

Cloud and Web Application

Part 2 : Client-Side Techniques

Table of Contents

- HTTP
- HTML
 - HTML 5
- JavaScript
- DOM
- Ajax
- JSON

HTTP

Client-Server Communication : HTTP

- Hyper Text Transfer Protocol (HTTP)
 - Client-server protocol for transferring Web resources (HTML files, images, styles, etc.)
- Important properties of HTTP
 - Request-response model
 - Text-based format
 - Relies on a unique resource URLs
 - Provides resource metadata (e.g. encoding)
 - Stateless

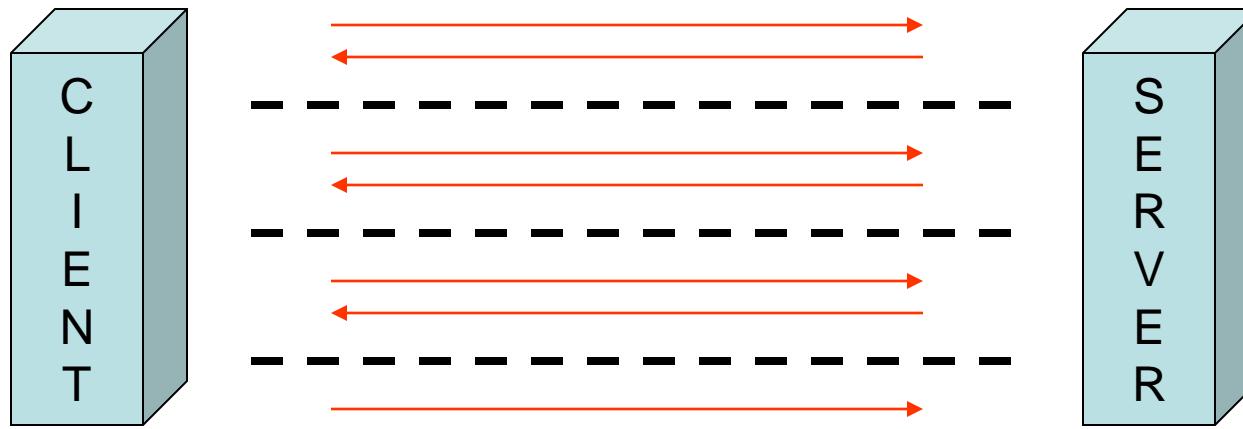
HTTP Transaction: Request-Response

- Client program
 - Running on end host, e.g. Web browser
 - Requests a resource
- Server program
 - Running at the server, e.g. Web server
 - Provides resources



Statelessness

- There is no memory (preservation of state) between HTTP transactions.



- Each HTTP transaction is independent of the one before it and the one after it.
- Cookies can overcome that.

Uniform Resource Identifier (URI)

- Each web resource is identified by a unique URL
- Uniform Resource Locator (URL) is a kind of URI
- General form for a URL:

`<scheme><domain name><port>`

For example,

`http://www.mywebsite.net:80`

(the port is usually omitted and a default is used)

HTTP Example

- HTTP request:

```
GET /academy/about.asp HTTP/1.1
```

```
Host: www.gzas.org.cn
```

```
User-Agent: Mozilla/5.0
```

```
<CRLF>
```

The empty line
denotes the end of
the request header

- HTTP response:

```
HTTP/1.1 200 OK
```

```
Date: Mon, 5 Jul 2013 13:09:03 GMT
```

```
Server: Microsoft-HTTPAPI/2.0
```

```
Last-Modified: Mon, 12 Jul 2010 15:33:23 GMT
```

```
Content-Length: 54
```

```
<CRLF>
```

```
<html><title>Hello</title>
```

```
Welcome to our site
```

```
</html>
```

The empty line
denotes the end of
the response
header

HTTP Request

Request message sent by a client consists of

- **Request line**
 - request method (GET, POST, HEAD, ...)
 - resource URI,
 - and protocol version
- **Request headers**
 - additional parameters
- **Body** – optional data
 - E.g. posted form data, files, etc.

HTTP Request Methods

- Each HTTP request contains a method attribute that identifies its purpose
- Valid methods include
 - ***GET*** retrieve a resource
 - ***POST*** submit data to be processed
 - ***CONNECT*** create a TCP/IP tunnel
 - ***DELETE*** delete a resource
 - ***HEAD*** get response headers only
 - ***OPTIONS*** get a list of supported methods
 - ***PUT*** replace a resource
 - ***TRACE*** echo the request

Main HTTP Request Methods

– GET

- Return the specified resource, run a program at the server, or just download file, ...

– HEAD

- Return the meta-data associated with a resource (headers only)

– POST

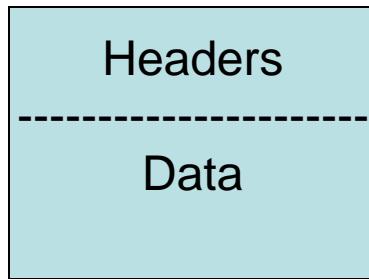
- Update a resource, provide input data for processing at the server, ...

HTTP Response Codes

- Each HTTP response contains a response code that indicates the general outcome
- Response code categories / examples:
 - **1xx**: informational (e.g., “**100 Continue**”)
 - **2xx**: success (e.g., “**200 OK**”)
 - **3xx**: redirection (e.g., “**304 Not Modified**”, “**302 Found**”)
 - **4xx**: client error (e.g., “**404 Not Found**”)
 - **5xx**: server error (e.g., “**503 Service Unavailable**”)
- “**302 Found**” is used for redirecting the Web browser to another URL

HTTP Headers

- Each request and response message begins with header lines that provide meta-information



Request



Response

- Request header data examples:
 - method, resource, protocol version, host
- Response header data examples:
 - protocol version, response code, content type, content length, date

HTTP Headers Example

HTTP Request Message

```
GET /hello.html HTTP/1.1  
Host: www.gzas.org.cn
```

HTTP Response Message

```
HTTP/1.1 200 OK  
Server: Apache-Coyote/1.1  
Content-Type: text/html  
Content-Length: 37  
Date: Fri, 07 Sep 2007 16:13:28 GMT
```

*A blank line separates
message headers from
message body*

```
<html>  
<body>  
Hello!  
</body>  
</html>
```

HTTP POST Request – Example

POST /webmail/login.html HTTP/1.1

Host: www.163.com

Accept: */*

Accept-Language: cn

Accept-Encoding: gzip, deflate

User-Agent: Mozilla/4.0(compatible;MSIE 6.0;
Windows NT 5.0)

Connection: Keep-Alive

Cache-Control: no-cache

Content-Length: 59

<CRLF>

LOGIN_USER=wang

DOMAIN_NAME=tianyang

LOGIN_PASS=122*secret!

<CRLF>

HTTP request line

HTTP headers

The request body
contains the submitted
form data

HTTP Request Handling

- Servers must correctly interpret HTTP request headers and respond appropriately
- **Static content** includes fixed files that are returned without any further computation
- **Dynamic content** is generated on request by an application component that is invoked by the server

HTTP Response

- Typical HTTP response (includes resource)
 - content type
 - length of file
 - content

```
HTTP/1.1 200 OK
```

```
Date: Fri, 31 Aug 2012 23:18:59 GMT
```

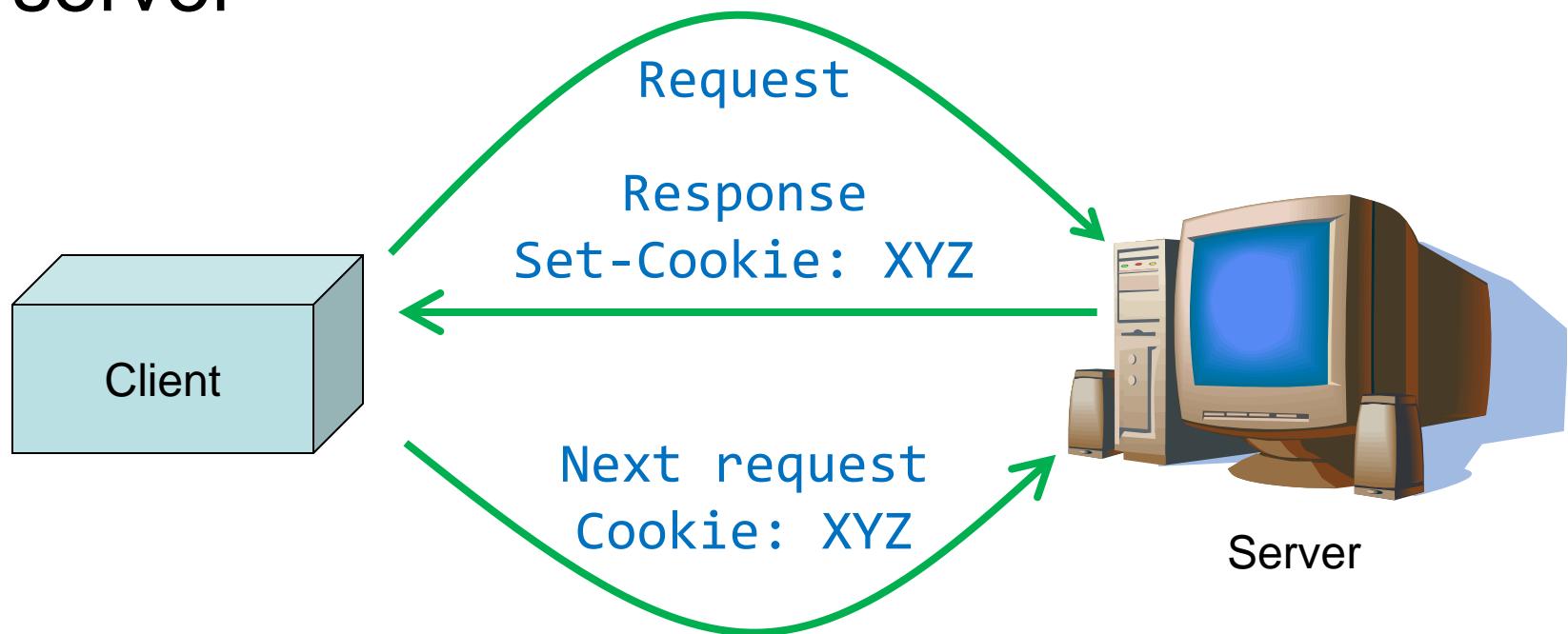
```
Content-Type: text/html
```

```
Content-Length: 1354
```

```
<html>
<body>
. . .
</body>
</html>
```

