Lecture 6
Web Technologies

Web history

- **1989: Tim Berners-Lee at CERN**
  - a way to make physics literature and research results accessible on the Internet
  - wrote HTTP, HTML, the first server, the first browser (text only), and the first pages
- **1991: first software distributions**
- **Feb 1993: Mosaic browser**
  - Marc Andreessen at NCSA (Univ of Illinois)
- **Mar 1994: Netscape**
  - first commercial browser
- **technical evolution managed by World Wide Web Consortium (W3C)**
  - non-profit organization at MIT, Berners-Lee is director
  - official definition of HTML and other web specifications
  - see **`www.w3.org`**

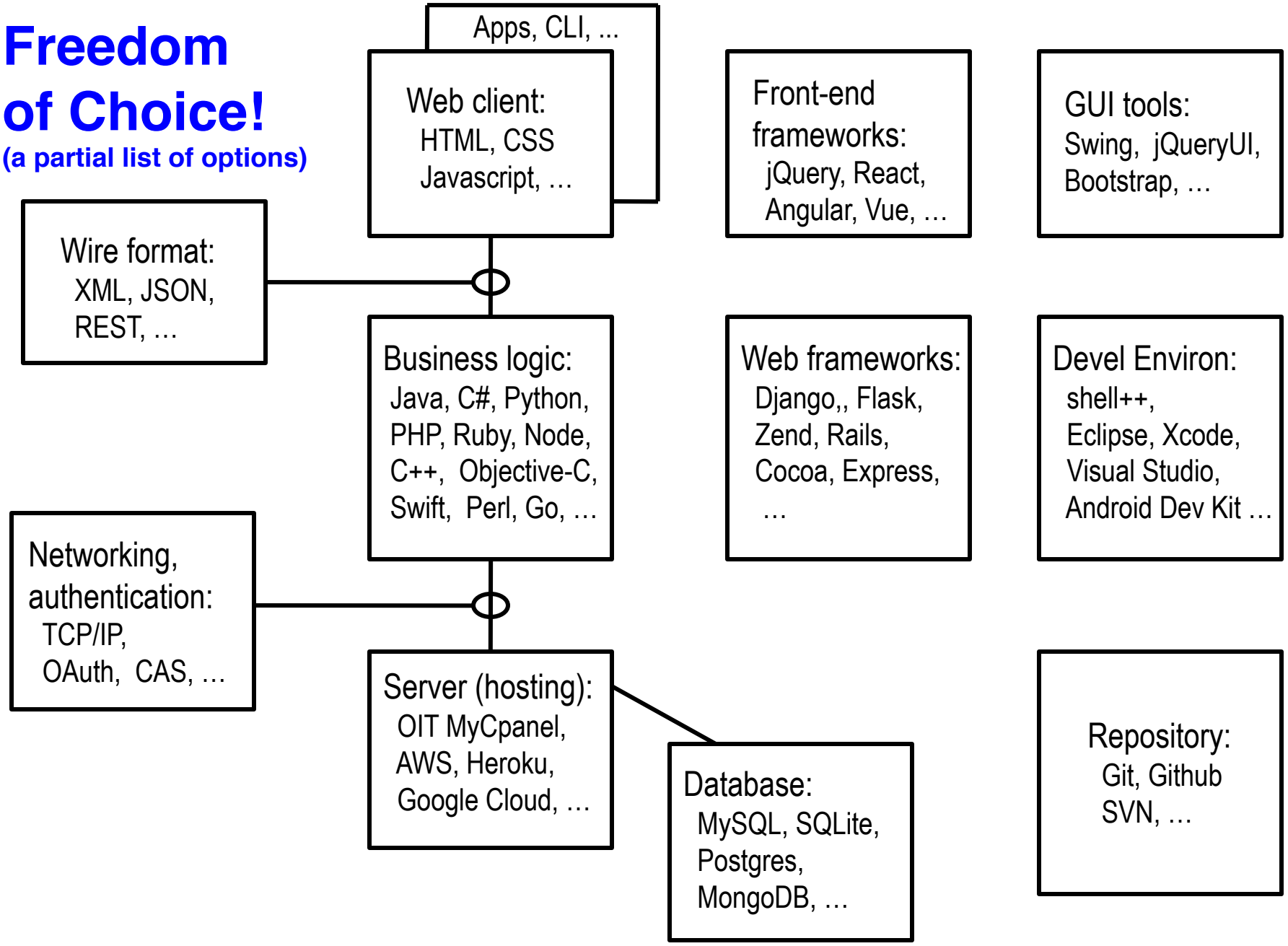
Tim Berners-Lee

Web technologies

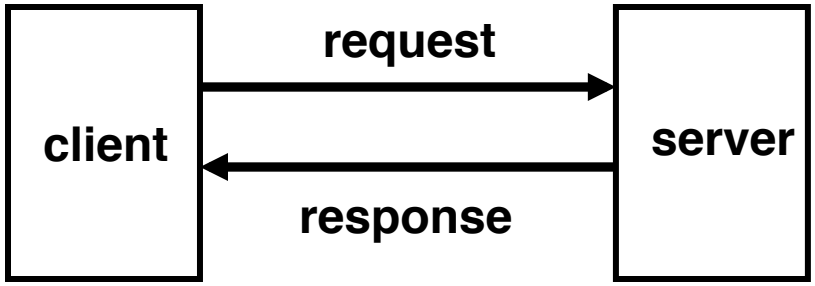
- **client-server architecture**
- **browser**
  - sends requests to server, displays results
  - DOM (document object model): structure of page contents
- **forms / CGI (common gateway interface)**
  - client side uses HTML/CSS, Javascript, XML, JSON, ...
  - server side code in Perl, PHP, Python, Ruby, Javascript, C, C++, Java, ...
    extracts info from a form, creates a response, sends it back
- **client-side interpreters**
  - Javascript, Java, Flash, HTML5 (animation, audio/video, …)
- **Ajax (asynchronous Javascript and XML)**
  - update page content asynchronously **(e.g., Google Maps, …)**
- **libraries, APIs, GUI tools**
  - client-side Javascript for layout, interfaces, effects, easier DOM access, ...
    JQuery, Bootstrap, Angular, React, ...
- **frameworks**
  - integrated server-side systems for creating web applications
    Rails (Ruby), Django, Flask (Python), Express (Javascript), ...
- **databases**
- **networks**
