COMP519 Web Programming
Lecture 6: Cascading Style Sheets: Part 2
Handouts

Ullrich Hustadt

Department of Computer Science
School of Electrical Engineering, Electronics, and Computer Science
University of Liverpool
Contents

1 Document Style Sheets
   Overview
   Selectors: Basic Forms
   Inheritance
   Specificity
   Multiple Classes
   Relative Selectors

2 Further Reading
Document Style Sheets

- **Inline styles** apply to individual elements in an HTML document.
- Using **inline styles** can lead to **inconsistencies**, as similar elements might end up formatted differently as this approach is error-prone.
- Also, **inline styles** mix content and presentation, which violates the (current) general philosophy of HTML and CSS.
- As a general rule, **inline styles** should be used as sparingly as possible.
- Alternatively, **document style sheets** allow for a cleaner separation of content and presentation.
  - Style definitions are placed in a **style-element** within the **head-element** of an HTML document.
  - The **style-element** contains a list of **style directives**.
  - Each **style directive** consists of a **selector** together with one or more **property:value** pairs separated by semi-colons and enclosed inside braces.
  - Selectors allow us to apply style definitions to all elements, a subclass of elements, or individual elements throughout the document.
The h1 selector means that the style will be applied to all h1-elements

The p.indented selector means that the style will only be applied to p-elements of class indented

Centred Red Heading

This paragraph will have the first line indented, but subsequent lines will be left-aligned.

This paragraph will not be indented, all lines are left-aligned as per default.
Selectors: Basic Forms

- **Selectors** allow us to apply style definitions to all elements, a subclass of elements, or individual elements throughout the document.

- The basic forms of selectors are:

<table>
<thead>
<tr>
<th>Selector</th>
<th>Example</th>
<th>Selects</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>*</td>
<td>All elements</td>
</tr>
<tr>
<td>element</td>
<td>h1</td>
<td>All h1-elements</td>
</tr>
<tr>
<td>element, element</td>
<td>h1, h2</td>
<td>All h1- and h2-elements</td>
</tr>
<tr>
<td>.class</td>
<td>.indented</td>
<td>All elements with class=&quot;indented&quot;</td>
</tr>
<tr>
<td>element.class</td>
<td>p.indented</td>
<td>All p-elements with class=&quot;indented&quot;</td>
</tr>
<tr>
<td>#id</td>
<td>#name</td>
<td>The element with id=&quot;name&quot;</td>
</tr>
</tbody>
</table>
Basic Selectors: Example

```css
/* Make headings at levels h1 to h3 blue, centred with an italic font */

h1, h2, h3 { color: Blue;
            text-align: center;
            font-style: italic; }

/* Remove underline from hyperlinks and make them red */
a { text-decoration: none;
   color: Red; }

/* Any element of class "alert" should be underlined, have a bold font, orange background, and 3px of extra bottom padding to make the underlining more visible */

.alert { text-decoration: underline;
         padding-bottom: 3px;
         background-color: orange;
         font-weight: bold; }

/* Use a font of 150% normal size for the element with id "c1" */
#c1 { font-size: 150%; }
```

Selectors, Classes and Inheritance

The style directive

```css
.alert { text-decoration: underline; padding-bottom: 10px;
background-color: orange; font-weight: bold; }
```

creates a style for the class “alert” that can (in principle) be applied to any HTML element by associating it with that class.

- Applied to a p-element

```html
<p class="alert">Help Me! I’m trapped on a COMP519 slide!</p>
```

we obtain the following:

```
Help Me! I’m trapped on a COMP519 slide!
```
Selectors, Classes and Inheritance

The style directive

```css
.alert { text-decoration: underline; padding-bottom: 10px;
background-color: orange; font-weight: bold; }
```

creates a style for the class “alert” that can (in principle) be applied to any HTML element by associating it with that class.

• Applied to an li-element

```html
<ol>
  <li class="alert">Help Me!</li>
  <li>I’m trapped on a COMP519 slide!</li>
</ol>
```

we obtain the following:

1. **Help Me!**
2. I’m trapped on a COMP519 slide!

• As one might expect, the first list item appears as specified by the style while the second item is unaffected
• Note the vertical space between the first and second item due to padding-bottom
Selectors, Classes, and Inheritance

The style directive

```css
.alert { text-decoration: underline; padding-bottom: 10px; background-color: orange; font-weight: bold; }
```

creates a style for the class “alert” that can (in principle) be applied to any HTML element by associating it with that class.

- **Applied to an ol-element**

```html
<ol class="alert">
  <li>Help Me!</li>
  <li>I’m trapped on a COMP519 slide!</li>
</ol>
```

we obtain the following:

1. **Help Me!**
2. **I’m trapped on a COMP519 slide!**

in contrast to:

1. **Help Me!**
2. I’m trapped on a COMP519 slide!

