Cascading Style Sheets

CSE/ISE 102: Introduction to Web Design (Section 02)
Stony Brook University

Overview of Topics

- CSS Basics
- Formatting Text
- Colors and Backgrounds
- Working with the Box Model
- Page Layout with CSS
- Transitions and Animations

CSS Fundamentals

Why Use Style Sheets?

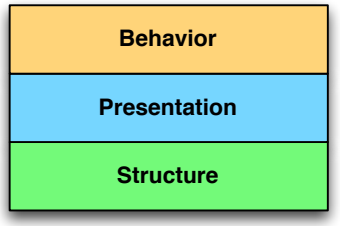
- Precise type and layout controls
- Less work — one style sheet can control an entire site's appearance
- More accessible sites — HTML can focus on meaningful semantic markup
- Reliable browser support

Style Sheets: 1–2–3

1. Mark up your document in HTML
2. Write style rules for different elements
3. Attach the style rules to your document

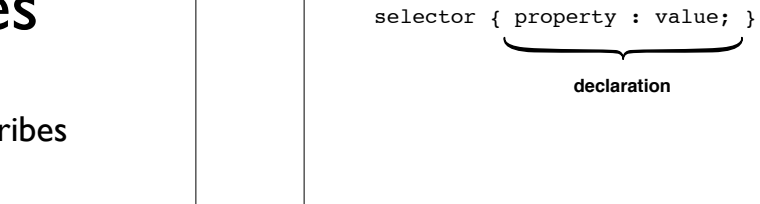
Document Markup

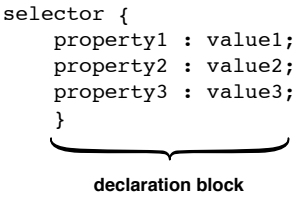
- Markup creates the structure of the document
- Choose elements that describe content meaning, not its appearance



Writing Style Rules

- A *rule* is a style instruction that describes how an element should be displayed
- Each rule *selects* an element and *declares* how it should look
  - A declaration is made up of a *property* and its *value*
  - Declarations go inside curly brackets






```
h1 { color: green; }
 p { font-size: small;
     font-family: sans-serif;
   }
```

Attaching Styles

- Three techniques:
  1. External style sheets (text file with a .css extension)
  2. Embedded style sheets (`<style>` tags in the document head)
  3. Inline styles

Aside: CSS Comments

- CSS comment syntax uses `/*` and `*/` to surround comment text:

```
/* this is a comment */
```

- Comments may span multiple lines

External Style Sheets

- An external style sheet is a plain text document containing at least one style rule
  - it may *NOT* include any HTML tags
  - it should have a *.css* suffix
- We can link to an external style sheet in two ways: `link` and `@import`

Using `link`

- Use a `link` element in the document head:

```
<link rel="stylesheet"
   href="/path/to/sheet.css">
```

  - `rel` always has the value "stylesheet" when linking to a CSS style sheet
- A document may have multiple `links` in it

Importing Styles

- Use an `@import` rule in your `<style>` element:

```
@import url("path/style.css")
```

- `@import` must come before *any* selectors
- An external style sheet may import other style sheets for a modular structure

Embedded Style Sheets

- Style rules are placed inside `<style>` tags in the document head
- Only apply to the current document

```
<style>
    p { font-size: small; }
</style>
```

Inline Styles

- Apply to a single element
  - Can be used to override styles from an embedded or external style sheet
- Use the `style` attribute:

```
<h1 style="color: red;">Contact</h1>
```

Big CSS Concepts

- Inheritance
- Conflicting Styles
- The Box Model
- Grouped Selectors

Inheritance

- Styled HTML elements pass down certain style properties to their "inner" elements
  - e.g., *em* elements inherit *p* properties
- Think of the document tree as a family tree
  - the elements inside a given element are its *descendants*

Conflicting Styles

- We can apply several style sheets to the same document
  - different weights apply to various sources of style information

Which Style Prevails?

- When several style sources vie for control of a given element, style information "cascades" down until it is overridden
- Who wins depends on the presence of various style sheets, specificity, and rule order
  - first choose a style sheet, then look at the rule level

Style Sheet Hierarchy

- Browser default settings
- User style settings (reader style sheets)
- Linked external style sheets
- Imported style sheets
- Embedded style sheets
- Inline style information
- Author-specified **`!important`** style rules
- User-specified **`!important`** style rules

Specificity

- Use the selector type to determine who wins when two or more style rules conflict
  - more-specific selectors have more weight
- We'll talk more about this in a bit...