- **hosting: Platform/Infrastructure as a service (PaaS, IaaS)** **[foaas]**

Freedom of Choice!

(a partial list of options)



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, computation, communication

URL: Uniform Resource Locator

- **URL format**

  protocol://hostname:port/filename

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

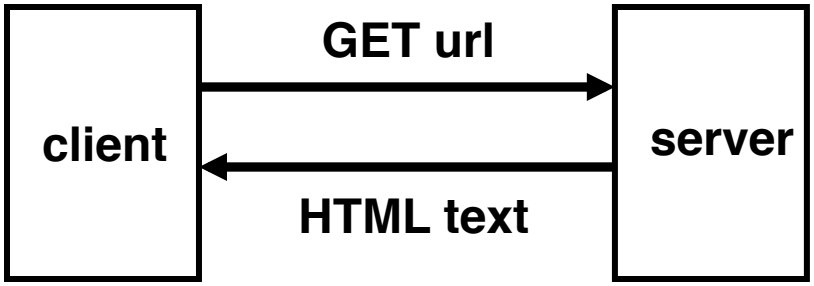
1 - Protocol
2 - Sub-Domain
3 - Domain
4 - Top-Level Domain
5 - Folder/Path
6 - Page

- ***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 encoded content of different types

uses MIME (Multipurpose Internet Mail Extensions) format

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 may interpret HTML differently, but standardization is good
- **tags with attributes bracket content (very incomplete set):**

```
<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>

<img src="http://www.princeton.edu/main/css/images/
  pu-logo-retina.png" />

<table … > … </table>

<script> alert("hello"); </script>
```

PRINCETON UNIVERSITY

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, with methods, properties, events
- can perform actions on objects
- can set or change values of properties

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 elems of a type, or with a specific name
- defined in a separate .css file (best), a style tag in an HTML document, or a style attribute in a tag (worst)

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

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

</style>

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

- can change appearance without changing structure or content
- style properties can be queried and set by Javascript