- Note the extra vertical space between the two list items is gone while there is extra padding below the list
- The li-elements have inherited text decoration, font weight and background colour from the ol-element, but not padding-bottom
Inheritance

- In CSS, inheritance controls what happens when no value is explicitly specified for a property of an element.
- CSS distinguishes between inherited properties and non-inherited properties.
- When no value for an inherited property has been specified for an element, it inherits the value of that property from its parent element.

Example:

```html
<ol class="alert">
  <li>Help Me!</li>
  <li>I’m trapped on a COMP519 slide!</li>
</ol>
```

Here, the `ol`-element is the parent element of both `li`-elements.

- The inheritance chain potentially goes up to the root element.
  - The root element is the `html`-element of a document.
  - The root element is given some initial, default value for each property.
Inheritance

- In CSS, inheritance controls what happens when no value is explicitly specified for a property of an element.
- CSS distinguishes between inherited properties and non-inherited properties.
- When no value for a non-inherited property has been specified for an element, the property is given the initial, default value for that property.
Inheritance

• CSS distinguishes between inherited properties and non-inherited properties

• When no value for an inherited property has been specified for an element, it inherits the value of that property from its parent element

• When no value for a non-inherited property has been specified for an element, the property is given the initial, default value for that property

Example:

color is an inherited property,
padding-bottom is a non-inherited property

→ If color and padding-bottom are not specified for an li-element, then color is inherited from the parent ol-element (or other list construct) while padding-bottom gets the initial value 0
Inheritance

Consider the style directive

```
table { background-color: yellow; 
       border: 1px solid black; padding: 10px; margin: 10px; }
```

together with

```
<table>
  <tr><td>Cell</td><td>Cell</td></tr>
  <tr><td>Cell</td><td>Cell</td></tr>
</table>
```

that will be rendered as

```
Cell Cell
Cell Cell
```

- **background-color** will be inherited by the table cells (td-elements) from the table element
- **border** and **padding** are **not** inherited by the table cells (td-elements) from the table element
  - table cells have no individual borders and no padding
- **border** and **padding** only apply to the table as a whole
We can change what properties are inherited as follows:

```html
table { background-color: yellow;
        border: 1px solid black; padding: 10px; margin: 10px; }
tbody, tr, td { border: inherit; padding: inherit; }
```

```html
<table>
  <tr><td>Cell</td><td>Cell</td></tr>
  <tr><td>Cell</td><td>Cell</td></tr>
</table>
```

- **border** and **padding** are now inherited by the table cells (td elements) from the table element
- table cells now have individual borders and extra padding
- Note that tbody and tr not just td must be set to inherit
- there must be an uninterrupted chain of inheritance from table to td
Debugging CSS

Google Chrome and Mozilla Firefox allow you to inspect individual HTML elements and can tell you how its properties come about.
Specificity

- Several kinds of style directives can apply to the same HTML element
- It is also possible that these style directives assign different values to the same property

**Example:** Consider the style directives

```html
h1 { background-color: yellow; color: blue; }
.orange { background-color: orange; text-decoration: underline; }
#i1 { background-color: red; font-style: italic; }
```

together with

```html
<h1 id="i1" class="orange">What properties does this heading have?</h1>
```

that will be rendered as

*What properties does this heading have?*

- The selector `#i1` is the *most specific* one and therefore determines `background-color`
- For other properties, the heading inherits from all three directives
Specificity

- Several kinds of style directives can apply to the same HTML element
- It is also possible that these style directives assign different values to the same property
- For such ‘conflicting’ assignments, CSS uses specificity to determine which assignment applies to an element:
Multiple Classes

- It is possible to declare that an HTML element belongs to more than one class:

```html
<tagName class="class1 class2...">
```

Example: Consider the style directives

```css
p.alert { text-decoration: underline; padding-bottom: 10px;
          background-color: orange; font-weight: bold; }
p.blue  { color: blue; }
```

together with the HTML markup

```html
<p class="alert blue">What colours do I have?</p>
```

that will be rendered as

What colours do I have?