Rule Order

- If there are conflicts between style rules of identical weight, later rules override earlier ones:

```
p { color: red; }
p { color: blue; }
p { color: green; }
/* green is last; it wins */
```

Assigning Importance

- Add the **`!important`** indicator just after the property value (before the semicolon) to prevent that rule from being overridden:

```
p { color: blue !important; }
```

- `!important` rules can *ONLY* be overridden by `!important` rules in a reader style sheet

The Box Model

- Browsers see every page element as being contained in a little rectangular box
  - we apply properties to and can reposition these boxes on the page
- Block element boxes expand to fill the window width
- Inline boxes only encompass their own text

Grouped Selectors

- We can combine style rules to apply the same properties to multiple elements
  - use commas to separate the selectors

```
h1, h2, p {border: 1px
                solid blue; }
```

Formatting Text

List of Topics

- Font-related properties
- Web fonts and font stacks
- Text line settings and text treatments
- Letter and word spacing
- Selector types and specificity
- Styles for lists

Font Properties

- `font-family`
- `font-size`
- `font-weight`
- `font-style`
- `font-variant`
- `font` (shortcut property)

Specifying a Font Name

- Use the font-family property to specify a font or list of fonts (font stack) by name
  - All font names (except generic families) must be capitalized
  - Use commas to separate multiple names
  - Names that include spaces must appear in quotation marks

```
body { font-family: Arial; }

var { font-family: Courier,
                   monospace; }

p { font-family: "Duru Sans",
         Verdana, sans-serif;}
```

Font Stacks

- Browsers can only display fonts they have access to
- If the browser can't find a specified font, it uses its default font instead
- Use a font stack to provide an ordered list of backup fonts

Web Fonts

- Can be self-hosted (in multiple formats for multiple browser types) or use an embedding service
- Use the `@font-face` rule in your stylesheet to define font information (including the URL) for later use with a `font-family` property

Generic Font Families

- Available in all browsers
- Types: serif, sans-serif, monospace, cursive, fantasy

Font Size

- Can be specified in several ways:
  - at a specific size using CSS length units
  - at a percentage value (e.g., 150%)
  - using an absolute keyword: xx-small, x-small, small, medium, large, x-large, xx-large
  - using a relative keyword: larger/smaller

Measurement Units

- *em* — relative unit of measurement (width of a capital 'M')
  - set body element to `font-size: 100%` and then use ems to resize text afterward (e.g., `1.5em`)

Font Size Keywords

- Default font size is medium
- xx-small, x-small, etc. are scaled in relation to each other (default size is 16 pixels)
- Can be imprecise and unpredictable across multiple browsers

Font Weight (Boldness)

- Can be a descriptive term or a numeric value:
  - normal, bold, bolder, lighter
  - 100, 200, 300, 400, 500, 600, 700, 800, 900
    - 600+ is usually bold text

Font Style and Variant

- `font-style` can be normal, italic, or oblique (slanted, not true italics)
- `font-variant` can be either normal or small-caps

The `font` Shortcut

- Use the font property to combine all of the previous properties into one rule:

```
{ font: style weight variant
   size/line-height font-family; }
```

- The property values MUST be supplied in this order!

`font` Shortcut Shortcuts

- At minimum, you need font size and family, in that order
  - other values are optional and may appear in any order prior to size

```
p { font: 1em sans-serif; }
```

Line Height

- If used, line height immediately follows font size and specifies the height of the text line
  - preceded/separated by a slash

    e.g., 1.5em/1.8em

Changing Text Color

- The value of the `color` property can be a predefined color name or an RGB value
  - Predefined colors: black, white, purple, lime, navy, aqua, silver, maroon, fuchsia, olive, blue, orange, gray, red, green, yellow, teal (plus more in CSS 3)
- `color` technically changes the foreground of an element (and its border)

More Selectors

- Descendant selectors: selects an element based on its relation to another element
  - spaced list of elements (e.g., `li em`)
- ID selectors: use # to target elements by `id` values (e.g., `#abc123`)
- Class selectors: use . to target by `class` values (e.g., `.special` or `p.special`)

Specificity (again)

- List of selector types (most specific to least specific):
  - ID selectors
  - Class selectors
  - Contextual selectors (e.g., descendants)
  - Individual element selectors

Text Line Adjustments

- `line-height`: can be a number (multiplier), a percentage, or a length (em) measurement
- `text-indent`: indents the first line of a block (can also create *hanging indents*)
- `text-align`: horizontal block alignment: `left, right, center, justify`

Decoration and Transformation

- `text-decoration`: affects lines around text (like link underlines)
  - `none, underline, overline, line-through`
- `text-transform`: affects capitalization
  - `none, capitalize, lowercase, uppercase`

Text Spacing

- `letter-spacing`
- `word-spacing`

Text Shadows

- `text-shadow` takes three or four values:
  - horizontal offset
  - vertical offset
  - blur radius (optional, can be 0+)
  - shadow color
- Can apply multiple shadows at once

Changing List Bullets

- `list-style-type`: selects the type of marker before each list item
  - none, disc, circle, square, decimal, decimal-leading-zero, lower-alpha, upper-alpha, lower-roman, upper-roman, lower-greek
- `list-style-position`: inside or outside the content area

Your Own List Bullets

- Use list-style-image to specify a URL for your preferred image bullet

```
list-style-image:
    url(/images/rainbow.gif);
```

- In case the image doesn't display, the `list-style-type` is set to disc by default