three digits
- Alternate syntax:

```
var acTest = /^\\d\\d\\d/;
```

- Simplest regular expression is any character that is not a special character:

```
^ $ \ . * + ? ( ) [ ] { } |
```

- Ex: `_` is a regular expression representing {“_”}
- Backslash-escape special character is also a regular expression
 - Ex: `\\$` represents {“\$”}
- Special character. (dot) represents any character except a line terminator
- Several escape codes are regular expressions representing sets of chars:

TABLE 4.10 JavaScript Multicharacter Escape Codes

Escape Code	Characters Represented
<code>\\d</code>	Digit: 0 through 9
<code>\\D</code>	Any character except those matched by <code>\\d</code>
<code>\\s</code>	Space: any JavaScript white space or line terminator (space, tab, line feed, etc.)
<code>\\S</code>	Any character except those matched by <code>\\s</code>
<code>\\w</code>	“Word” character: any letter (a through z and A through Z), digit (0 through 9), or underscore (-)
<code>\\W</code>	Any character except those matched by <code>\\w</code>

Chapter5

The Document Object Model (DOM) is an API that allows programs to interact with HTML (or XML) documents

- In typical browsers, the JavaScript version of the API is provided via the document host object

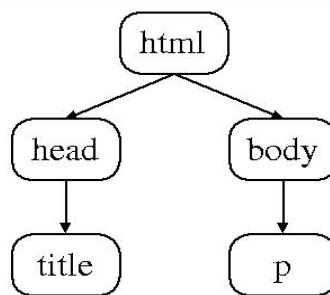
Intrinsic Event Handling

Attribute	When Called
<code>onload</code>	The body of the document has just been fully read and parsed by the browser (this attribute pertains only to <code>body</code> and <code>frameset</code>).
<code>onunload</code>	The browser is ready to load a new document in place of the current document (this attribute only pertains to <code>body</code> and <code>frameset</code>).
<code>onclick</code>	A mouse button has been clicked and released over the element.
<code>ondblclick</code>	The mouse has been double-clicked over the element.
<code>onmousedown</code>	The mouse has been clicked over the element.
<code>onmouseup</code>	The mouse has been released over the element.
<code>onmouseover</code>	The mouse has just moved over the element.
<code>onmousemove</code>	The mouse has moved from one location to another over the element.
<code>onmouseout</code>	The mouse has just moved away from the element.

<code>onblur</code>	The element has just lost the keyboard focus (attribute pertains only to the same elements as <code>onfocus</code>).
<code>onkeypress</code>	This element has the focus, and a key has been pressed and released.
<code>onkeydown</code>	This element has the focus, and a key has been pressed.
<code>onkeyup</code>	This element has the focus, and a key has been released.
<code>onsubmit</code>	This element is ready to be submitted (applies only to form elements).
<code>onreset</code>	This element is ready to be reset (applies only to form elements).
<code>onselect</code>	Text in this element has been selected (highlighted) in preparation for editing (applies only to <code>input</code> and <code>textarea</code> elements).

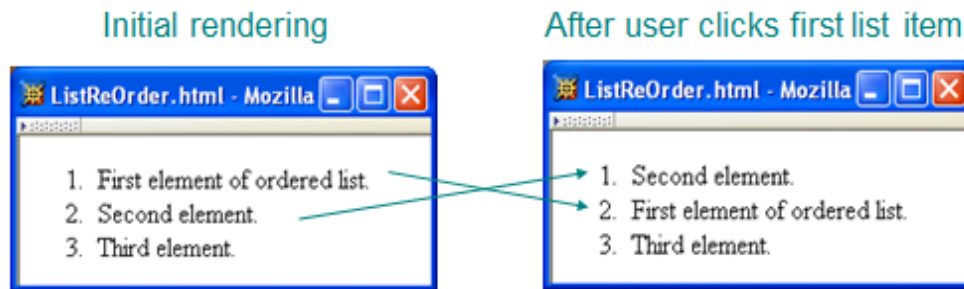
Document Tree

- Recall that HTML document elements form a tree structure, *e.g.*,
- DOM allows scripts to access and modify the document tree



Property	Description
<code>nodeType</code>	Number representing the type of node (Element, Comment, etc.). See Table 5.3.
<code>nodeName</code>	String providing a name for this Node (form of name depends on the <code>nodeType</code> ; see text).
<code>parentNode</code>	Reference to object that is this node's parent.
<code>childNodes</code>	Acts like a read-only array containing this node's child nodes. Has length 0 if this node has no children.
<code>previousSibling</code>	Previous sibling of this node, or null if no previous sibling exists.
<code>nextSibling</code>	Next sibling of this node, or null if no next sibling exists.
<code>attributes</code>	Acts like a read-only array containing <code>Attr</code> instances representing this node's attributes.
Method	Functionality
<code>hasAttributes()</code>	Returns Boolean indicating whether or not this node has attributes.
<code>hasChildNodes()</code>	Returns Boolean indicating whether or not this node has children.
<code>appendChild(Node)</code>	Adds the argument Node to the end of the list of children of this node.
<code>insertBefore(Node, Node)</code>	Adds the first argument Node to the list of children of this node immediately before the second argument Node (or at end of child list if second argument is null).
<code>removeChild(Node)</code>	Removes the argument Node from this node's list of children.
<code>replaceChild(Node, Node)</code>	In the list of children of this node, replace the second argument Node with the first.