HTTP Cookies

- Cookie
- Cookies are small pieces of data stored by the client on behalf of the server
- Included in all future HTTP requests to the server



Cookies – Example

- The client requests to Google:

```
GET / HTTP/1.1
```

```
Host: www.google.com
```

- The server sets a cookie in the HTTP response:

```
HTTP/1.1 200 OK
```

```
Set-Cookie: PREFID=c0bf5fd5c3a25209; expires=Wed,  
11-Jul-2013 16:13:22 GMT; domain=.google.com
```

- In further requests to Google the web browser sends the cookie in the HTTP head

```
GET / HTTP/1.1
```

```
Host: www.google.com
```

```
Cookie: PREFID=c0bf5fd5c3a25209
```

HTTP Tools

- **Firebug** plug-in for Firefox
 - A must have for web developers
 - The ultimate tool for monitoring, editing and debugging HTTP, HTML, CSS, JavaScript, etc.
 - Free, open-source – www.getfirebug.com
- **Fiddler** – HTTP proxy
 - Intercepts the HTTP traffic
 - Analyzes the HTTP conversation
 - Free tool – www.fiddler2.com
- **curl** – command line tool

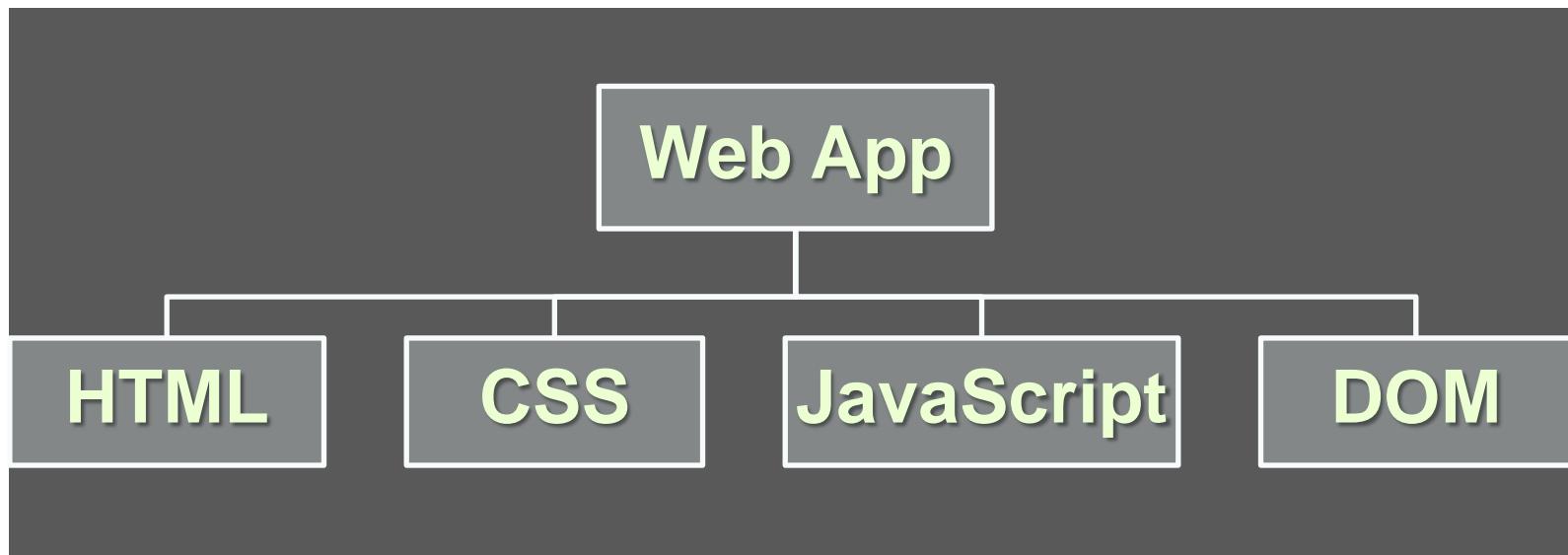


Client-Side Stack

- Five layers:
 - Design: Photoshop/Illustrator
 - Presentation: CSS
 - Interaction: JavaScript, DOM
 - Markup: HTML
 - Browser:

Web Apps in Front-end

- = HTML + CSS + JavaScript



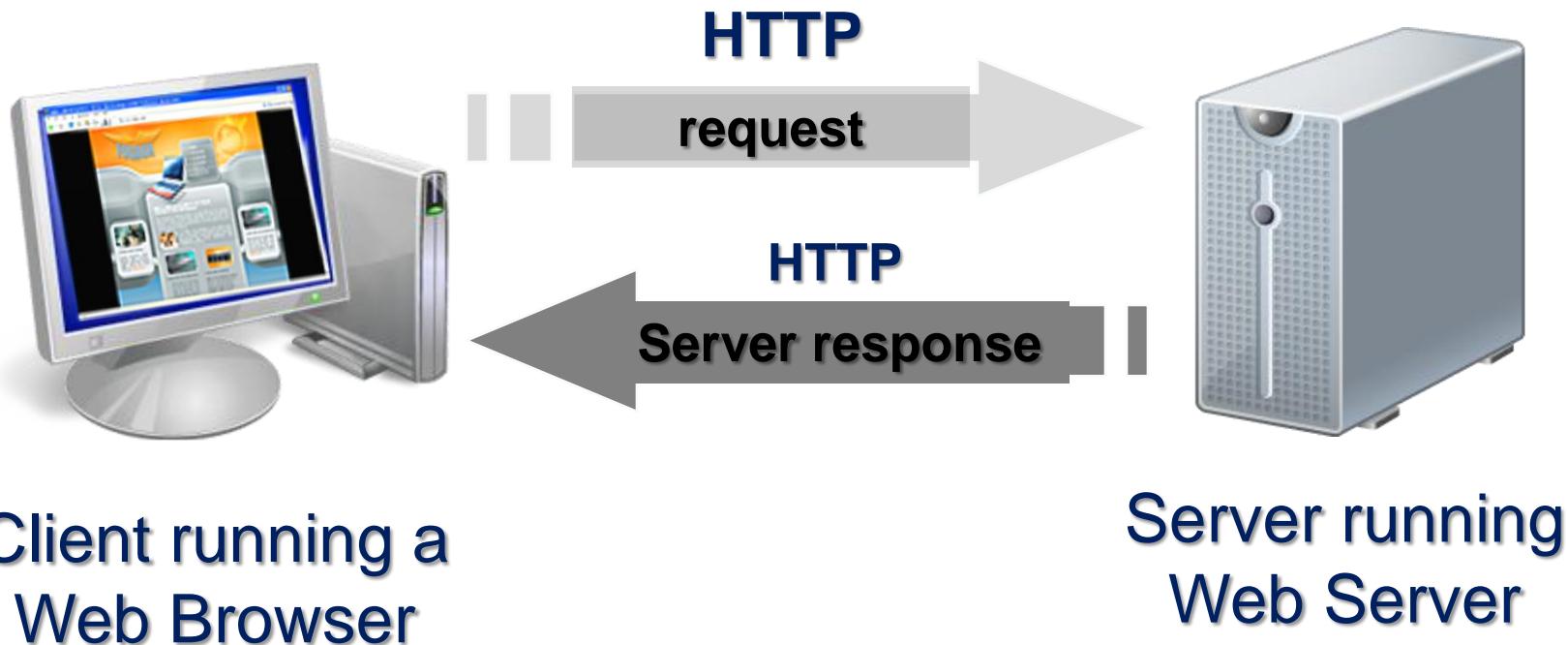
HTML + CSS + JavaScript

- **HTML** defines web page content through semantic tags (headings, paragraphs, lists, ...)
- **CSS** defines 'rules' or 'styles' for presenting every aspect of an HTML document
 - Font (family, size, color, weight, etc.)
 - Background (color, image, position, repeat)
 - Position and layout (of any object on the page)
- **JavaScript** defines dynamic behavior
 - Programming logic for interaction with the user, to handle events, etc.

HTML

How the Web Works?

