CSS Cont'd: Cascading Style Sheets

Lecture 15
Resolving Conflicts

- Generally, (text) styles are inherited
- Overridden by selectors that match children
- But conflicts can still arise when multiple selectors match an element
  - Multiple rules with same selector
  - Element part of 2 different classes
  - Two different paths (ancestors) match
  - Different sources of css (author vs user)
Priority of Styling

- Rough sketch:
  - Divide rules into categories
  - Within category, most specific rule wins
  - Break ties with order of rule declaration

- More detail, there are 4 factors:
  1. Location
  2. Importance
  3. Specificity
  4. Declaration order
Location

☐ Three sources of CSS rules:
  ■ Author of document
    ☐ Direct style attribute on element (ugly)
    ☐ <style> in head element
    ☐ <link> to CSS style sheets in header
  ■ User (eg userContent.css for FF)
  ■ Browser (defaults, eg blue underline)

☐ Priority order (decreasing):
  1. Author (direct, head style, linked)
  2. User
  3. Browser
Importance

- Preference given to document author
- But some users *really* need control
- Solution: !important modifier
  ```css
  h1 {font-family: arial !important;}
  ```
- Priority order (decreasing):
  1. User important
  2. Author important
  3. Author (normal)
  4. User (normal)
  5. Browser (normal)
- Use with caution! (eg for debugging)
Specificity

- Within a given category, *most specific* rule has highest priority
- Specificity of selector: a triple \((x, y, z)\)
  - \(X = \) no. of id's
  - \(Y = \) no. of classes (and pseudo-classes)
  - \(Z = \) no. of elements (and pseudo-elts)
- Compare specificity lexicographically
- Larger value = more specific = higher priority
Source Order

- Remaining ties broken by the order in which rules are encountered
- Later rule overrides previous one
- Example: order matters!
  
  h1, h2 {padding: 25px;}
  h2 {padding-left: 10px;}

- Example: order matters!
  
  p {
    padding: 25px;
    padding-left: 80px;
  }
Your Turn

- Which rule has higher priority?
  
  ```
  #main li {}
  .draft ul li {}
  ```

- Order the following from high to low:
  
  ```
  .draft div .warning li {}
  .draft div #main li { !important; }
  .draft div #main ul li {}
  .draft .warning ul li {}
  ```
Problem: Selectors Beat Inherit.
Explicit Inheritance

- Problem: How to style `<a>`?
  - Default: `<a>` inherits color (good)
  - Browsers have default color for `<a>` (bad)
  - Could override this with author styling
    ```css
    a {color: black;}
    ```
  - But I want the color dictated by styling of *parent* of `<a>`
    ```css
    body {color: darkred;}
    ```

- Solution: explicit inheritance
  ```css
  a {color: inherit}
  ```
Pseudo-classes

- Virtual classes
  - Implicitly declared (a few standard ones)
  - Implicit membership (no class attribute)
- CSS syntax: \texttt{elt:pseudo}
  \[
  \text{ul li:nth-child(odd) \{...\}}
  \]
Some Useful Pseudo-classes

- **Classic**
  - `:link`, `:visited`, `:active`
  - `:hover`, `:focus`
- **Structural**
  - `:nth-child(N)`, `:nth-of-type(N)`
  - `:first-child`, `:last-child`, `:only-child`, `:only-of-type`
  - `:empty`, `:root`
- **State of UI elements**
  - `:enabled`, `:disabled`
  - `:checked`
- **Target**
  - `:target`
- **Negation**
  - `:not(S)`
Pseudo-elements

- Virtual elements
  - Implicitly exist
  - Not part of structural tree (just rendering)
- CSS syntax: \texttt{elt::pseudo}
  
  \begin{verbatim}
  .summary th::after {content: "!";}
  \end{verbatim}
Some Useful Pseudo-Elements

- Match start
  - ::first-line, ::last-line

- Insert content
  - ::before, ::after
  - Inserted as (first/last) child of element
  - Requires content property
  - Beware using CSS to inject content!
Summary

- Classes and Ids
- Divs and Spans
- Selectors with ancestors, siblings
- Conflict resolution in CSS
- Pseudo-classes and pseudo-elements