Example1



```
<ol>
```

```
  <li onclick="switchItems(this);">First element of ordered list.</li>
```

```
  <li onclick="switchItems(this);">Second element.</li>
```

```
  <li onclick="switchItems(this);">Third element.</li>
```

```
</ol>
```

```

function switchItems(itemNode) {
  var elementType = window.Node ? Node.ELEMENT_NODE : 1;
  var nextItem = itemNode.nextSibling;
  while (nextItem &&
    !(nextItem.nodeType == elementType &&
      nextItem.nodeName.toLowerCase() == "li")) {
    nextItem = nextItem.nextSibling;
  }
  if (nextItem) {
    itemNode.parentNode.removeChild(nextItem);
    itemNode.parentNode.insertBefore(nextItem, itemNode);
  }
  return;
}
function switchItems(itemNode) {
  var elementType = window.Node ? Node.ELEMENT_NODE : 1;
  var nextItem = itemNode.nextSibling;
  while (nextItem &&
    !(nextItem.nodeType == elementType &&
      nextItem.nodeName.toLowerCase() == "li")) {
    nextItem = nextItem.nextSibling;
  }
  if (nextItem) {
    itemNode.parentNode.removeChild(nextItem);
    itemNode.parentNode.insertBefore(nextItem, itemNode);
  }
  return;
}

```

Find the li Element following the selected one (if it exists)

Returns null if no next sibling

Convert null to Boolean produces false

Remove following element from tree

Re-insert element earlier in tree

Swap nodes if an li element follows

Operate on a node by calling methods on its parent

Example2

Body of original HTML document:

```
<body onload="makeCollapsible('collapse1');">
  <ol id="collapse1">
    <li>First element of ordered list.</li>
    <li>Second element.</li>
    <li>Third element.</li>
  </ol>
  <p> Paragraph following the list (does not collapse). </p>
</body>
```

Effect of executing makeCollapsible():

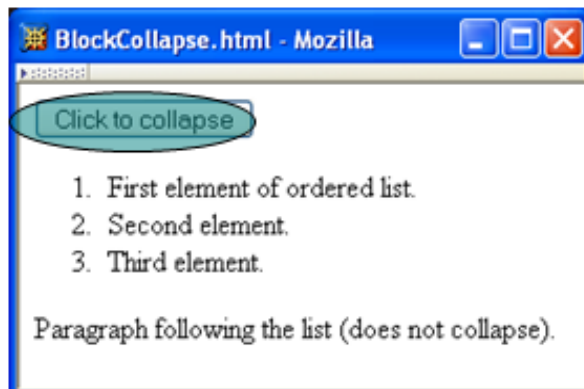
```
<body onload="makeCollapsible('collapse1');">
```

Added
to DOM
tree:

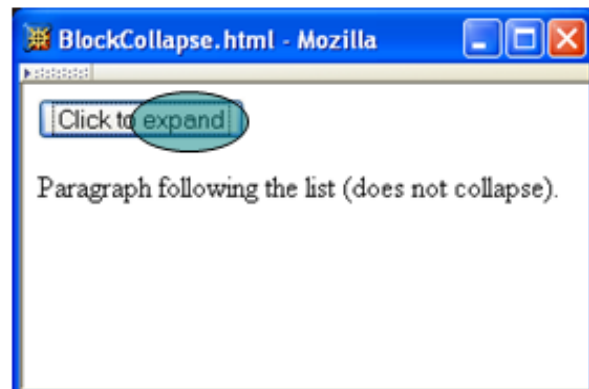
```
<div>
  <button type="button"
    onclick="toggleVisibility(this, 'collapse1')">
    Click to collapse
  </button>
</div>
```

```
<ol id="collapse1">
  <li>First element of ordered list.</li>
  <li>Second element.</li>
  <li>Third element.</li>
</ol>
<p>
  Paragraph following the list (does not collapse).
</p>
</body>
```

Before clicking button:



After clicking button:



```
// Add a button before the specified element (assumed
// to be block style) that will make the element
// disappear when clicked once and re-appear when
// clicked a second time.
// The button is placed within a div to ensure that
// the markup we generate is valid XHTML.
```

Node
creation

```
function makeCollapsible(elementId) {
  var element = window.document.getElementById(elementId);
  if (element) {
    var div = window.document.createElement("div");
    element.parentNode.insertBefore(div, element);
    var button = window.document.createElement("button");
    div.appendChild(button);
    button.setAttribute("type", "button");
    var buttonText = window.document.createTextNode("Click to collapse");
    button.appendChild(buttonText);
    button.setAttribute("onclick",
      "toggleVisibility(this,'" + elementId + "');");
  }
  return;
}
```

Attribute
addition

```
function toggleVisibility(button, elementId) {
  var element = window.document.getElementById(elementId);
  if (element) {
    if (element.style.display == "none") {
      element.style.display = "block";
      button.childNodes[0].data = "Click to collapse";
    } else {
      element.style.display = "none";
      button.childNodes[0].data = "Click to expand";
    }
  }
  return;
}
```

Modifying text.

DOM Event Propagation

- Target of event is lowest-level element associated with event
 - Ex: target is the a element if the link is clicked:


```
<td><a href=...>click</a></td>
```

- However, event listeners associated with ancestors of the target may also be called

Three types of event listeners:

```

<body>
<p id="p1">
| <a id="a1" href="somewhere">Over the rainbow</a>
</p>
<script>
    var target = document.getElementById("a1");
    var ancestor = document.getElementById("p1");
    ancestor.addEventListener("click", listener1, true);
    target.addEventListener("click", listener2, false);
    ancestor.addEventListener("click", listener3, false);
</script>
</body>

```

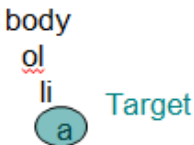
Capturing: Listener on ancestor created with true as third arg.

Bubbling: Listener on ancestor created with false as third arg.

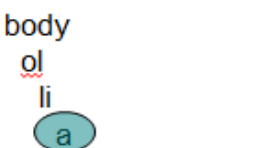
Target: Listener on target element

Priority of event handlers:

1. Capturing event handlers; ancestors closest to root have highest priority



2. Target event handlers



3. Bubbling event handlers; ancestors closest to target have priority.

Propagation-related properties of Event instances:

- eventPhase: represents event processing phase:
 - 1: capturing
 - 2: target
 - 3: bubbling
 -
- currentTarget: object (ancestor or target) associated with this event handler
- Propagation-related method of Event instances:
 - stopPropagation(): lower priority event handlers will not be called
- Typical design:
 - Use bubbling event handlers to provide default processing (may be stopped)
 - Use capturing event handlers to provide required processing (e.g., cursor trail)

Other Common Host Objects

- Browsers also provide many non-DOM host objects as properties of window
- While no formal standard defines these objects, many host objects are very similar in IE6 and Mozilla

TABLE 5.10 Some Common window Methods

Method	Functionality
<code>alert(String)</code>	Display alert window displaying the given String value.
<code>confirm(String)</code>	Pop up a window that displays the given String value and contains two buttons labeled OK and Cancel. Return boolean indicating which button was pressed (<code>true</code> implies that OK was pressed).
<code>prompt(String, String)</code>	Pop up a window that displays the first String value and contains a text field and two buttons labeled OK and Cancel. Second String argument is initial value that will be displayed in the text field. Return String representing final value of text field if OK is pressed, or <code>null/undefined</code> (browser-dependent) if Cancel button is pressed.
<code>open(String, String)</code>	Open a new browser window, and load the URI specified by the first String argument into this window. The second String specifies a name for this window suitable for use as the value of a <code>target</code> attribute in an HTML anchor or form element. Optional String third argument is comma-separated list of “features,” such as the window width and height; see example in text. Returns an object that is a reference to the global object for the new window.
<code>close()</code>	Close the browser window executing this method.

TABLE 5.11 Common window Methods Related to Time

Method	Functionality
<code>setTimeout(String, Number)</code>	Execute (once) the JavaScript code represented by the first argument value after the number of milliseconds specified by the second (integer) argument value has elapsed, unless the timeout is cleared (see next method). Return Number representing an ID for the timeout that can be used to clear it.
<code>clearTimeout(Number)</code>	Clear the timeout having the ID specified by the Number argument.
<code>setInterval(String, Number)</code>	Repeatedly execute the JavaScript code represented by the first argument value every time the number of milliseconds specified by the second (integer) argument value has elapsed, unless the interval timer is cleared (see next method). Return Number representing an ID for the interval timer that can be used to clear it.
<code>clearInterval(Number)</code>	Clear the interval timer having the ID specified by the Number argument.