- Use classical client / server architecture
 - HTTP is text-based request-response protocol



HyperText Markup Language (HTML)

- A language for writing web pages
- HTML provides a notation for describing
 - document structure (semantic markup)
 - formatting (presentation markup)
- HTML is only about structure, not appearance
- Browsers tolerate invalid HTML code and parse errors – you should not.
- The current version is HTML 4.xx?

Creating HTML Pages

- An HTML file must have an `.htm` or `.html` file extension
- HTML files can be created with text editors:
 - NotePad, NotePad ++, PSPad
- Or HTML editors (WYSIWYG Editors):
 - Microsoft FrontPage
 - Macromedia Dreamweaver
 - Microsoft Word

HTML-enabled devices

Small screen
smartphones

320 x 240
4" display
16-bit color



larger screens
(desktop computers)

Desktop HD monitor
1920 x 1080
96 ppi
22" display



moderate screens
(laptops)

Laptop:
15.6" diagonal
1,366 x 768
96 ppi



large screen
devices

Tablet computer:
1024 x 600
170 ppi
7"



Foundations of HTML

Separation of concerns:

- structure
- presentation
- logic
- Data



HTML document

3 major groupings/elements:

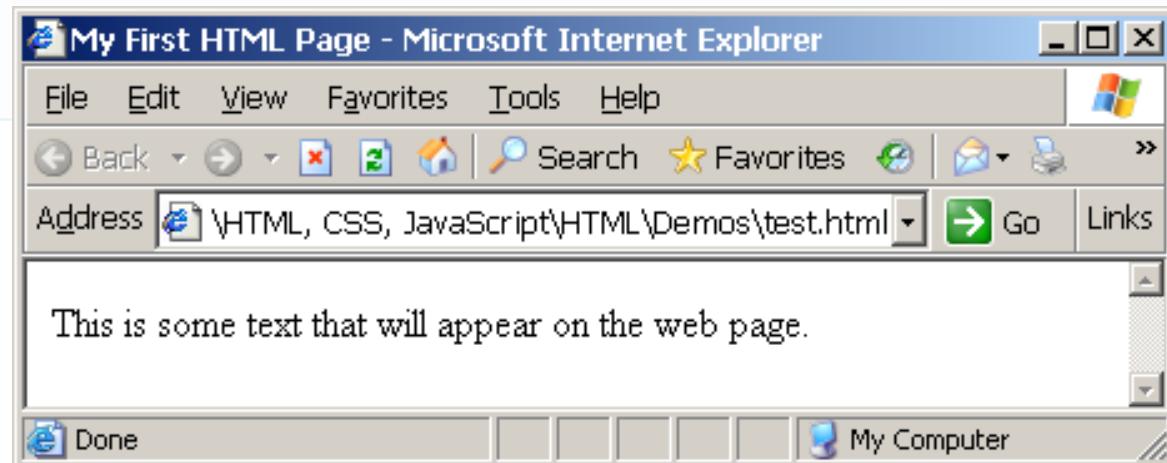
- **<html>** defines which markup language, DTD
- **<head>** defines properties of page, meta, style and js connect
- **<body>** defines page content

HTML Syntax

- Each HTML element is defined by a tag, or pair of tags
 - tags are delimited by angle brackets: < >
 - tags may include attributes (*name*=“*value*”)
- Paired tags have content
 - e.g., paragraph: `<p>Hello!</p>`
- Empty tags contain nothing
 - e.g., `
`, ``

First HTML Page

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>My First HTML Page</title>
  </head>
  <body>
    <p>This is some text...</p>
  </body>
</html>
```



Some Tags

- Hyperlink Tags

```
<a href="http://www.gzas.org.cn/"  
target="_blank">Link to GZAS Web Site</a>
```

- Link to a document called `cat.html` on the same server in the subdirectory `stuff`:

```
<a href="stuff/cat.html">Catalog</a>
```

- Link to a document called `parent.html` on the same server in the parent directory:

```
<a href=".../parent.html">Parent</a>
```

Hyperlinks

- Hyperlinks take the browser to other documents when clicked

- Format:

```
<a href="...url ... ">...link text... </a>
```

e.g.,

- `Current Team`

Tags Attributes

- Tags can have attributes
 - Attributes specify properties and behavior
 - Example: image tag Attribute alt with value "logo"

```

```
 - Few attributes can apply to every element:
 - id, name, style, class, title
 - The id is unique in the document
 - Content of title attribute is displayed as hint when the element is hovered with the mouse
 - Some elements have obligatory attributes

Headings、 Paragraphs and Sections

- Heading Tags (h1 – h6)

```
<h1>Heading 1</h1>
<h2>Sub heading 2</h2>
<h3>Sub heading 3</h3>
```

- Paragraph Tags

```
<p>This is my first paragraph</p>
<p>This is my second paragraph</p>
```

- Sections: div and span

```
<div style="background: skyblue;">
    This is a div
</div>
```

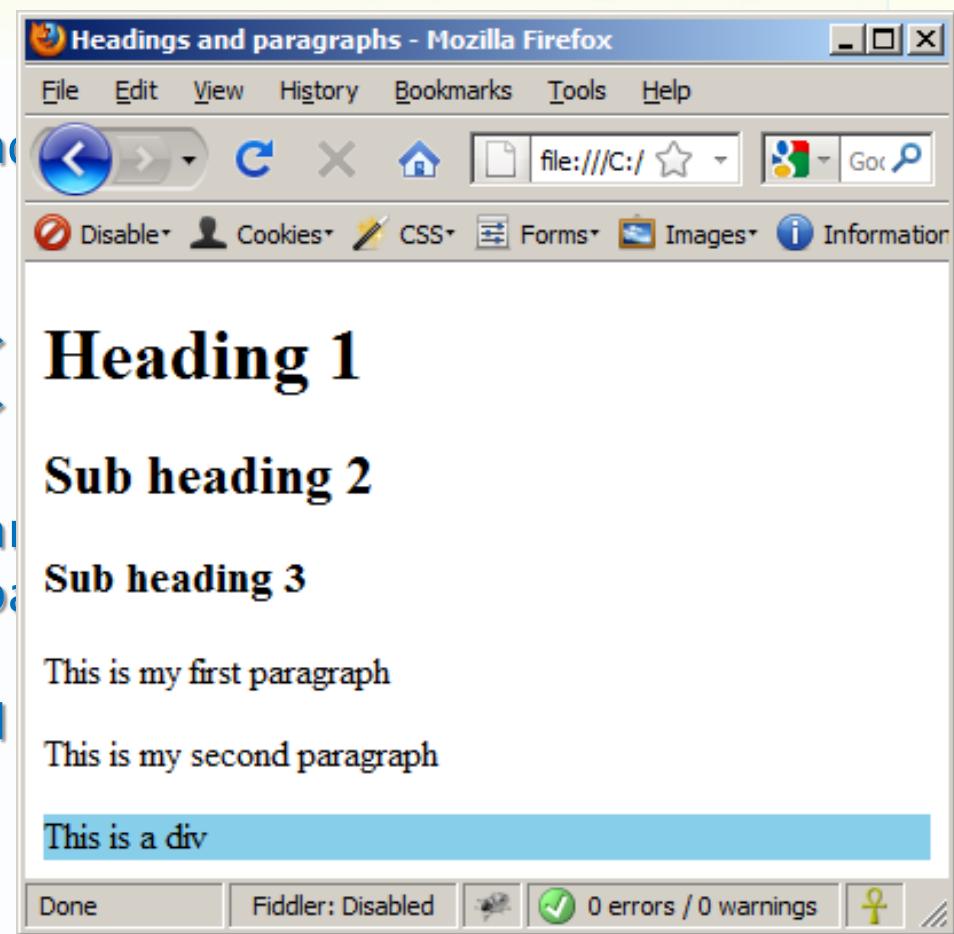
Example

headings.html

```
<!DOCTYPE HTML>
<html>
  <head><title>Headings and paragraphs</title>
  <body>
    <h1>Heading 1</h1>
    <h2>Sub heading 2</h2>
    <h3>Sub heading 3</h3>

