

# Distributed Software Development CSS

Chris Brooks

Department of Computer Science  
University of San Francisco

# 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 how it is displayed.

# 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.

# 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.

# Invasion of the Web Designers

- Previously: `<h1> Here's a heading of a section </h1>`
- Replacing this with `<font size="+3" face="Times" color="blue"> Here's the heading </font>` 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.

# CSS

- CSS is intended to provide 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: bold Times, serif; background: #FFEEFF}`
- Display is controlled in a single location
- Clients that can't render some aspect of a display can fall back on HTML.
  - graceful degradation - this is a principle we'll return to in other contexts.

# 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.

# 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}`

# 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.

# Matching on class

- You can also assign class attribute/value pairs to elements and match on that.
- Period separates element type from class.
- `<div>` and `<span>` are useful for this.
  - `div.cb {font-size: 110%;}`
  - HTML:`<div class="cb"> 30% labs </div>`
  - `span.quote {font-style: italic}`
  - HTML: Bart said: `<span class="quote">Don't have a cow, man!</span>`
- You can also specify a class without an element type:
  - `.urgent { color: red; }`
  - This will be applied to any element with the 'urgent' class

# Multiple classes

- An element can have multiple classes:
  - `<div class="stuff things"> content here </div>`
- We can write CSS rules that match either or both of these
  - `div.stuff { font-size: 100%; }`
  - `div.things { font-size: 120%; }`
  - `div.stuff.things { font-size: 150%; }`

# 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"> ... </ul>`
- IDs are unique.
- Many elements can share a class.

# Matching on Attribute

- You can also use the [ ] characters to match on attributes.
- For example, let's suppose you want to give all images that have an "alt" attribute a red border:
  - `img[alt] {border: 3px solid red;}`
- You can also test attribute values:
  - `img[alt="foo"] {border: 3px solid red;}`
  - `a[href="http://www.usfca.edu"][title="USF"]  
{color: green }`

# 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.
- Colon is used as the element/pseudo-class separator
  - `a {color: #663366} a:visited {color: #333366}`
  - `p:hover {background-color: #CCCCCC}`

# 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.

# Conflict resolution

- The *cascade* refers to the 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 elements?
  - `<h3> hello </h3>`
  - `<h3 class="class1"> there </h3>`

# Conflict resolution

- 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.

# Cascading

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

# 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

# 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.

# 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.

# Summary

- CSS gives presentational control without sacrificing document markup.
  - Separates logical structure from display
- 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 with XSLT
- Makes it possible to develop nice-looking Web pages that can actually be maintained.