Lecture 9(-ish)
Web Programming
DOM: Document Object Model

• browser presents an object interface
  – accessible from and modifiable by Javascript

• DOM entities have methods, properties, events
  – element properties can be accessed & changed
  – elements can be added or removed

• document object holds page contents
  – elements stored in a tree: HTML tags, attributes, text, code, ...
  – each element is accessible through the DOM
  – through functions called from Javascript

• page is "reflowed" (smart redraw) when anything changes

• window object also has methods, properties, events
  – alert(msg), prompt(msg), open(url), ...
  – size, position, history, status bar, ...
  – onload, onunload, ...
  – window.document: the document displayed
Basic events on forms

```html
<head>
  <script>
    function setfocus() { document.srch.q.focus(); }
  </script>
</head>
<body onload='setfocus();'>

<H1>Basic events on forms</H1>
<form name=srch
  action="http://www.google.com/search?q="+srch.q.value">
  <input type=text size=25
    id=q name=q value="" onmouseover='setfocus()'>
  <input type=button value="Google" name=but
    onclick='window.location="http://www.google.com/
    search?q="+srch.q.value'>
  <input type=button value="Wikipedia" name=but
    onclick='window.location="http://en.wikipedia.com/
    wiki/"+srch.q.value'>
  <input type=reset onclick='srch.q.value=""'>
</form>
```
More examples...

• in a form:

```html
<form>
  <input type=button value="Hit me"
      onClick='alert("Ouch! That hurt.")'> <P>
  <input type=text name=url size=40 value="http://">
  <input type=button value="open"
      onClick='window.open(url.value)'> <P>
  <input type=text name=url2 size=40 value="http://">
  <input type=button value="load"
      onClick='window.location=url2.value'> <P>
  <input type=button value="color it 
      onClick='document.bgColor=color.value'>
  <input type=text name=color placeholder='type a color'>
  <input type=button value='make it white'
      onClick='document.bgColor="white"'>
</form>
```

• in a tag

```html
<body onLoad='alert("Welcome to my page")'>
```

• on an image

```html
<img src="smiley.jpg" onMouseover='src="frowny.gif"
    onMouseout='src="smiley.jpg"'>
```

• etc.
Dynamic CSS

- style properties can be set dynamically
  - color, alignment, border, margins, padding, ...
  - for individual elements, or all elements of a type, or of a given name
  - can be queried and set by Javascript

```html
<script>
window.onload = function() {
    var p = document.getElementsByTagName("P");
    for (var i = 0; i < p.length; i++) {
        p[i].onmouseover = function() {
            this.style.backgroundColor = "#deadbe";
        };
        p[i].onmouseout = function() {
            this.style.backgroundColor = "white";
        };
    }
}
</script>
```
XMLHttpRequest ("XHR")

- interactions between client and server are usually synchronous
  - there can be significant delay
  - page has to be completely redrawn
- XMLHttpRequest provides asynchronous communication with server
  - often no visible delay
  - page does not have to be completely redrawn
- first widespread use in Google Suggest, Maps, Gmail (Feb 2005)
  - "The real importance of Google's map and satellite program, however, is not its impressive exterior but the novel technology, known as Ajax, that lies beneath."  (James Fallows, NY Times, 4/17/05)
- Ajax: Asynchronous Javascript and XML
  (shorthand/marketing/buzzword term coined 2/05)
  - (X)HTML + CSS for presentation
  - DOM for changing display
  - Javascript to implement client actions
  - XML for data exchange with server (but it doesn't have to use XML)
  - "server agnostic": server can use any technology
Basic structure of Ajax code in browser

```javascript
var req;
function geturl(s) {
    if (s.length > 1) {
        url = 'http://www.cs.princeton.edu/~bwk/phone3.cgi?' + s;
        loadXMLDoc(url);  // loads asynchronously
    }
}
function loadXMLDoc(url) {
    req = new XMLHttpRequest();
    if (req) {
        req.onreadystatechange = processReqChange;
        req.open("GET", url);
        req.send(null);
    }
}
function processReqChange() {
    if (req.readyState == 4) {    // completed request
        if (req.status == 200)     // successful
            show(req.responseText); // could be responseXML
    }
}
function show(s) {   // show whatever came back
    document.getElementById("place").innerHTML = s
}
```
XHR with nested function definition

```javascript
function loadXMLDoc(url) {
    req = new XMLHttpRequest();
    if (req) {
        req.onreadystatechange = function() {
            window.status = req.statusText;
            if (req.readyState == 4) { // completed request
                if (req.status == 200) { // successful
                    show(req.responseText);
                }
            }
        };
        req.open("GET", url);
        req.send(null);
    }
}
```
Callbacks