    <p>This is my first paragraph</p>
    <p>This is my second paragraph</p>

    <div style="background-color: #ADD8E6">
      This is a div</div>
  </body>
</html>
```

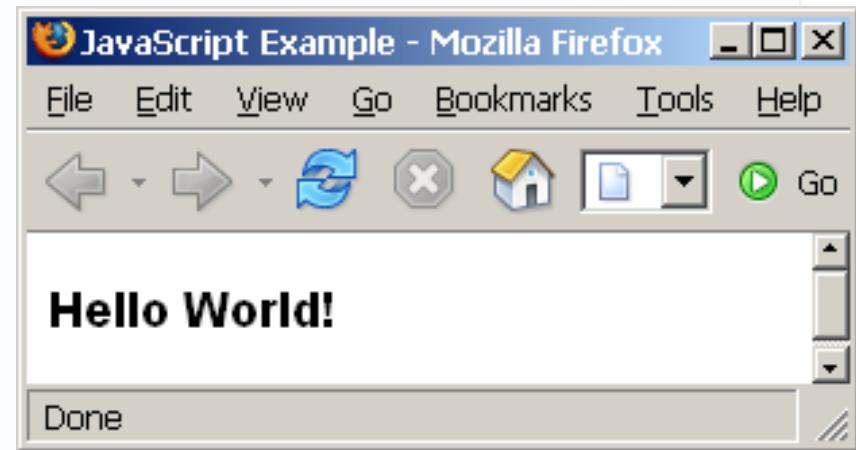


<head> Section: <script>

- The <script> element is used to embed scripts into an HTML document
 - Scripts are executed in the client's Web browser
 - Scripts can live in the <head> and in the <body> sections
- Supported client-side scripting languages:
 - **JavaScript** (it is not Java!)
 - VBScript
 - JScript

The <script> Tag – Example

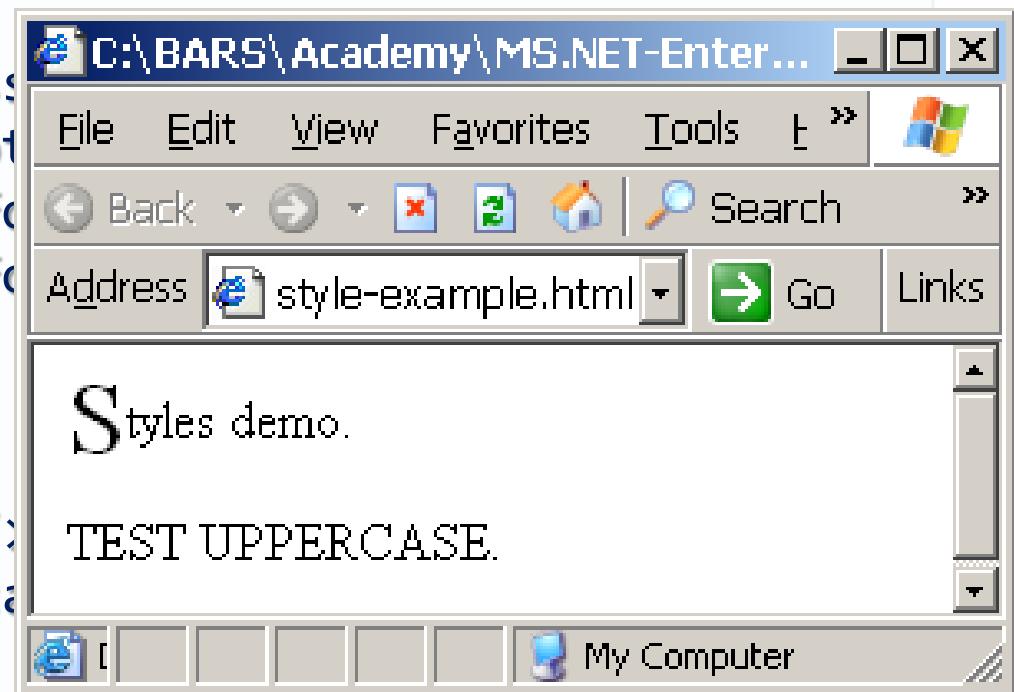
```
<!DOCTYPE HTML>
<html>
  <head>
    <title>JavaScript Example</title>
    <script type="text/javascript">
      function sayHello() {
        document.write("<p>Hello World!</p>");
      }
    </script>
  </head>
  <body>
    <script type=
      "text/javascript">
      sayHello();
    </script>
  </body>
</html>
```



<head> Section: <style>

- The <style> element embeds formatting information (CSS styles) into an HTML page

```
<html>
  <head>
    <style type="text/css">
      p { font-size: 12pt; }
      p:first-letter { font-size: 24pt; }
      span { text-transform: uppercase; }
    </style>
  </head>
  <body>
    <p>Styles demo.<br />
       <span>Test uppercase</span></p>
    </body>
  </html>
```



Lists

- unordered (bullet) list
- ordered (numbered) list
- list item

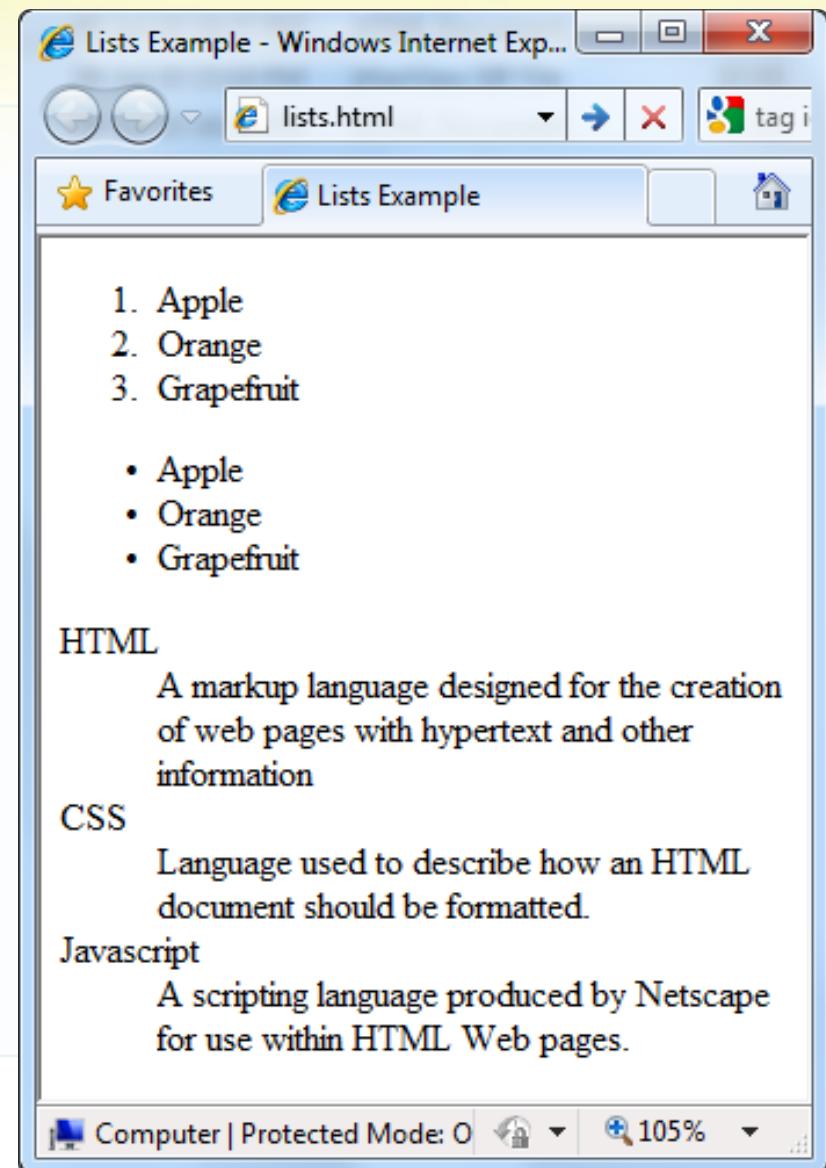
```
<ul>
  <li>
    <a href="standings.html">Current Team</a>
  </li>
  <li>
    <a href="matches.html">This Week's Matches</a>
  </li>
  <li><a href="mailto:info@EHS.org">Contact Us</a></li>
</ul>
```

List Tags – Example

```
<ol type="1">
  <li>Apple</li>
  <li>Orange</li>
  <li>Grapefruit</li>
</ol>

<ul type="disc">
  <li>Apple</li>
  <li>Orange</li>
  <li>Grapefruit</li>
</ul>

<dl>
  <dt>HTML</dt>
  <dd>A markup lang...</dd>
</dl>
```



<div> and Tags

- <div> is a block element that creates logical divisions within a page
- is an inline element for modifying a specific portion of text
- Used with CSS and JavaScript

```
<div id="examDiv" style="font-size:14pt;  
color:red">DIV example</div>  
  
<p>This one is <span id="aSpan" style="font-weight:bold">only a test</span></p>
```



HTML Tables

- An HTML table presents data in a grid format

```
<table border="1">
  <caption>Table Caption</caption>
  <tr>
    <td>R1,C1</td>
    <td>R1,C2</td>
    <td>R1,C3</td>
  </tr>
  <tr>
    <td>R2,C1</td>
    <td>R2,C2</td>
    <td>R2,C3</td>
  </tr>
</table>
```

Table Caption

| | | |
|-------|-------|-------|
| R1,C1 | R1,C2 | R1,C3 |
| R2,C1 | R2,C2 | R2,C3 |

HTML Tables (2)

- Start and end of a table

```
<table> ... </table>
```

- Start and end of a row

```
<tr> ... </tr>
```

- Start and end of a cell in a row

```
<td> ... </td>
```

HTML Forms

- HTML Form presents primary method for gathering data from client
- Form controls include
 - *text field, text area, and password box*
 - *select*
 - *check box*
 - *radio buttons*
 - *drop-down menus*
 - *buttons*
- Often used by JavaScript code

HTML Form Example

The "method" attribute tells how the form data should be sent – via GET or POST request

```
<form name="myForm" id="formId"  
method="post" action="path/to/some-script">  
  ...  
</form>
```

The "action" attribute tells where the form data should be sent

Form Fields

- Single-line text input fields:

```
<input type="text" id="FirstName" value="This  
is a text field" />
```

- Multi-line textarea fields:

```
<textarea id="Comments">This is a multi-line  
text field</textarea>
```

- Hidden fields contain data not shown to the user:

```
<input type="hidden" id="Account" value="This  
is a hidden text field" />
```

Form Input Controls

- Checkboxes:

```
<input type="checkbox" name="fruit"  
value="apple" />
```

- Radio buttons:

