CS1115/CS5002 Web Development 1

Dr Derek Bridge

School of Computer Science & Information Technology

University College Cork

Conflicts

- Often the same element may be selected by more than one rule
- It's not a problem in this case:

```
h1  {
    color: red;
}

h1 {
    font-style: italic;
}
```

- But it is a problem in this case:

```
h1 {
    color: red;
}

h1 {
    color: blue;
}
```

Conflicts

- The problem is often less obvious, e.g.

```
#instructions {
    color: blue;
}

section {
    color: green;
}
```

in the Mojitos web page

Resolving conflicts: importance and origin

- CSS has a well-defined way of deciding which of several conflicting rules will 'win'
- A full treatment of this would begin by discussing rule **importance** and rule **origin**
- We'll look at these two briefly — but you don't need to learn these two

Resolving conflicts: importance

- You can declare a property within a rule to be important:

```
h1 {
    color: blue !important;
}
```

- If two rules conflict, the `!important` one wins
- Using this is generally a horrible hack — please do **not** use
- But suppose both or neither are `!important`…

Resolving conflicts: origin

- Stylesheets have different **origins**
  1. author stylesheets
  2. user stylesheets (if your browser allows them)
  3. the browser's default stylesheet
- If two rules conflict, the following is the order of precedence:
  1. `!important` rules from user stylesheets
  2. `!important` rules from author stylesheets
  3. other rules from author stylesheets
  4. other rules from user stylesheets
  5. the browser's default stylesheet

Resolving conflicts: specificity

- Some selectors are more **specific** than others
- Roughly(!),
  - Selecting by id is the most specific, then selecting by class
  - Then, a selector with more children/descendants /siblings is more specific

  (See W3C's Specification for a precise definition)
- If two rules conflict, the more specific wins

Resolving conflicts: specificity

Class exercise: Which is the more specific in each of these pairs of rules

```
i {
    color: blue;
}

.cocktail
{
    color: green;
}
```

```
nav li {
    color: yellow;
}

li {
    color: gray;
}
```

```
#instructions b {
    color: teal;
}

ol li b {
    color: purple;
}
```

Resolving conflicts: ordering

- But suppose both rules have the same specificity...
- In that case, the **order** in which rules appear is relevant:

```
h1 {
    color: red;
}

h1 {
    color: blue;
}
```

- If two rules conflict, the later rule wins

Resolving conflicts: the cascade

- Note the way that the conflict resolution **cascades**:
  1. importance/origin
  2. specificity
  3. order

  and when rules select an element, they beat inheritance

Class exercise

In the *Instructions* section of the Mojitos web page, what colour will the word "crush" be and what colour will the rest of that list item be?

```
<section id="instructions">

   <h1>Instructions</h1>

   <p>
      For the best <i class="cocktail" lang="es">Mojitos</i> this side
      of Waterford,
   </p>
   <ol>
      <li>Gently <b>crush</b> the mint leaves in a cool tall glass;</li>
      <li><b>squeeze</b> the lime over the leaves;</li>
      <li><b>sprinkle</b> the sugar into the mix;</li>
      <li><b>fill</b> with ice;</li>
      <li><b>toss in</b> the rum;</li>
      <li><b>trickle in</b> in the club soda; and finally</li>
      <li><b>stir</b> lightly.</li>
   </ol>

</section>
```

```
#instructions {
    color: green;
}

li > b {
    color: red;
}

section b {
    color: blue;
}

section {
    color: yellow;
}
```

Conflicts are not irrelevant, just because they're difficult!

- A stylesheet where there is blocking of inheritance or conflicts is not necessarily a bad stylesheet
- CSS experts *exploit* blocking of inheritance and conflicts to get a more elegant stylesheet
- E.g. in the Mojitos navigation menu, suppose we want *Cocktails*, *Finger Food* and *Party Games* to be red, but *Cuba Libre*, *Mojitos*, etc. to be blue
- The left-hand example uses rules that don't conflict whereas the right-hand example uses rules that do conflict:

```
nav > ul > li {
    color: red;
}

nav li li {
    color: blue;
}
```

```
nav li {
    color: red;
}

nav li li {
    color: blue;
}
```