Lecture 7
Web Technologies
World Wide Web

- a network of interlinked hypertext data

- developed at CERN ’89-’91
  - Tim Berners-Lee proposed the project,
    wrote HTTP, HTML, the first server,
    the first browser (text only), and the first pages

- World Wide Web Consortium (W3C)
  - develops standards for the web
  - Berners-Lee is director
Freedom of Choice!
(a partial list of options)

Wire format:
XML, JSON, REST, ...

Networking, authentication:
TCP/IP, OAuth, CAS, ...

Server (hosting):
OIT MyCpanel, AWS, Heroku, Google Cloud, ...

Business logic:
Java, C#, Python, PHP, Ruby, Node, C++, Objective-C, Swift, Perl, Go, ...

App, CLI, ...

Web client:
HTML, CSS, Javascript, ...

Web frameworks:
Django, Flask, Zend, Rails, Cocoa, Express, ...

Front-end frameworks:
jQuery, React, Angular, Vue, ...

GUI tools:
Swing, jQueryUI, Bootstrap, ...

Devel Environ:
shell++, Eclipse, Xcode, Visual Studio, Android Dev Kit ...

Repository:
Git, Github, SVN, ...

Database:
MySQL, SQLite, Postgres, MongoDB, ...

Networking, authentication:
TCP/IP, OAuth, CAS, ...

Web client:
HTML, CSS, Javascript, ...

Front-end frameworks:
jQuery, React, Angular, Vue, ...

GUI tools:
Swing, jQueryUI, Bootstrap, ...

Devel Environ:
shell++, Eclipse, Xcode, Visual Studio, Android Dev Kit ...

Repository:
Git, Github, SVN, ...

Database:
MySQL, SQLite, Postgres, MongoDB, ...
The Big Picture

- **client – server**
- **URL**
  - making requests
- **HTTP**
  - sending information back and forth
- **HTML**
  - logical structure of a page
- **DOM**
  - hierarchical representation of the HTML
- **CSS**
  - separating appearance/style from logical structure
- **Javascript**
  - dynamic effects
URL: Uniform Resource Locator

- **URL format**
  
  \[\text{protocol://hostname:port/filename}\]

- **hostname** is domain name or IP address

- **protocol or service**
  - http, https, file, ftp, mailto, ...

- **port** is optional
  - defaults to 80 for HTTP

- **filename** is an arbitrary string, can encode many things
  - data values from client (forms)
  - request to run a program on server (cgi-bin)

- **encoded in very restricted character set**
  - special characters as `%hh` (hex), space as `+`
HTTP: Hypertext transfer protocol

• what happens when you click on a URL?

• client sends request:
  
  GET url HTTP/1.0
  [other header info]
  (blank line)

• server returns
  
  header info
  (blank line)

  HTML

  server returns text that can contain content of different types
  uses MIME (Multipurpose Internet Mail Extensions)
  encoded in Base 64
HTML: Hypertext Markup Language

- plain text description of content and markup for a page’s structure
- interpreted by a browser
  - browsers differ in how they interpret HTML, but standardization is improving
- tags with attributes bracket content (very incomplete set):

```html
<html><title>...</title><body>...</body></html>
<h1>...</h1> <h2>...</h2> <p> <em>emphasis</em> </p>
<ul><li>...</li>...</ul>  
<ol><li>...</li>...</ol>
<a href="http://www.google.com">link to Google</a>
<table ... > ... </table>
<script> alert("hello"); </script>
```
DOM: the Document Object Model

- object model and programming interface for what's on a web page

- the DOM describes the logical structure of a page
- a (usually big and complicated) tree
- nested blocks define structure
- used for layout
- provides an API for manipulating content, format, ...
- notification of events that occur in API, e.g., button push

- objects, methods, properties, events
- can perform actions on elements
- can set or change values of properties
Javascript and the DOM

• **JavaScript can**
  – change all the HTML elements in the page
  – change all the HTML attributes in the page
  – change all the CSS styles in the page
  – remove existing HTML elements and attributes
  – add new HTML elements and attributes
  – react to all existing HTML events in the page
  – create new HTML events in the page
CSS: Cascading Style Sheets