```
<input type="radio" name="title" value="Mr." />
```

- Radio buttons can be grouped, allowing only one to be selected from a group:

```
<input type="radio" name="city" value="Lom" />  
<input type="radio" name="city" value="Ruse" />
```

Other Form Controls

- Dropdown menus:

```
<select id="gender" name="gender">
  <option value="Value 1"
    selected="selected">Male</option>
  <option value="Value 2">Female</option>
  <option value="Value 3">Other</option>
</select>
```

- Submit button:

```
<input type="submit" id="submitBtn"
value="Apply Now" />
```

Other Form Controls (2)

- File input – a field used for uploading files

```
<input type="file" name="photo" />
```

- When used, it requires the form element to have a specific attribute:

```
<form enctype="multipart/form-data">  
...  
  <input type="file" name="photo" />  
...  

```



HTML 5

What is HTML5?

- Latest standard for HTML
- HTML5 page start with one DOCTYPE declaration

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

- Getting supported by all major browsers (Chrome, Firefox, Internet Explorer, Safari, Opera)

HTML5 - New Features

- Some of the most interesting new features in HTML5 are:
 - The `<canvas>` element for 2D drawing
 - The `<video>` and `<audio>` elements for media playback
 - Support for local storage
 - New content-specific elements, like `<article>`, `<footer>`, `<header>`, `<nav>`, `<section>`
 - New form controls, like `calendar`, `date`, `time`, `email`, `url`, `search`

HTML5 Layout

HTML5 offers new elements to clearly define different parts:

- <header>
- <nav>
- <section>
- <article>
- <aside>
- <figure>
- <figcaption>
- <footer>
- <details>
- <summary>
- <mark>
- <time>



<nav> Element



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

<nav>
<a href="/html/">HTML</a> |
<a href="/css/">CSS</a> |
<a href="/js/">JavaScript</a> |
<a href="/jquery/">jQuery</a>
</nav>

</body>
</html>
```

Some New Form Attributes

- **autofocus** : get focus when the page loads

```
<input type="text" name="fname" autofocus>
```

- **placeholder** : display a short hint in the input field

```
<input type="text" name="fname" placeholder="First name">
```

- **required** : field must be filled

```
<input type="text" name="usrname" required>
```

- **multiple**: allowed more than one value

```
<input type="file" name="img" multiple>
```

Some Input Types/ Form Elements

- **email**: input an e-mail address

```
<input type="email" name="email">
```

- **number**: restrict on the number accepted

```
<input type="number" name="quant" min="1" max="5">
```

- **url**: input a URL address

```
<input type="url" name="homepage">
```

- **<datalist>**: pre-defined values for list

```
<input list="browsers">
<datalist id="browsers">
  <option value="IE">
  <option value="Firefox">
</datalist>
```

Removed Elements

- The following HTML 4 elements has been removed from HTML5:

<acronym>

<applet>

<basefont>

<big>

<center>

<dir>

<frame>

Note on HTML Editors

- You can easily edit HTML files using a WYSIWYG (what you see is what you get) editor like FrontPage or Dreamweaver, instead of writing your markup tags in a plain text file.
- However, if you want to be a skillful Web developer, we strongly recommend that you use a plain text editor to learn your primer HTML.

Lab: Install NotePad++

- Download NotePad++ from
<http://notepad-plus-plus.org/download/>
- Install NotePad++

JAVASCRIPT

JavaScript Features (1/2)

- A scripting language
 - informal syntax, minimal coding
- Dynamic variable typing
 - boolean, number, string, function, object
 - implicit type conversions
 - potential problem w.r.t. correctness, security
- First-class functions
 - Functions can be passed as variable

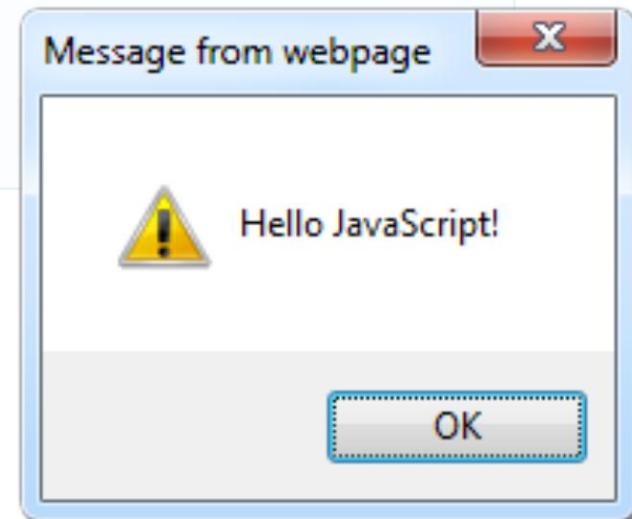
JavaScript Features (2/2)

- Event-driven
 - Programs respond to user interface actions (mouse movement, click, keystroke, etc.)
- Server-Side or Client-Side
 - JavaScript can be used on either platform
- Client-Side functionality
 - JavaScript functions can add, change, or delete any HTML element by using DOM
 - Interaction for user interface

The First Example

first-script.html

```
<html>  
  
<body>  
  <script type="text/javascript">  
    alert('Hello JavaScript!');  
  </script>  
</body>  
  
</html>
```



Using JavaScript

- The JavaScript code can be placed in:
 - <script> tag in the head
 - <script> tag in the body
 - External files, linked via <script> tag
 - Files usually have .js extension

```
<script src="scripts.js" type="text/javascript">
<!-- code placed here will not be executed! -->
</script>
```

- Highly recommended
- The .js files get cached by the browser
- Often placed before the <body> tag

Executing JavaScript

- JavaScript code is executed during the page loading or when the browser fires an event
 - Two blocks of code can't run simultaneously
 - Some statements just define functions that can be called later
- Function calls or code can be attached as "event handlers" via tag attributes
 - Executed when the event is fired by the browser

```

```

Calling Function from Event Handler

```
<html>
<head>
<script type="text/javascript">
    function test (message) {
        alert(message);
    }
</script>
</head>

<body>
    <input type="button" value="Press"
        onclick="test('clicked!')"/>
</body>
</html>
```



JavaScript Syntax

- The JavaScript syntax is similar to C# and Java
 - Operators (`+`, `*`, `=`, `!=`, `&&`, `++`, ...)
 - Variables (typeless)
 - Conditional statements (`if`, `else`)
 - Loops (`for`, `while`)
 - Arrays (`my_array[]`) and associative arrays (`my_array['abc']`)
 - Functions (can return value)
 - Function variables (like the C# delegates)

Data Types

- JavaScript data types:
 - Numbers (integer, floating-point)
 - Boolean (true / false)
- String type – string of characters

```
var myName = "You can use both single or double  
quotes for strings";
```

- Arrays

```
var my_array = [1, 5.3, "aaa"];
```

- Associative arrays (hash tables)

```
var my_hash = {a:2, b:3, c:"text"};
```

String Operations/methods

| | |
|---------------|-----------------------------------|
| indexOf() | finds first instance of character |
| lastIndexOf() | finds last instance of character |
| charAt() | finds character at a position |
| length() | number characters in string |
| toLowerCase() | all letters lower case |
| toUpperCase() | all letters upper case |
| replace() | replaces one string with another |
| match() | looks for matches |
| split() | splits string into pieces |
| toString() | number-to-text, text array |

Arrays Operations and Properties

- Declaring new empty array:

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

- Declaring an array holding few elements:

```
var arr = [1, 2, 3, 4, 5];
```

- Appending an element / getting the last element:

```
arr.push(3);  
var element = arr.pop();
```

- Reading the number of elements (array length):

```
arr.length;
```

Message Boxes

- Alert box with text and [OK] button
 - Just a message shown in a dialog box:

```
alert("Some text here");
```

- Confirmation box
 - Contains text, [OK] button and [Cancel] button:

```
confirm("Are you sure?");
```

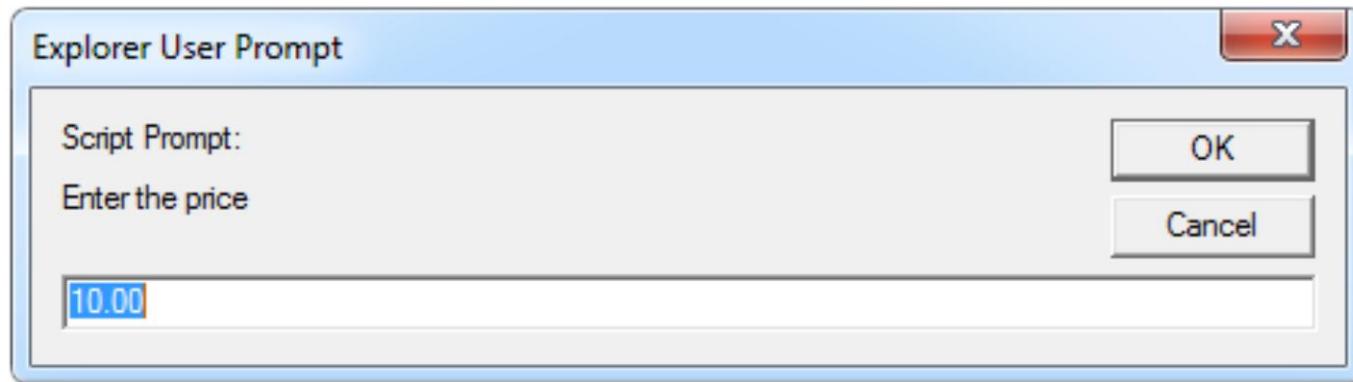
- Prompt box
 - Contains text, input field with default value:

```
prompt ("enter amount", 10);
```

