cis3.2 — electronic commerce — 27 sep 2005 — lecture # 9

today

topics:

- web servers

reading: Ince chapter 6 (sections 1–3)

introduction

- web servers dispense documents that reside on the *world wide web*
- web clients use a *web browser* to request, receive and display the documents
- messages are sent from server to client using *HTTP* (hypertext transfer protocol)

browsers and html

- process
  1. web client runs a browser
  2. browser asks server for a web page
  3. server locates page
  4. server returns page to client ("download")
  5. browser displays page
- locating data
  - web servers have an IP address
  - they also have a URL (named IP)
  - URL = *uniform resource locator*
- components
  - *HTML* (hypertext markup language)
  - a markup language contains *tags* interspersed in the text and images that you actually want to display, indicating formatting instructions
  - document structure: `<html>`, `<head>`, `<body>`, `<title>`
  - character formatting: `<h1>`, `<h2>`, `<i>`, `<b>`, `<font>`
  - paragraph and line formatting: `<p>`, `<br>`
  - including images: `<img>`
  - lists
    - un-ordered list:

```
<ul>
<li>the first list item</li>
<li>the second list item</li>
<li>the third list item</li>
</ul>
```

    * ordered list:

```
<ol>
<li>the first list item</li>
<li>the second list item</li>
<li>the third list item</li>
</ol>
```

    * definition list:

```
<dl>
<dt>the first list item</dt>
<dd>the definition of the first item...</dd>
<dt>the second list item</dt>
<dd>the definition of the second item...</dd>
<dt>the third list item</dt>
<dd>the definition of the third item...</dd>
</dl>
```

    * notes:
        · end of list item tags (e.g., `</li>` and `</dt>` and `</dd>`) are optional
        · lists can be *nested*, which means you can put one list inside another list:

```
my shopping list
<ul>
<li>bread
<li>apples
<li>ice cream
<ul>
<li>vanilla
<li>chocolate
<li>mango
</ul>
<li>peanut butter
</ul>
```

  – special characters
    * forcing spaces: ` `
    * some accents and other special characters:

symbol	code
copyright	`&copy;`
registered trademark	`&reg;`
trademark	`&#8482;`
less than	`<`
greater than	`>`
ampersand	`&`
long dash	`&#8212;`
double quotes	`&quot;`
é	`&eacute;`
è	`&egrave;`
ö	`&ouml;`
ñ	`&ntilde;`

- links and anchors

```
This is a link to <a href="http://www.google.com">google</a>.
This ia a link to <a href="http://www.google.com" target=_blank>google</a>
```

```
that opens up another window.
This is a link that sends me
<a href="mailto:sklar@sci.brooklyn.cuny.edu">email</a>.
```

- tables
  - begin a table with `<table>`
  - end a table with `</table>`
  - begin each row with `<tr>` and end each row with `</tr>`
  - begin each column with `<td>` and end each column with `</td>`
  - options:
    - borders
    - *cellpadding* (padding within a cell)
    - *cellspacing* (spacing between cells)
    - width and height (in pixels)
  - tricks:
    - empty cells — use ` `
    - *spanning* multiple rows or columns
  - coloring cells:
    `<td bgcolor="red">ASDF</td>`
  - aligning cell content:
    - horizontally: left, center, right
    - vertically: top, middle, bottom
- forms
  - provide a way for a user to enter input and send it to the web browser
  - we'll talk about forms later in the semester
- cascading style sheets
  - text color and background color:

```
<style type="text/css">
body { color: black; background: white; }
</style>
```

  - linking to a separate style sheet (in another file):

```
<link type="text/css" rel="stylesheet" href="style.css">
```

    and the "style file" looks like this:

```
/* style.css - a simple style sheet */
body {
color: black; background: white;
}
```

  - margins, left and right indents:

```
      body { margin-left: 10%; margin-right: 10%; }
h1 { margin-left: -8%;}
  h2,h3,h4,h5,h6 { margin-left: -4%; }
    p { text-indent: 2em; margin-top: 0; margin-bottom: 0; }
```

- white space above and below:

```
      h2 { margin-top: 8em; margin-bottom: 3em; }
```

- fonts:
  - styles:

```
em { font-style: italic; font-weight: bold; }
strong { text-transform: uppercase; font-weight: bold; }
```

  - text-transform can be: uppercase, lowercase
  - font families: Verdana, Garamond, "Times New Roman", sans-serif, e.g. `body { font-family: Verdana, sa`
- divisions:
  - borders
  - colors
  - name the divisions using "class"
  - example:
    in the style file:

```
div.box { border: solid; border-width: thin; width: 100% }
```

    in the html file:

```
<div class="box">
The content within this DIV element will be enclosed
in a box with a thin line around it.
</div>
```

- link colors:

```
:link { color: rgb(0, 0, 153) }  /* for unvisited links */
:visited { color: rgb(153, 0, 153) } /* for visited links */
a:active { color: rgb(255, 0, 102) } /* when link is clicked */
a:hover { color: rgb(0, 96, 255) } /* when mouse is over link */
```

- no style sheet?
  - body tag:

```
<body bgcolor="white" background="texture.jpeg" text="black"
link="navy" vlink="maroon" alink="red">
```

    where "link" for unvisited links, "vlink" for visited links, "alink" for the color when you click on the link (before you release the mouse)

- web site design
  - be consistent
  - focus on user content
  - make sure your site is compatible with as many browsers as possible, and looks the same on all browsers

  - don't over-use color
  - don't design pages that take long to download
  - use links so as not to put too much on a page
  - keep links small
  - use conventions for links (like underlining)
  - use style sheets
  - provide printable versions of pages
  - don't put too much text on a page
  - make text readable (kind and accessible colors and fonts)
  - signal the use of multimedia
  - make the *home* page recognizable (distinguishable from the other pages)
  - don't let the user get lost

how web servers work

- a web server is a sophisticated file server
- server can deliver a page and/or execute a program (which delivers results in the form of a page); i.e., *static* vs *dynamic*
- http
  - message (request) from client to server: (1) request line, (2) header lines
  - message (response) from server to client: (1) status line, (2) header lines, (3) content (html)
  - examples p148
  - MIME (multipurpose internet mail extension): correlates file name extensions to applications
- status codes: report errors back to client in a standard way
- *CGI* (common gateway interface): allows client to initiate program execution on the server; instead of fetching a static page, the server executes a program (which probably results in HTML output that goes back to the client which displays the results)
- logging — server keeps track of every IP that makes requests; and what the server does in response; used primarily for: web server performance; marketing analysis