CSS Cont'd:
Cascading Style Sheets

Lecture 15
Resolving Conflicts

- Generally, (text) styles are inherited
- Inherited styles are overridden by selectors that match children
- But a conflicts can arise: multiple selectors match the same 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:
  - Place conflicting rules into categories
  - Within category, most specific rule wins
  - Break remaining ties with order of declaration

- More detail: There are 3 stages, made from 4 factors:
  1. Location and Importance
  2. Specificity
  3. Declaration order
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 (e.g., `userContent.css` for FF)
- Browser (defaults, e.g. 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 of categories:
  1. User important
  2. Author important
  3. Author (normal)
  4. User (normal)
  5. Browser (normal)
- Use with caution! (e.g. 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 { }
Your Turn

Which rule has higher priority?

#main li {}
.draft ul li {}

Order the following from high to low:

4) .draft div .warning li {}
1) .draft div #main li { !important; }
2) .draft div #main ul li {}
3) .draft .warning ul li {}
Problem: Selectors Beat Inherit.
Explicit Inheritance

- Problem: How to style `<a>`?
  - Children inherit color from parent (good)
  - But browser defines default color for `<a>`
    ```css
    a {color: blue;
       text-decoration: underline;}
    ```
  - Author styling overrides browser rule
    ```css
    a {color: black;}
    ```
  - But I want the color dictated by styling of `parent` of `<a>`
    ```css
    .warning {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: `elt:pseudo`
  - Same specificity as (non-pseudo) class
    ```
    ul li:nth-child(2n) {...}
    ```
Some Useful Pseudo-classes

- **Classic**
  - :link, :visited, :active
  - :hover, :focus

- **Structural**
  - :nth-child(N), :nth-of-type(N)
  - :first-child, :last-child, :first-of-type
  - :only-child, :only-of-type
  - :empty, :root

- **State of UI elements**
  - :enabled, :disabled
  - :checked

- **Target**
  - :target /*elt whose id matches url fragment*/

- **Negation**
  - :not(S)
a.button:hover {
    background: green;
}

tbody tr:nth-of-type(odd) {
    background: #ccc;
}
Pseudo-elements

- Virtual elements
  - Implicitly exist
  - Not part of structural tree (just rendering)
- CSS syntax: \texttt{elt::pseudo}

```css
.summary th::after {content: "!";}
```

![Diagram showing the structure with pseudo-elements]
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