Prompt Box- Example

prompt.html

```
price = prompt("Enter the price", "10.00");
alert('Price + VAT = ' + price * 1.2);
```



Conditional Statement (if)

```
var unitPrice = 1.30;  
if (quantity > 100) {  
    unitPrice = 1.20;  
} else {  
}
```

| Symbol | Meaning |
|---------|--------------|
| > | Greater than |
| < | Less than |
| && | and |
| | or |
| ==, === | Equal |
| != | Not equal |

Switch Statement

- The switch statement works like in C# / Java:

```
switch (variable) {  
    case 1:  
        // do something  
        break;  
    case 'a':  
        // do something else  
        break;  
    case 3.14:  
        // another code  
        break;  
    default:  
        // something completely different  
}
```

Loops

- Like in C# / Java / C++
 - **for** loop
 - **while** loop
 - **do ... while** loop

```
var counter;  
for (counter=0; counter<4; counter++) {  
    alert(counter);  
};  
  
while (counter < 5) {  
    alert(++counter);  
};
```

Function Arguments/Return Value

- Functions are not required to return a value
- When calling function it is not obligatory to specify all of its arguments
 - The function has access to all the arguments passed via arguments array

```
function sum() {  
    var sum = 0;  
    for (var i = 0; i < arguments.length; i++)  
        sum += parseInt(arguments[i]);  
    return sum;  
}  
alert(sum(1, 2, 4));
```

Debugging JavaScript

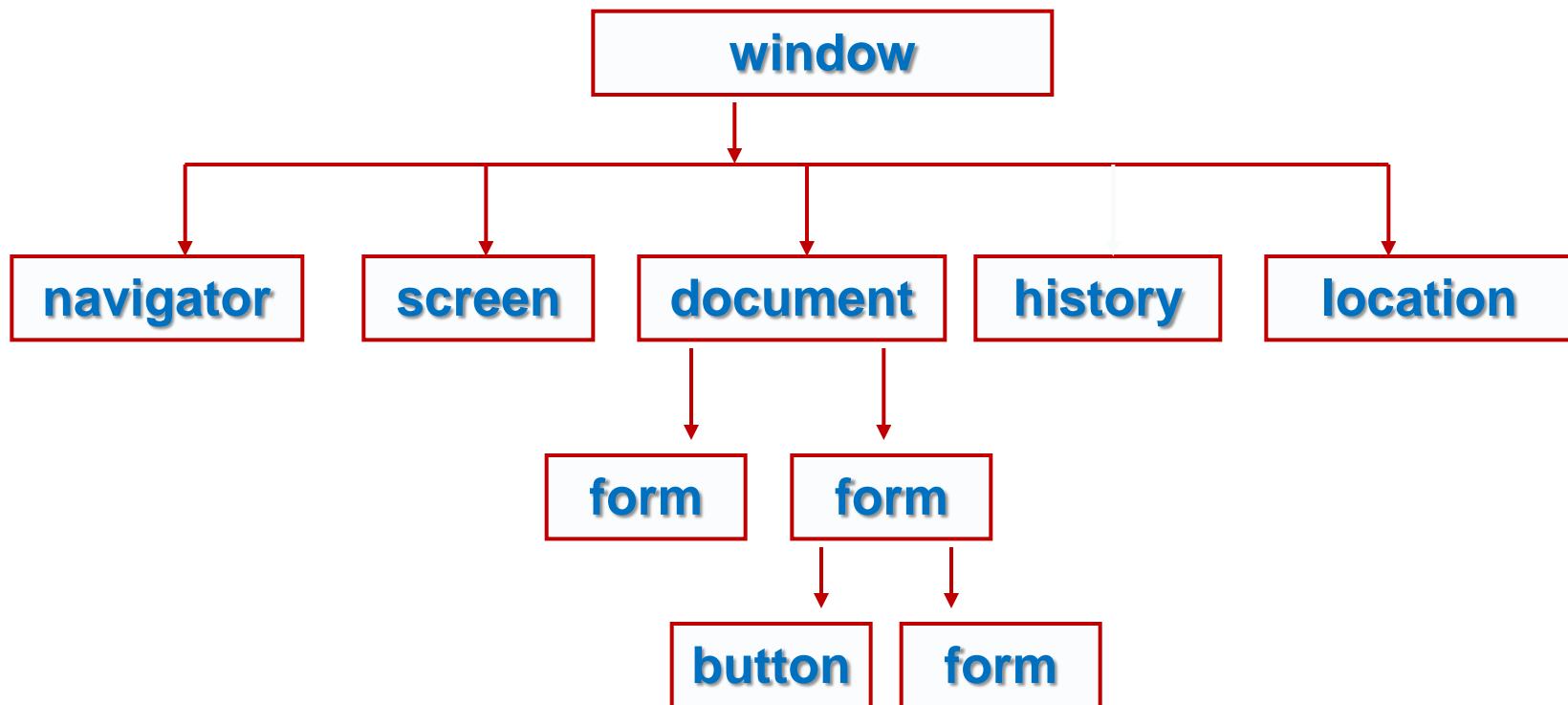
- Microsoft Script Editor in IEs
 - Add-on for Internet Explorer
 - Supports breakpoints, watches
 - JavaScript `debugger` statement opens the debug window
 - Press F12
- Firebug- Firefox add-on for debugging JavaScript, CSS, HTML
 - Supports breakpoints, watches, JavaScript console editor
 - Shows AJAX requests and responses

DOM

Document Object Model (DOM)

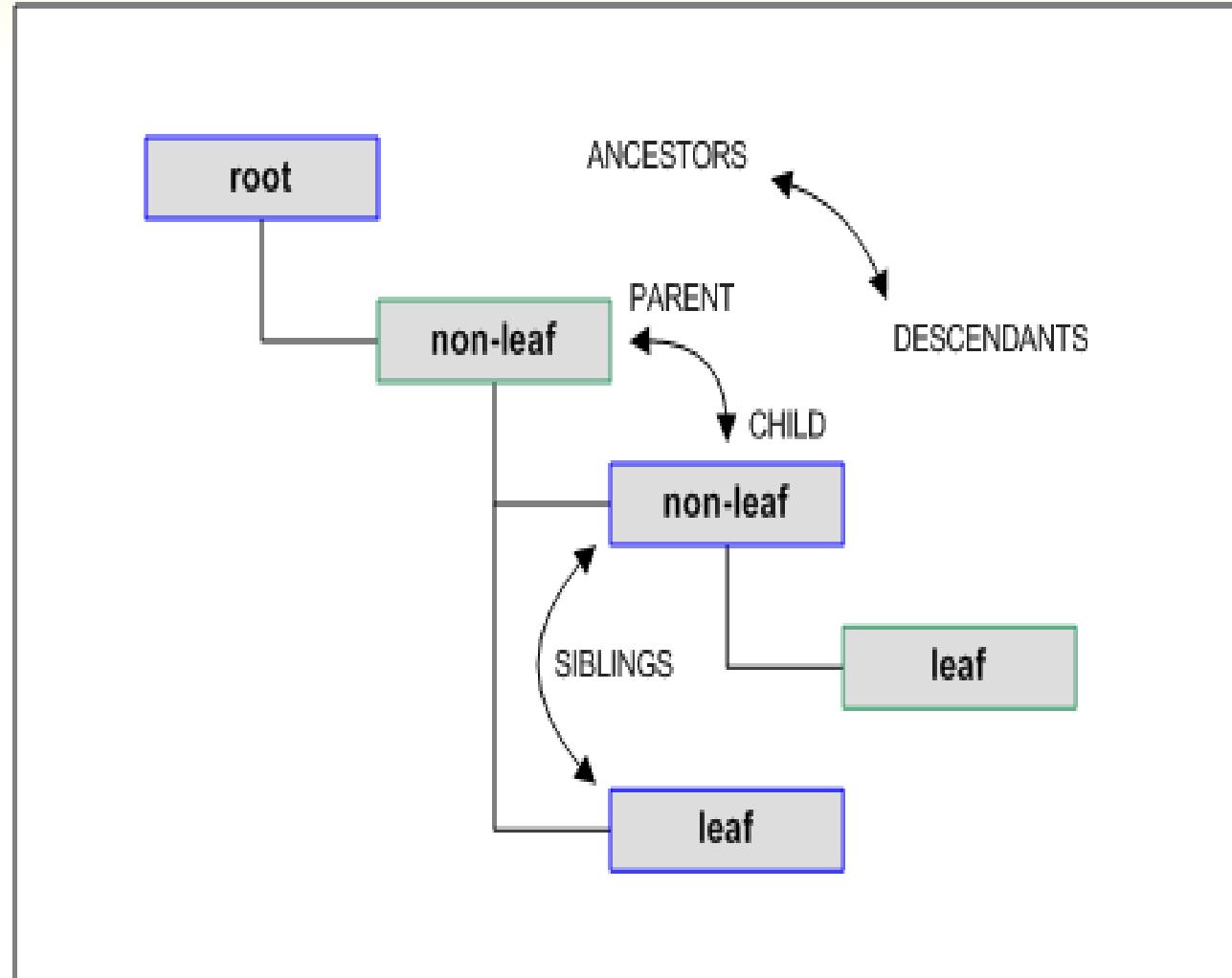
- An interface for manipulating HTML documents
- Document is represented as a tree of objects
- Program can traverse the tree, add/delete/read/write nodes