• a language for describing presentation of a markup language document
• can control color, size, alignment, position, padding, borders, ...
• style properties can be set by declarations
  – for individual elements, or all elements of a type, or with a particular name
• defined in a separate .css file (best), a style tag in an HTML document, or a style attribute in a tag (worst)

```html
<link rel="stylesheet" href="333.css" />

<style type="text/css" media="all"
    body {background: #fff; color:#000; }
</style>

<p style="color:red;"> </p>
```

• can dramatically change appearance without changing structure or content
• style properties can be queried and set by Javascript
CSS Syntax

• **general format:**
  
  ```
  optional-selector {prop:"val"; prop:"val"; ...}
  ```

• **selectors:**

  ```
  tag       h1 {color: "red";}
  .class    .big {font-size: "200%";}
  #id       #first {padding-top: "10px";}
  :pseudo-class  a:hover {color: "DeepPink";}
  q:lang(fr)  {quotes: "«" "»";}
  ```
<html>
  <style>
    body {background-color: #bobbob}
    p { text-align: left; color: #000088; background-color: #aaaaaa }
    h1, h2 {text-align: center; color: #0000aa; background-color: #888888 }
    ul { color: #008800; background-color: #bobbob }
    .first { padding-top: 20px}
    .last { padding-bottom: 20px }
    #special {color: #880000 }
  </style>
  <body>
    <h1>Primary Header</h1>
    <p class="first">This is the first paragraph.</p>
    <p>This is just another paragraph.</p>
    <h2>Subheader</h2>
    <p id="special">This is a special paragraph.</p>
    <ul>
      <li class="first">First item in the list.</li>
      <li>Another item in the list.</li>
      <li>Yet another item in the list.</li>
      <li class="last">Last item in the list.</li>
    </ul>
    <p>This is just another paragraph.</p>
    <p class="last">This is the last paragraph.</p>
  </body>
</html>
Page layout with HTML and CSS

• use HTML <div> tag for layout (not tables)

<html>
<body style="font-size: 24pt">
  <div id="outer" style="color: #ff0000; background-color: #eeeeaa">
    <p>Here we are in the outer div</p>
    <div id="inner1" style="color: #0000ff; background-color: #00ff00">
      <p>Here we are in inner div 1</p>
      <p>Another paragraph</p>
    </div><!-- inner1 -->
  </div><!-- outer -->
  <div id="inner2" style="color: #0000ff; background-color: #00ff00">
    <p>Here we are in inner div 2</p>
  </div><!-- inner2 -->
  <p>and back in the outer</p>
</body>
</html>
Forms and CGI-bin programs

• "common gateway interface"
  – standard way for client to ask the server to run a program
  – using information provided by the client
  – usually via a form

• if target file on server is executable program,
  – e.g., in /cgi-bin directory
  – and if it has right permissions, etc.,

• server runs it to produce HTML to send to client
  – using the contents of the form as input
  – server code can be written in any language
  – most languages have a library for parsing the input

• OIT offers "Personal cPanel"
  – http://helpdesk.princeton.edu/kb/display.plx?ID=1123
OIT cPanel service

- LAMP environment for individuals, projects, etc.
- personal url, e.g., bwk.mycpanel.princeton.edu
- Perl, Python, PHP (but not very current versions)
- MySQL, phpMyAdmin
- OIT backups, updates, ...
- usage statistics
- web admin interface
- ssh access
HTML form hello1.html

<FORM
  ACTION="http://bwk.mycpanel.princeton.edu/cgi-bin/hello1.cgi"
  METHOD=GET>
<INPUT TYPE="submit" value="hello1: shell script, plain text">
</FORM>

<FORM
  ACTION="http://bwk.mycpanel.princeton.edu/cgi-bin/hello2.cgi"
  METHOD=POST>
<INPUT TYPE="submit" value="hello2: shell script, html">
</FORM>

[and a bunch of others]
Simple echo scripts hello[12].cgi

- **plain text...** (hello1.cgi)

  ```bash
  #!/bin/sh
  echo "Content-type: Text/plain"
  echo
  echo Hello, world.
  ```

- **HTML ...** (hello2.cgi)

  ```bash
  #!/bin/sh
  echo 'Content-Type: text/html
  <html>
  <title> Hello2 </title>
  <body bgcolor=cyan>
  <h1> Hello, world </h1>'
  
  echo "<h2> It's `date` </h2>"
  ```