CSS Syntax

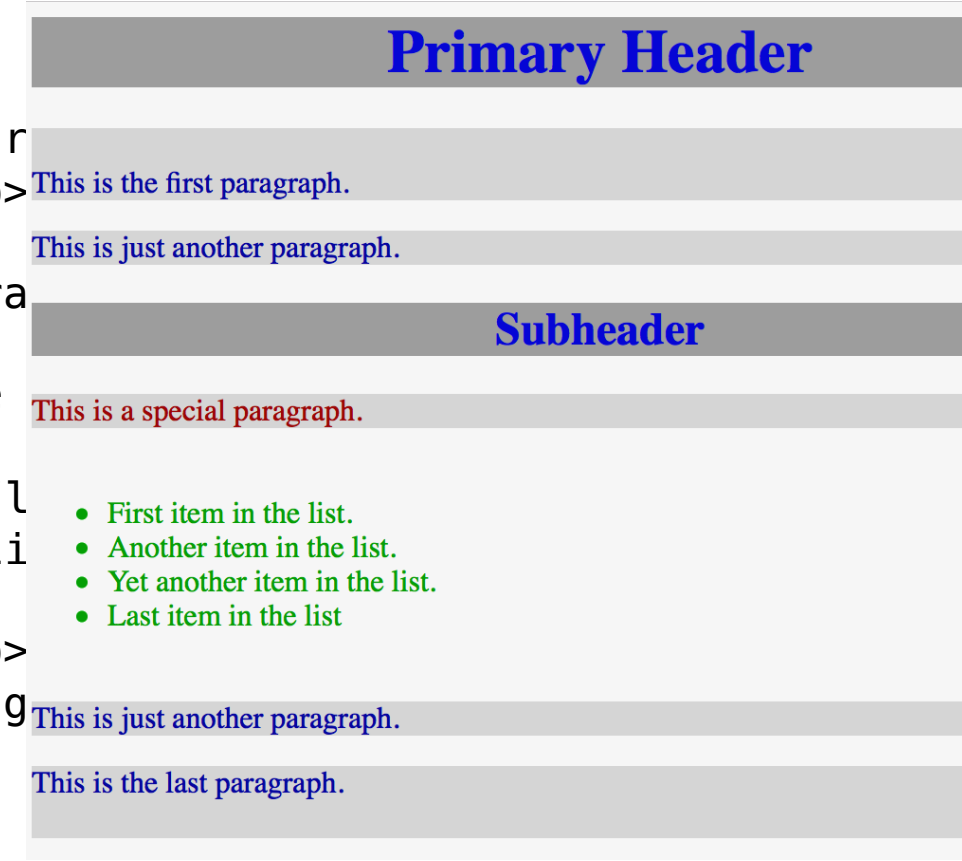
- **general format:**

```
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: "«" "»";}`

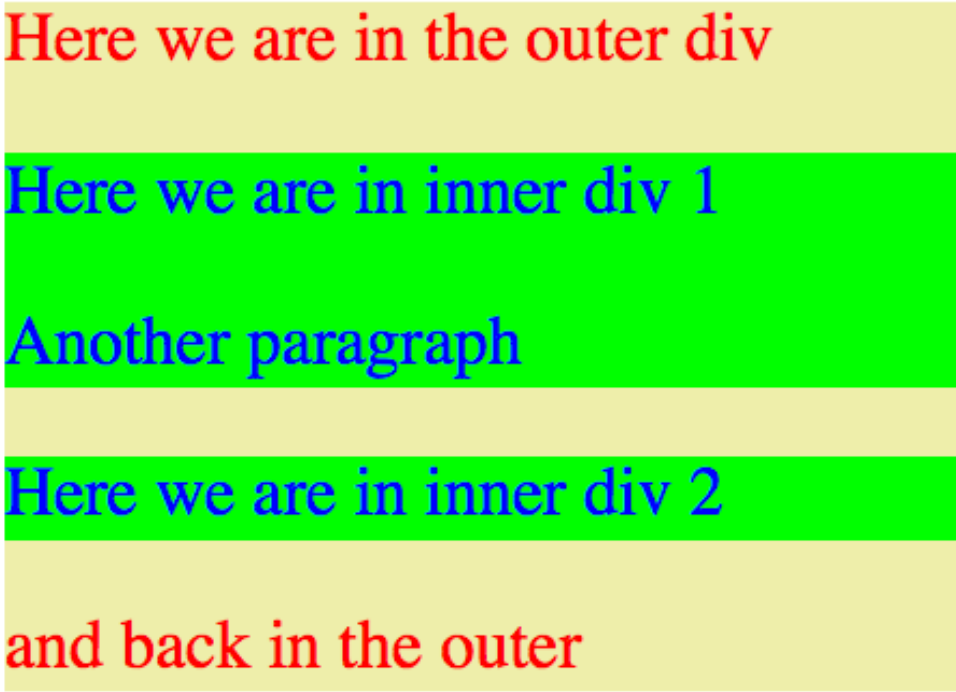
```
<html>
 <style>
   body {background-color: #bbbbbb }
   p { text-align: left; color: #000088; background-color: #aaaaaa }
   h1, h2 {text-align: center;  color: #0000aa; background-color: #888888 }
   ul { color: #008800; background-color: #bbbbbb }
   .first { padding-top: 20px}
   .last { padding-bottom: 20px }
   #special {color: #880000 }
 </style>
 <body>
   <h1>Primary Header</h1>
   <p class="first">This is the first par
   <p>This is just another paragraph.</p>
   <h2>Subheader</h2>
   <p id="special">This is a special para
   <ul>
     <li class="first">First item in the
     <li>Another item in the list.</li>
     <li>Yet another item in the list.</l
     <li class="last">Last item in the li
   </ul>
   <p>This is just another paragraph.</p>
   <p class="last">This is the last parag
 </body>
</html>
```



Page layout with HTML and CSS

- use HTML 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
  <div id="inner1" style="color: #0000ff; background-color: #00ff00">
    <p> Here we are in inner div 1
    <p> Another paragraph
  </div> <!-- inner1 -->
  <div id="inner2"
     style="color: #0000ff;
     background-color: #00ff00">
    <p> Here we are in inner div 2
  </div> <!-- inner2 -->
  <p> and back in the outer
</div> <!-- outer -->
</body>
</html>
```