DOM Hierarchy – Example



DOM Terminology

- `_parentNode`
- `childNodes`
- `firstChild`
- `lastChild`
- `nextSibling`
- `previousSibling`



Built-in Top DOM Objects

- The browser provides some read-only data via:
 - **window**
 - The top node of the DOM tree
 - Represents the browser window
 - **document**
 - holds information the current loaded document
 - **screen**
 - Holds the user display properties
 - **browser**
 - Holds information about the browser

Properties of DOM

- Every HTML element is accessible via the JavaScript DOM API
- Most DOM objects can be manipulated by the programmer
- The event model lets a document to react when the user does something on the page
- Advantages
 - Create interactive pages
 - Updates the objects of a page without reloading it

JavaScript coding in DOM

- **Handling interactions:** *addEventListener*
- *removeEventListener*
- **Accessing elements:** *getElementById* -
parentNode - *childNodes*
- **Manipulating elements:** *createElement* -
appendChild - *removeChild*
- **Setting attributes:** *getAttribute* -
setAttribute - id - className - style - width
- height - left - top

Access Elements through DOM Tree

- We can access elements in the DOM through some tree manipulation properties:
 - `element.childNodes`
 - `element.parentNode`
 - `element.nextSibling`
 - `element.previousSibling`
 - `element.firstChild`
 - `element.lastChild`

DOM Tree Navigation

```
<html>
  <head><title>Headings and paragraphs</title></head>
  <body>
    <h1>Heading 1</h1>
    <p>This is a div</p>
  </body>
</html>
```

- The HTML page itself is called 'document'

```
var objHTML = document.documentElement;           //<html>
var objHEAD = objHTML.firstChild;                 //<head>
var objBODY = objHTML.lastChild;                 //<body>
var objHEAD_1 = objHTML.childNodes.item(0);       //<head>
var objBODY_1 = objHTML.childNodes.item(1);       //<body>
document.write (objHTML + '<br />');
document.write (objHEAD + '<br />');
document.write (objBODY + '<br />');
```

Access Elements for Forms

```
document.myform.FirstName
```

```
<form name="myForm" action="doForm">  
    <input type="text" name="firstName" size="10">  
</form>
```

The diagram shows a form structure with a red dashed border. Inside, there are three input fields: 'First Name' (with a red arrow pointing to its input box), 'Last Name', and 'Age'. Below these is a 'Submit' button.

Access Elements by Id and Others

- Access elements via their ID attribute

```
var elem = document.getElementById("some_id")
```

- Via the name attribute

```
var arr = document.getElementsByName("some_name")
```

- Via tag name

```
var imgTags = el.getElementsByTagName("img")
```

- Returns array of descendant elements of the element "el"

Access Elements

| | |
|-------------------------------|--------------------|
| getElementById("maintitle") | //one object |
| getElementsByName("bodytext") | //array of objects |
| getElementsByTagName("img") | //array of objects |
| getElementsByClass("tt") | //array of objects |

```
<body>
  <h1 name="title" class="tt" id="maintitle">
    <img name="mavi" id="leftfloat">
    <p name="bodytext" id="intro1">
    <p name="bodytext" id="izmirdescription">
      <img name="izmir" id="rightfloat">
    <p name="navtext" id="mainnav">
    <h2 name="title" class="tt" id="secondtitle">
      <p name="bodytext" id="intro2">
      <p name="bodytext" id="balcovadescription">
```

Access Element Properties

- Most of the properties are derived from the HTML attributes of the tag
 - E.g. `id`, `name`, `href`, `alt`, `title`, `src`, etc...
- `innerHTML` – holds all the entire HTML code inside the element.
- Read-only properties with information for the current element and its state
 - `tagName`, `offsetWidth`, `offsetHeight`,
`scrollHeight`, `scrollTop`, `nodeType`, etc...

Changing the Document

- To put information into the document:
 - Create new elements; replace, or append to, existing nodes

```
// Find thing to be replaced
var mainDiv = document.getElementById("main-page");
var orderForm = document.getElementById("target");
// Create replacement
var paragraph = document.createElement("p");
var text = document.createTextNode("Here is the new
text.");
paragraph.appendChild(text);
// Do the replacement
mainDiv.replaceChild(paragraph);
```

.innerHTML

- A property that gets or sets the text between the start and end tags
- Argument completely replaces elements' existing content
- If the new value contains HTML tags, it is parsed and formatted before being placed into the document.
- Example:

```
document.getElementById('fo').innerHTML='<p>tex</p>'
```

<div> and

- Use to identify an element in the DOM
- Get or update its inner content.

```
<body>
    <h1>Test</h1>
    This is some normal text with
    a <span id="fo">span</span> in it.
    <div id="bar">And this is a div.</div>
    Here is some more normal text.
</body>
```

```
...
document.getElementById('fo').innerHTML = "<p>text</p>";
...
```

DOM Events

- JavaScript can register event handlers
 - Events are fired by the Browser and are sent to the specified JavaScript event handler function
 - Can be set with HTML attributes:

```

```

- Can be accessed through the DOM:

```
var img = document.getElementById("myImage");
img.onclick = imageClicked;
```

DOM Events

User-Initiated

- click
- dblclick
- keydown
- keyup
- keypress
- mouseover
- mouseout
- mousedown
- mouseup
- mousemove
- change
- resize
- scroll
- select
- blur
- focus
- reset
- submit

Browser-Initiated

- load
- unload
- error
- abort

Common DOM Events

- Mouse events:
 - **onclick**, **onmousedown**, **onmouseup**
 - **onmouseover**, **onmouseout**, **onmousemove**
- Key events: only for input fields
 - **onkeypress**, **onkeydown**, **onkeyup**
- Interface events:
 - **onblur**, **onfocus**
 - **onscroll**

Common DOM Events (2)

- Form events
 - `onchange` – for input fields
 - `onsubmit`
 - Allows you to cancel a form submission
 - Useful for form validation
- Miscellaneous events
 - `onload`, `onunload`
 - Allowed only for the `<body>` element
 - Fires when all content on the page was loaded / unloaded

Review: Importance Events

| | |
|----------|---------------------------|
| onblur | //exit field |
| onfocus | //entry to field |
| onload | //page entry |
| onselect | //highlight text |
| onchange | //data in field is edited |
| onclick | //when clicked on |
| onsubmit | //when submit form |

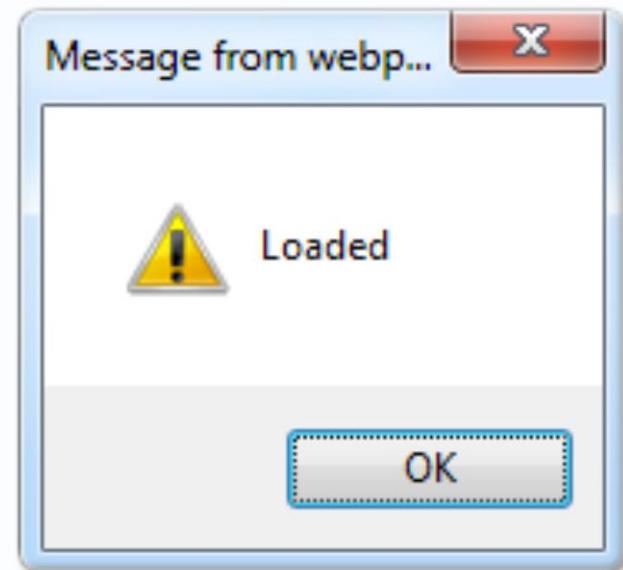
onload Event – Example

.onload.html

```
<html>
<head>
  <script type="text/javascript">
    function greet() {
      alert("Loaded.");
    }
  </script>
</head>

<body onload="greet()" >
</body>

</html>
```



Review: JavaScript/DOM

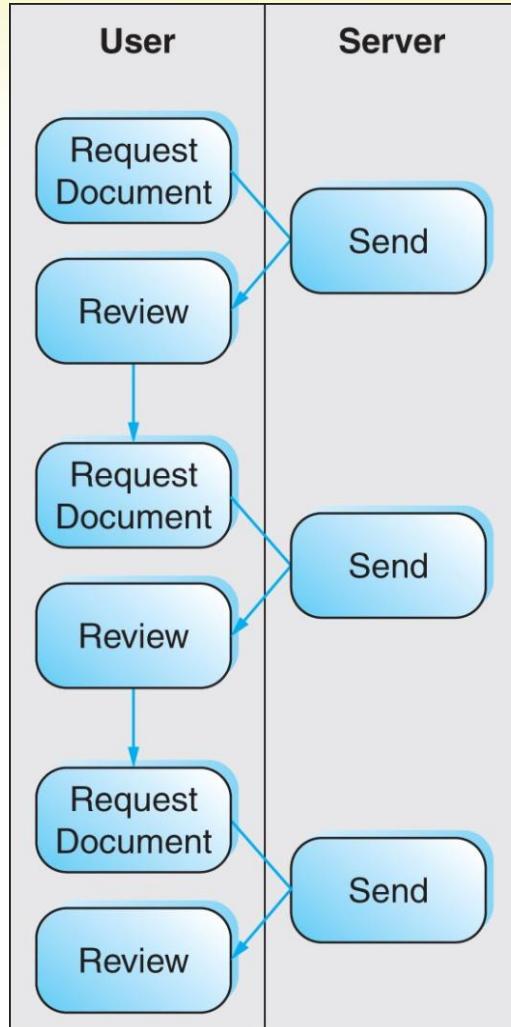
- A scripting language for browsers
 - Can make pages interactive w/o sending requests to server
 - Syntax is very similar to Java, but internals are very different
 - Lots of small differences between browser implementations
- JavaScript can interact with the DOM
 - Examples: Read data from forms, replace parts of the page
 - Can register event handlers with DOM elements to respond to clicks, keypresses, mouse movements, selections...

