

Web technologies

- **browser**
 - sends requests to server, displays results
 - DOM (document object model): structure of page contents
- **forms / CGI (common gateway interface)**
 - client side uses HTML/CSS, Javascript, XML, JSON, ...
 - server side code in Perl, PHP, ASP, JSP, ...
extracts info from a form, creates a page, sends it back
- **client-side interpreters**
 - Javascript, Java, Flash, Silverlight, HTML5 (animation, audio/video, ...)
- **Ajax (asynchronous Javascript and XML)**
 - update page content asynchronously (e.g., Google Maps)
- **libraries, APIs, GUI tools**
 - client-side Javascript code for interfaces, effects, ...
jQuery, YUI, Dojo, XUL, ...
- **frameworks**
 - integrated systems for creating web applications
Rails (Ruby), Django (Python), Google Web Toolkit (Java→Javascript), ...
- **mashups**
 - combining data from multiple web sources into single application
- **databases**
- **networks**

Web

- **basic components**

- URL (uniform resource locator)
- HTTP (hypertext transfer protocol)
- HTML (hypertext markup language)
- browser

- **embellishments in browser**

- helpers or plug-ins to display non-text content
pictures (e.g., GIF, JPEG), sound, movies, ...
- forms filled in by user
client encodes form information in URL or on stdout
server retrieves it from environment or stdin
usually with cgi-bin program
can be written in anything: Perl, PHP, shell, Java, ...
- active content: download code to run on client
 - Javascript
 - add-ons and extensions
 - Java applets
 - plug-ins (Flash, Quicktime, Silverlight, ...)
 - ActiveX

HTML and CSS

- plain text description of content and markup for a page
- markup describes presentation (appearance)
- interpreted by a browser
 - browsers differ significantly in how they interpret HTML
- tags bracket content

```
<html><title>...</title><body>...</body></html>
<h1>...</h1> <p> <b>bold</b> <ul><li>...<li>...</ul>
<a href="http://www.google.com">link to Google</a>
<form ... > ... </form>
<table ... > ... </table>
<script> alert("hello"); </script>
```
- and many, many more
- tags can have attributes

```
<font size=-1 color="red"> ... </font>
```
- CSS (cascading style sheets) assign attributes to tags in a more orderly way

URL: Uniform Resource Locator

- **URL format**

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 be created as needed
- can contain encoded material of many different types
uses MIME (Multipurpose Internet Mail Extensions)

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
- **CGI programs can be written in any language**
 - typically Perl, PHP, ASP, JSP, ...
- **OIT CGI facility: campuscgi.princeton.edu**
 - anyone can run CGI scripts; just register
 - restrictions on what scripts can access and what they can do
- **CS department also runs cgi server, with different rules.**

HTML form hello.html

```
<FORM
  ACTION="http://www.cs.princeton.edu/~bwk/temp/hello1.cgi"
  METHOD=GET>
<INPUT TYPE="submit" value="hello1: shell script, plain text">
</FORM>
```

```
<FORM
  ACTION="http://www.cs.princeton.edu/~bwk/temp/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)

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

- HTML ... (hello2.cgi)

```
#!/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)

HTML forms: data from users (surv0.html)

```
<html>
<title> COS 333 Survey </title>
<body>
<h2> COS 333 Survey </h2>
<form METHOD=GET
  ACTION="http://www.cs.princeton.edu/~bwk/temp/surv0">
Name: <input type=text name=Name size=40>
<p> Password: <input type=password name=Pwd
<p> Class: <input type=radio name=Class value=10> '10
          <input type=radio name=Class value=11> '11
<p> CS courses:
      <input type=checkbox name=c126> 126
      <input type=checkbox name=c217> 217
<p> Experience?
      <textarea name=Exp rows=3 cols=40 wrap></textarea>
<p>
      <input type=submit> <input type=reset>
</form>
</body></html>
```

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

Retrieving info 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)

```
#!/usr/local/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)
for i in os.environ.keys():
    print "%s = %s <br>" % (i, os.environ[i])
```

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)

- **an alternative to Perl for 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>
<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

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