- no user input or parameters but content can change (as in hello2)
Retrieving information from forms (surv2.py)

- HTTP server passes info to cgi program in environment variables
- form data available in environment variable QUERY_STRING (GET) or on stdin (POST)

```python
#!/usr/bin/python

import os
import cgi
form = cgi.FieldStorage()

print "Content-Type: text/html"
print ""
print "<html>"
print "<title> COS 333 Survey </title>"
print "<body>"
print "<h1> COS 333 Survey </h1>"
for i in form.keys():
    print "%s = %s <br>" % (i, form[i].value)
print "<p>"
for i in os.environ.keys():
    print "%s = %s <br>" % (i, os.environ[i])
```
URL encoding of form data

- **how form data gets from client to server**
  - http://hostname/restofpotentially/very/very/longline
  - everything after hostname is interpreted by server
  - usually /program?encoded_arguments

- **if form uses GET, encoded in URL format in QUERY_STRING environment variable**
  - limited length
  - visible in browser, logs, ...; can be bookmarked
  - usually used if no change of state at server

- **if form uses POST, encoded in URL format on stdin (CONTENT_LENGTH bytes)**
  - sent as part of message, not in URL itself
  - read from stdin by server, no limit on length
  - usually used if causes change of state on server

- **URL format:**
  - keywords in keyword lists separated by +
  - parameters sent as name=value&name=value
  - funny characters encoded as %NN (hex)
  - someone has to parse the string
    - most scripting languages have URL decoders in libraries
Cookies

- **HTTP is stateless**: doesn't remember from one request to next
- cookies intended to deal with stateless nature of HTTP
  - remember preferences, manage "shopping cart", etc.
- **cookie**: one line of text sent by server to be stored on client
  - stored in browser while it is running (transient)
  - stored in client file system when browser terminates (persistent)
- **when client reconnects to same domain,**
  - browser sends the cookie back to the server
    - sent back verbatim; nothing added
    - sent back only to the same domain that sent it originally
    - contains no information that didn't originate with the server

- in principle, pretty benign
- but heavily used to monitor browsing habits, for commercial purposes
PHP  (www.php.com)

• another scripting language for generating web pages
  – Rasmus Lerdorf (1997), Andi Gutmans, Zeev Suraski
  – originally Personal Home Pages, then PHP Hypertext Processor

• sort of like Perl turned inside-out
  – text sent by server after PHP code within it has been executed

```html
<html>
<title> PHP hello </title>
<body>
<h2> Hello from PHP </h2>
<?php
  echo $_SERVER['SCRIPT_FILENAME'] . "<br>";
  echo $_SERVER['HTTP_USER_AGENT'] . "<br>";
  echo $_SERVER['REMOTE_ADDR'] . "<br>";
  echo $_SERVER['REMOTE_HOST'] . "<br>";
  phpinfo();
?>
</body>
</html>
```
Formatter in PHP

```php
<?
$line = ''; $space = '';
$rh = STDIN;
while (!feof($rh)) {
    $d = rtrim(fgets($rh));
    if (strlen($d) == 0) {
        printline();
        print "\n";
    } else {
        #$words = split("/\[s]+/", $d); # doesn't work
        $words = explode(" ", $d);
        $c = count($words);
        for ($i = 0; $i < $c; $i++)
            if (strlen($words[$i]) > 0)
                addword($words[$i]);
    }
}
fclose($rh);
printline();

function addword($w) {
    global $line, $space;
    if (strlen($line) + strlen($w) > 60)
        printline();
    $line .= $space . $w;
    $space = ' ';
}
function printline() {
    global $line, $space;
    if (strlen($line) > 0)
        print "$line\n";
    $line = ''; $space = '';
}
# the \n after the next line shows up in the output!! even if it's removed!!
?>
```
Formatter in Ruby

```
$space = ''
$line = ''

def addword(wd)
    printline() if $line.length() + wd.length() > 60
    $line = "#{$line}#{$space}#{wd}"
    $space = ''
end

def printline()
    print "#{$line}\n" if ($line.length() > 0)
    $line = $space = ''
end

while line = gets()
    line.chomp # get rid of newline
    if (line =~ /^$/)
        printline()
        print "\n"
    else
        line.split().each { |wd| addword(wd) }
    end
end
printline()
```