AJAX

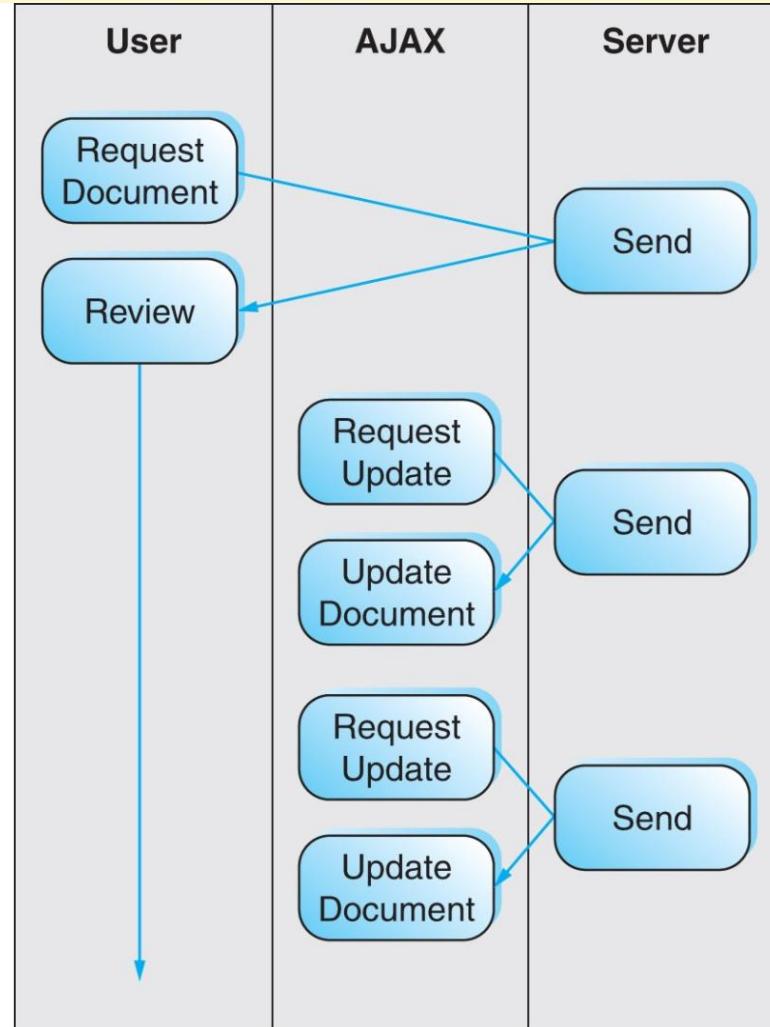
What is Ajax?

- **Asynchronous JavaScript and XML**
 - First mentioned by Jesse James Garrett in 2005
- Not a single technology - a mix of technologies for building faster web apps
 - **HTML** and **CSS** for presentation
 - **DOM** for dynamic display
 - **XML** or **JSON** for data interchange
 - **XMLHttpRequest** for asynchronous requests
 - **JavaScript** for binding everything together

Ajax



(a) Plain HTML



(b) HTML/AJAX

Ajax History

- Cornerstone for web apps.
- Invented by Microsoft; popularized by Google
- Microsoft (1999): XMLHttpRequest with IE 5
- Support from other browsers: 2001-2004
- Gmail (2004)
- Google Maps (2005)

Advantages of Ajax

- Rich and responsive user interfaces
- Reduced HTTP traffic, increased response time.
- Merrill (2008)*:
 - 61% speed increase
 - 73% decrease in bytes transferred
- Full browser support
- Many libraries available

Disadvantages of Ajax

- Security; sandbox/server of origin
- Browser navigation tools defeated
- More complex than traditional web pages
- Increases design/development time

XMLHttpRequest

- A JavaScript object that enables web pages to dynamically load more content
- Request can be **asynchronous**
 - Browser performs the HTTP request in the background while the user continues to interact with the web page
 - Script defines a callback function that should be invoked when the requested content has arrived

XMLHttpRequest workflow

1. Instantiate a new XMLHttpRequest object
2. Prepare the object
 - Call `open()` to set the URL and the method (GET, POST, ...)
 - Can add headers, HTTP authentication, ...
 - Define a `callback function` that will be called by the browser when the results are available
3. Send the request
 - Invoke `send()`, optionally with data to submit (for POST)
4. Handle invocations of the callback function

Ajax Example

```
var xmlhttp = new XMLHttpRequest();  
  
xmlhttp.open('GET', url, true);  
  
xmlhttp.onreadystatechange = function() {  
    if (xmlhttp.readyState == 4) {  
        if (xmlhttp.status == 200) {  
            var mess=document.getElementById("messId");  
            mess.innerHTML = xmlhttp.responseText;  
        } else  
            alert('Ajax failed; status:' + xmlhttp.status);  
    }  
}  
  
xmlhttp.send(null);
```

Ajax ReadyState Values

| Ready State | Significance |
|-----------------|--------------------------------|
| 0 Uninitialized | Request not yet opened |
| 1 Loading | Not yet sent |
| 2 Loaded | Sent; no information available |
| 3 Interactive | Partial response received |
| 4 Completed | Response complete |

XMLHttpRequest properties

| Property | Description |
|--------------------|---|
| readyState | Current state of the object: 0 = UNSENT (before open() is called) 1 = OPENED (before send() is called) 2 = HEADERS_RECEIVED (header+status available) 3 = LOADING (partial data available) 4 = DONE (operation complete) |
| onreadystatechange | Can be assigned a function that is called whenever readyState changes (e.g., to process the response) |
| responseText | Response as text |
| responseXML | Response as a DOM document object (parsed as text/xml) |
| status | Status of the response (a HTTP result code, e.g. 200) |
| statusText | Response string returned by the server (e.g., '200 OK') |

HTTP Respond Status

Status codes:

200 = OK

301 = moved permanently

302 = found/redirected

401 = unauthorized

403 = access denied

404 = not found

```
var xmlhttp = new XMLHttpRequest();
if (xmlhttp.status == 200)
```

Instantiating XMLHttpRequest for Browsers Varies

```
function getXMLHttpRequest() {  
    var xmlhttp = null;  
    try {xmlhttp = new XMLHttpRequest(); }  
    catch(err1) {  
        try{xmlhttp=new ActiveXObject("Microsoft.XMLHTTP"); }  
        catch(err2) {  
            try {xmlhttp = new ActiveXObject("Msxml2.XMLHTTP"); }  
            catch(err3) {xmlhttp = null; }  
        }  
    }  
    if (xmlhttp == null) alert("Error creating xmlhttp !");  
  
    return xmlhttp;  
}
```

Popular JavaScript Libraries

- **jQuery.** One of the most popular libraries. It is lightweight and focused on encapsulating events and Ajax, providing a convenient shortcut to the `getElementById` function, and supplying user interface effects.
- **Prototype.** It has fundamental support for events and Ajax, and has similar—but more extensive—shortcuts for accessing elements of web page; but the differentiator is that Prototype emphasis on classes and it supports object oriented design using inheritance.
- **Dojo.** Dojo starts out with basic support for events and Ajax, and adds widgets—little collections of JavaScript, CSS, and HTML that together create a visual user interface object for displaying or manipulating data. The widgets developed for the library cover a wide range of needs: menus, sortable tables, drag-and-drop trees, calendars, faders, charts, etc.

Ajax without XML

- Despite its name, XMLHttpRequest can handle content other than XML
- JSON is another option to exchange data between client and server

JSON

JSON: JavaScript Object Notation

- Data interchange format
- Passes objects as strings from server to client
- Basic form: uses *name : value* pairs
- Strings in quotes; numbers not

```
{"firstname": "Wang"}
```

```
firstname = "Wang";
```

```
var student={"name": "Ali", "age": 20, "major": "MBBF" };  
  
var nm1=student.name;  
var nm2=student["name"];
```

JSON

- Curly brackets define entities
- Square brackets define arrays of entities

```
var students = [  
    {"name": "Ali", "age": 20, "major": "MBBF"},  
    {"name": "Bahar", "age": 21, "major": "ITF"}  
];  
  
var ag=students[1].age;
```

Working with JSON

- On the client side:
 - Request data from the server
 - When the server returns JSON-encoded string, parsed with `JSON.parse()` and got a JSON object
- On the server side:
 - In the route callback, use `JSON.stringify()` to serialize a got a JSON object into a string
 - Send the string to then client