- **callback**: a function that is passed as an argument to another function, and executed sometime later
  - functions can be passed around like variables
    - e.g., function pointers in C; like ordinary variables in most languages

- extensively used in Javascript because we don't want the browser to block waiting for response
Server script (phone2.cgi)

```
q1=`echo $QUERY_STRING | gawk '{split($0,x,"%20"); print x[1]}'`
q2=`echo $QUERY_STRING | gawk '{split($0,x,"%20"); print x[2]}'`
/usr/local/bin/ldapsearch -x -h ldap.princeton.edu -u -b "
o='Princeton University,c=US' "(cn=*$q1*)" uid cn telephoneNumber 
  studenttelephoneNumber studentstreet street ou |
php -r ' 
while (!feof(STDIN)) {
  $d = (fgets(STDIN));
  if (preg_match("/\^#/", $d)) continue;
  if (preg_match("/\^dn:\|^ufn:/", $d)) continue;
  if (preg_match("/\^cn:/", $d))
    if (strlen($d) > strlen($cn)) $cn = $d;
  if (preg_match("/telephoneNumber\|street/", $d))
    $out = $out . " " . trim($d);
  if (preg_match("/\^ou:/", $d)) $out = $out . " " . trim($d);
  if (strlen(trim($d))==0 && strlen($cn . $out) > 0) {
    $out = trim($cn) . " " . $out;
    $out = preg_replace("/Undergraduate Class of/", "", $out);
    $out = preg_replace("/cn:|ou:|telephoneNumber:|(student)?street:/", "",
    $out = preg_replace("/@Princeton.EDU/", "", $out);
    print "$out\n";
    $out = $cn = "";
  }
}' | grep -i ".*$q2" | sed -e /Success/d
Simpler server script (phone3.cgi)

#!/bin/sh

echo "Content-Type: text/html"; echo

q1=`echo $QUERY_STRING | gawk '{ n=split($0, x, "%20"); print x[1]}'`
q2=`echo $QUERY_STRING | gawk '{ n=split($0, x, "%20"); print x[2]}'`
q3=`echo $QUERY_STRING | gawk '{ n=split($0, x, "%20"); print x[3]}'`

grep -i "$q1" phone.txt | grep -i ".$q2" | grep -i ".$q3"

• works on precomputed data file (caching!)
Libraries, APIs, Frameworks

- browsers are not perfectly standardized
- DOM and CSS coding is messy and complicated
- web services are ever more complex

- how do we make it easy to create applications?

- libraries of common Javascript operations
  - especially access to DOM

- packages for layout with CSS

- API's, often Javascript, to access services

- frameworks: development environments for integrated client & server programming
Javascript libraries

- Javascript functions that typically provide some combination of
  - easier access to DOM
    including covering up incompatibilities
  - convenience functions for arrays, iterators, scope, etc.
  - uniform interface to Ajax
  - visual effects like fading, flying, folding, ...
  - drag and drop
  - in-place editing
  - widget sets / components: calendar, slider, progress bar, tabs, ...
  - templates for generating HTML

- there are lots of such libraries
  - jQuery, Vue, React, Angular, ...
jQuery example

<script>
function geturl(s) {
    if (s.length > 1) {
        var url = 'http://www.cs.princeton.edu/~bwk/phone3.cgi?' + s;
        $.get(url, function(res) {
            $('pre').empty().append(res);
        });
    }
}
</script>
<form name=phone>
Type here:
<input type="text" id="pat" onkeyup='geturl(pat.value);' />
</form>
<pre id="place"></pre>