

Distributed Software Development

CSS

Chris Brooks

Department of Computer Science

University of San Francisco

3-2: CSS

- CSS stands for Cascading Style Sheets
- Provides a way to specify how related document elements should be displayed.
 - Makes design and maintenance easier
- Also gives us an opportunity to introduce some topics we'll revisit in other forms:
 - Documents as trees
 - Specifying paths to nodes in trees
 - Separating out the meaning of a tag from its use.

3-3: Display and the Web

- HTML and the Web were initially developed at CERN in 1990.
- Goal - allow information to be shared across a wide variety of platforms and displays.
- HTML deliberately left out specific layout instructions
 - Particular rendering of a document was dependent on client's capabilities.
- Information could be displayed on a wide variety of devices, but there was little presentational control.

3-4: Invasion of the Web Designers

- Early versions of Yahoo! look very different from today.
 - Single font, no table layout, background colors, menus, dynamic behavior, etc.
- Later versions of HTML added presentational elements.
 - Ability to specify fonts within paragraph elements, control color more easily, use tables for layout, etc.
- Problem: these new elements don't convey structural information about a document.

3-5: Invasion of the Web Designers

- Previously: `<h1> Here's a heading of a section </h1>`
- Replacing this with ` Here's the heading ` gives the designer more control.
- But: the markup information is lost.
- Indexing is more difficult.
- No information for devices that can't render Times, or multiple-sized fonts.
- Also, difficult to maintain.

3-6: CSS

- CSS provides the best of both worlds.
- Within the document, tags are specified with structural markup elements. `<h1> Here's a heading </h1>`
- CSS lets you separately specify how to display H1 elements.
 - `h1 {color: red; font: Times, serif; background: #FFEEFF}`
- Display is controlled in a single location
- Clients that can't render some aspect of a display can fallback on HTML.

3-7: CSS rules

- CSS is essentially a set of if-then rules that tell:
 - What parts of a document to match
 - How to format elements that match a rule.
- The cascade details how to deal with conflicts between rules.

3-8: A Simple rule

- A simple rule to force all H1 elements to be large, red, sans-serif font would be: `h1 {font-size: large; font-color: red; font-family: sans-serif; }`
- `h1` is a selector
- The portion inside brackets is a declaration block.
 - Consists of a set of key-value pairs.
- You can also specify a set of elements to be matched: `h2, h3 {color: blue}`

3-9: Matching on document structure

- HTML documents have a hierarchical structure
 - Elements are nested within one another
- We can consider an HTML document as a tree.
- We can then write selectors that match portions of this tree.
 - `ul li {font-size: small }` This matches all list elements underneath a UL tag.
 - `ol li ul li {font-style: oblique}` **Similar.**
 - `p > em {font-style: oblique; color: green }`
Matches children only.
 - `h3 + p {text-indent: 12px }` matches siblings.

3-10: Matching on class

- You can also assign class attribute/value pairs to elements and match on that.
- Period separates element type from class.
 - `<div>` is useful for this.
- `div.cb {font-size: 110%;}`
- **HTML:**`<div class="cb"> 30% labs </div>`

3-11: Matching on ID

- You can also assign unique identifiers to an element and match on those.
- Pound sign separates element type from ID.
 - `ul#leclist {margin: 0px 0px 5px 150px; background-color: #CCBBBB; }`
 - **HTML:** `<ul id="leclist"> ... </div>`
- IDs are unique.
- Many elements can share a class.

3-12: Matching on Pseudo-classes

- CSS also lets you match on pseudo-classes (more accurately called state)
- This includes things like whether the mouse is hovering over an element, or whether a link has been visited:
 - `a {color: #663366}`
 - `a:visited {color: #333366}`
 - `p:hover {background-color: #CCCCCC}`

3-13: Declaration Blocks

- The fun of CSS comes in specifying how different elements should be rendered.
- You can control all the things you'd expect, plus some others.
 - Fonts: size, color, family, style
 - Text alignment: indentation, alignment, spacing, capitalization, underlining/shadowing.
 - Element alignment: margins and layout
 - Background colors, borders, images
 - List bullets, cursors
- The Meyer book and W3Schools do a nice job of enumerating all the options available.

3-14: Conflict resolution

- The *cascade* refers to the precedence rules that are used to determine which rule should apply to an element.
- For example:
 - `h3 { color: blue }`
 - `h3.class1 {color: green }`
- How does the browser know which color to use for the following tags?
 - `<h3> hello </h3>`
 - `<h3 class="class1"> there </h3>`

3-15: Conflict resolution

- (Roughly) The most specific rule wins.
- !important can be used to override this.
 - `p.special {color: #333 !important; background-color: white}`
- Child elements inherit their parents' properties
 - Except for box-model properties: borders, margins, padding, etc.

3-16: Cascading

- What about when two equal-specificity rules apply to the same element?
- Sort rules by:
 - Weight
 - Specificity
 - Order (later is weightier)

3-17: Page layout

- One of CSS's niftiest features is the ability to specify where elements should be placed relative to each other.
- The idea was to give designers something more flexible and appropriate than tables for arranging elements.
- There are two choices for layout:
 - Floating
 - Positioning

3-18: Floating

- Elements can be floated to the left or right.
 - `div.labs {float: right; }`
- Other elements “wrap around” them.
- The browser will avoid overlapping elements.
- This is nice for things like inset images, or drop-in textboxes.

3-19: Positioning

- Positioning gives you more direct control over where an element is placed.
- *Relative* positioning changes the offset of an element relative to the last element placed.
- *absolute* positioning changes the offset of an element relative to the outermost element.
- You can indicate the top, bottom, left, and right of the bounding box.
- You can also control the box's width and height.
- This makes it straightforward to add things like menus and sidebars.

3-20: Example: Two-column layout

- To do a two-column layout in CSS, do:
 - `.left {position: absolute; right: 50%; }`
 - `.right {left: 50%; }`
- **HTML:** `<div class="left">` Here's a bunch of content that should go in the left-hand column.
`</div>`
`<div class="right">` Here's stuff that should go in the right-hand column `</div>`

3-21: Summary

- CSS gives presentational control without sacrificing document markup.
- Allows you to specify classes or elements that behave similarly
- Provides single point of change
- Provides an alternative layout mechanism to tables.
- Specifies transformations in terms of if-then rules
 - This is an approach we'll see again
- Makes it possible to develop nice-looking Web pages that can actually be maintained.