Bootstrap

Build responsive, mobile-first projects on the web with the world's most popular front-end component library.

Bootstrap is an open source toolkit for developing with HTML, CSS, and JS. Quickly prototype your ideas or build your entire app with our Sass variables and mixins, responsive grid system, extensive prebuilt components, and powerful plugins built on jQuery.

Get started Download

Currently v4.3.1

Screensho

Installation

Include Bootstrap's source Sass and JavaScript files via npm, Composer or Meteor. Package managed installs don't include documentation, but do include our build system and readme.

```
$ npm install bootstrap

$ gem install bootstrap -v 4.3.1
```

BootstrapCDN

When you only need to include Bootstrap's compiled CSS or JS, you can use BootstrapCDN.

CSS only

```
<link rel="stylesheet" href="https://stackpath.boots
```

JS, Popper.js, and jQuery

Official Themes

Take Bootstrap 4 to the next level with official premium themes—toolkits built on Bootstrap with new components and plugins, docs, and build tools.

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

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)**

```
#!/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=POST ACTION=
   "http://bwk.mycpanel.princeton.edu/cgi-bin/surv2.py">
Name: <input type=text name=Name size=40> <p>
Password: <input type=password name=password> <p>
Class: <input type=radio name=Class value=17> '17
       <input type=radio name=Class value=16> '16
<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>
```

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)

```
#!/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)

- **scripting language for generating web pages**
  - Rasmus Lerdorf (1997)
  - 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

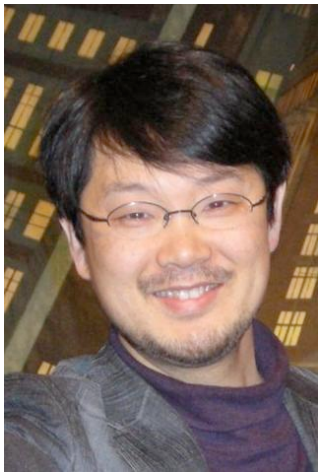
```
<html>
<body>
<h2> Hello from PHP </h2>
<?php
echo $_SERVER["SCRIPT_FILENAME"] . "<br>";
echo $_SERVER["HTTP_USER_AGENT"] . "<br>";
echo $_SERVER["REMOTE_ADDR"] . "<br>";
echo $_SERVER["REMOTE_HOST"] . "<br>";
phpinfo();
?>
</body>
</html>
```

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!!
?>
```

Ruby



- **scripting language inspired by Perl & Smalltalk**
- **Yukihiro Matsumoto ~1995**
- **open source**
- **www.ruby-lang.org**

- **Ruby on Rails**
  - server-side web application framework
  - David Heinemeier Hansson, 2005

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()
```