The properties of the paragraph come from both `p.alert` and `p.blue`
Multiple Classes

Consider the style directives

\[
\text{.red1 \{ color: red; background-color: yellow; margin: 5px; \}}
\]
\[
\text{.blue \{ color: blue; \}}
\]
\[
\text{.red2 \{ color: red; background-color: yellow; margin: 5px; \}}
\]

together with the HTML markup

\[
\text{<p class="red1 blue"> class="red1 blue", blue after red1 in style sheet</p>}
\]
\[
\text{<p class="blue red1"> class="blue red1", blue after red1 in style sheet</p>}
\]
\[
\text{<p class="red2 blue"> class="red2 blue", red2 after blue in style sheet</p>}
\]
\[
\text{<p class="blue red2"> class="blue red2", red2 after blue in style sheet</p>}
\]

that will be rendered as

\[
\text{class="red1 blue", blue after red1 in style sheet}
\]
\[
\text{class="blue red1", blue after red1 in style sheet}
\]
\[
\text{class="red2 blue", red2 after blue in style sheet}
\]
\[
\text{class="blue red2", red2 after blue in style sheet}
\]

• For two class style directives with conflicting assignments that apply to
  the same element, the assignment that comes last in the style sheet is
  applied

  \(\sim\) problematic, as the style sheet might change
Multiple Classes: Resolving ‘conflicts’

• ‘Conflicts’ between multiple classes can be resolved in advance by defining a directive that specifically applies to their combination.

Consider the style directives

```css
.red1.blue { color: white; background-color: orange; }
.red1 { color: red; background-color: yellow; margin: 5px; }
.blue { color: blue; margin: 5px; }
.red2 { color: red; background-color: yellow; margin: 5px; }
```

together with the HTML markup

```html
<p class="red1 blue">class="red1 blue", blue after red1 in style sheet</p>
<p class="blue red1">class="blue red1", blue after red1 in style sheet</p>
<p class="red2 blue">class="red2 blue", red2 after blue in style sheet</p>
<p class="blue red2">class="blue red2", red2 after blue in style sheet</p>
```

that will be rendered as

```
class="red1 blue", blue after red1 in style sheet
class="blue red1", blue after red1 in style sheet
class="red2 blue", red2 after blue in style sheet
class="blue red2", red2 after blue in style sheet
```

• .red1.blue has higher specificity than both .red1 and .blue

• margin is still determined by .blue
Relative Selectors

- It is possible to specify selectors that take into account the context in which a matching HTML element occurs:

<table>
<thead>
<tr>
<th>Selector</th>
<th>Example</th>
<th>Selects</th>
</tr>
</thead>
<tbody>
<tr>
<td>element element</td>
<td>ol a</td>
<td>All a elements inside ol elements</td>
</tr>
<tr>
<td>element&gt;element</td>
<td>ol&gt;li</td>
<td>All li elements where the parent is an ol element</td>
</tr>
<tr>
<td>element+element</td>
<td>p+ol</td>
<td>All ol elements placed immediately after a p element</td>
</tr>
<tr>
<td>element~element</td>
<td>p~ol</td>
<td>All ol elements preceded by a p element</td>
</tr>
</tbody>
</table>
**Relative Selectors: Example**

Consider the style directives

```css
ol a { color: red; text-decoration: none } /* Hyperlink in ol */
ol>li { font-style: italic; } /* li children of ol */
p+ol { font-weight: bold; } /* ol immediately after p */
p~ol { background-color: yellow; } /* ol after p */
```

together with the HTML markup

```html
<p>A paragraph with a <a href="/~ullrich/">hyperlink</a></p>
<ol>
  <li>A list item in an ordered list with 
      <a href="/~ullrich/">hyperlink</a></li>
</ol>
<ul>
  <li>A list item in an unsorted list with 
      <a href="/~ullrich/">hyperlink</a></li>
</ul>
<ol>
  <li>A list item in an ordered list with 
      <a href="/~ullrich/">hyperlink</a></li>
</ol>
```

that will be rendered as

A paragraph with a hyperlink

1. A list item in an ordered list with a hyperlink

   • A list item in an unsorted list with a hyperlink

1. A list item in an ordered list with a hyperlink
Why Do We Need All These Selectors?

- All ‘styling’ could be done by #id selectors, but should not be (just assign a unique id to every element and define its style)
  - just emulating inline style directives
  - enormous duplication of style directives
  - error-prone (difficult to get uniform style)

- All ‘styling’ could be done by .class selectors, but should not be (just assign a (unique) class to every element and define its style)
  - in extremis just emulating inline style directives
  - classes should be based on semantics not style (e.g., shopping lists versus shopping baskets versus playlist)
  - still error-prone (you could miss an element or assign the wrong class)

If possible we should use existing properties of elements to determine what style to apply, e.g., relative selectors and attribute selectors
Revision and Further Reading

Read about Selectors and Inheritance in

- Chapter 11: Introducing Cascading Style Sheets
- Chapter 12: Formatting Text
- Chapter 13: Colors and Backgrounds

of

E-book https://library.liv.ac.uk/record=b5647021