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

Document Style Sheets

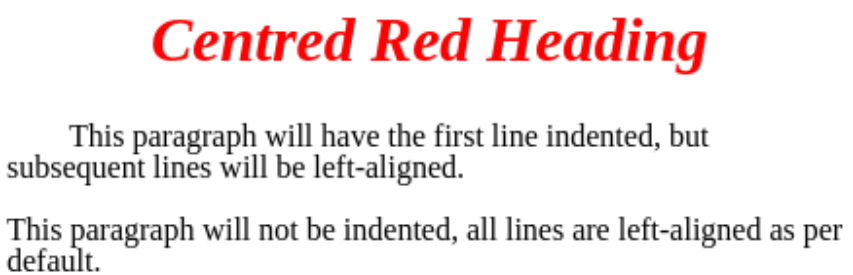
- Inline styles apply to individual elements in an HTML document
- Using inline styles can lead to inconsistencies, as similar elements might end up formatted differently as this approach is error-prone
- Also, inline styles mix content and presentation, which violates the (current) general philosophy 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 ore 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

Document Style Sheets: Example

```
<html lang="en-GB">
 <head>
  <title>Document Style Sheets</title>
  <style type="text/css">
   h1 { color: Red;
        text-align: center;
        font-style: italic; }
   p.indented {  text-indent: 2em; }
  </style>
 </head>
 <body>
  <h1>Centred Red Heading</h1>
  <p class="indented">This paragraph
    will have the first line indented,
    but subsequent lines will be
    left-aligned.
  </p>
  <p>This paragraph will not be
     indented, all lines are
     left-aligned as per default.
  </p>
 </body>
</html>
```

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`



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:

Selector	Example	Selects
*	*	All elements
element	h1	All h1-elements
element, *element*	h1, h2	All h1- and h2-elements
.*class*	.indented	All elements with class="indented"
element.*class*	p.indented	All p-elements with class="indented"
#*id*	#name	The element with id="name"

Basic Selectors: Example

```
<style type="text/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%; }
</style>
```

Selectors, Classes and Inheritance

The style directive

```
.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

  ```
  <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

```
.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

```
<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

```
.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

```
<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:

```
<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

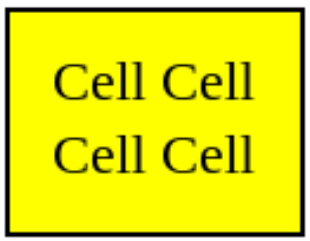
Consider the style directive

```
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



- `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
  $\rightsquigarrow$ table cells have no individual borders and no padding
- `border` and `padding` only apply to the table as a whole

Inheritance

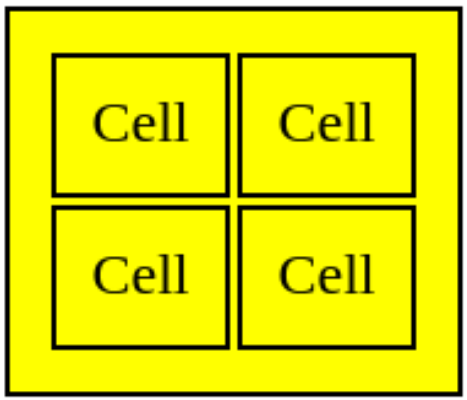
We can change what properties are inherited as follows:

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

together with

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

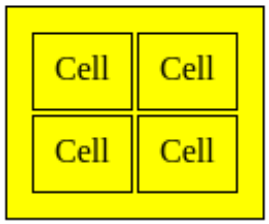
that will now be rendered as

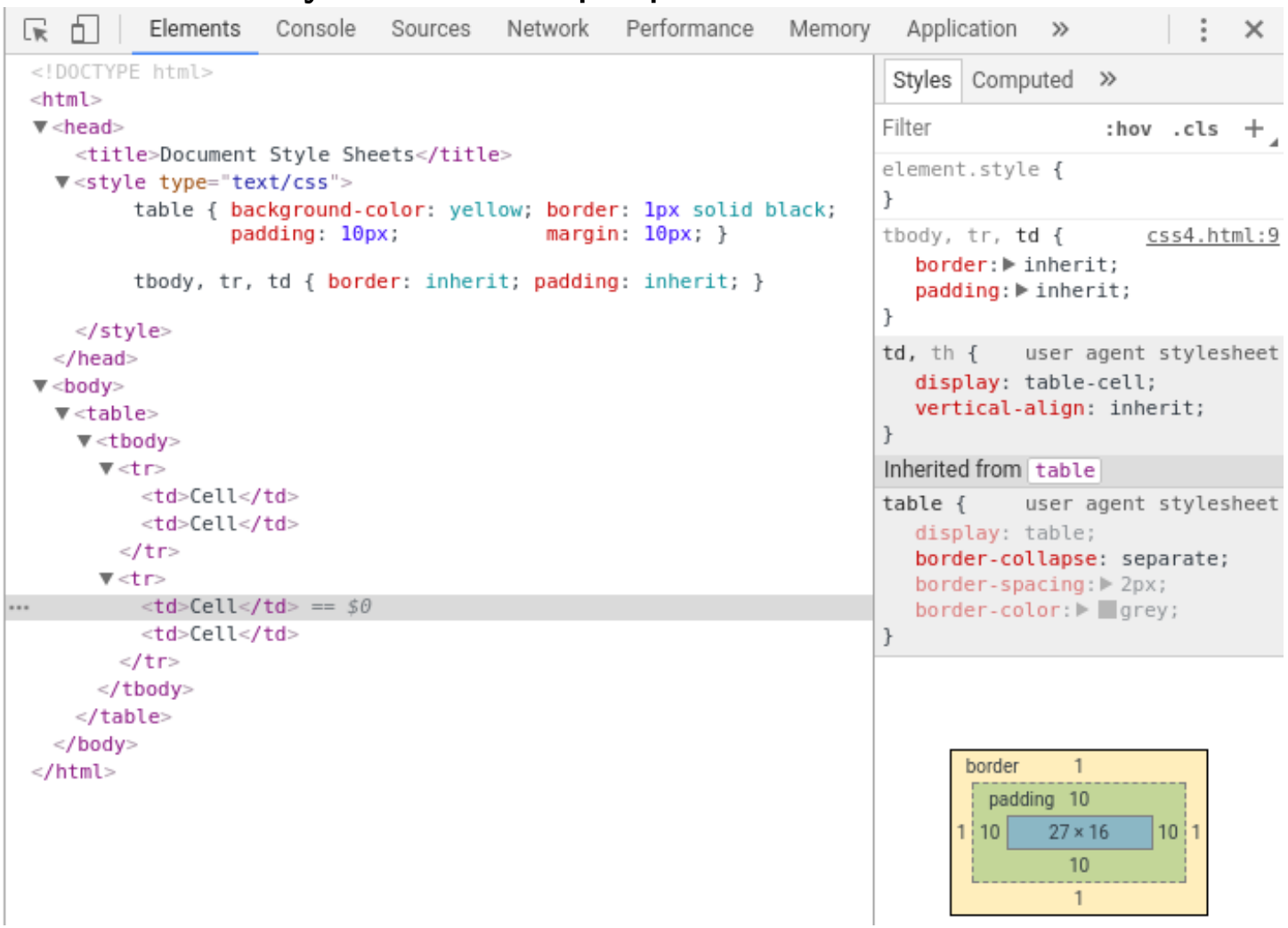


- `border` and `padding` are now inherited by the table cells (`td` elements) from the table element
  - ⇝ table cells now have individual borders and extra padding
- Note that `tbody` and `tr` not just `td` must be set to inherit
  - ⇝ there must be an uninterrupted chain of inheritance from `table`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

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

together with

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

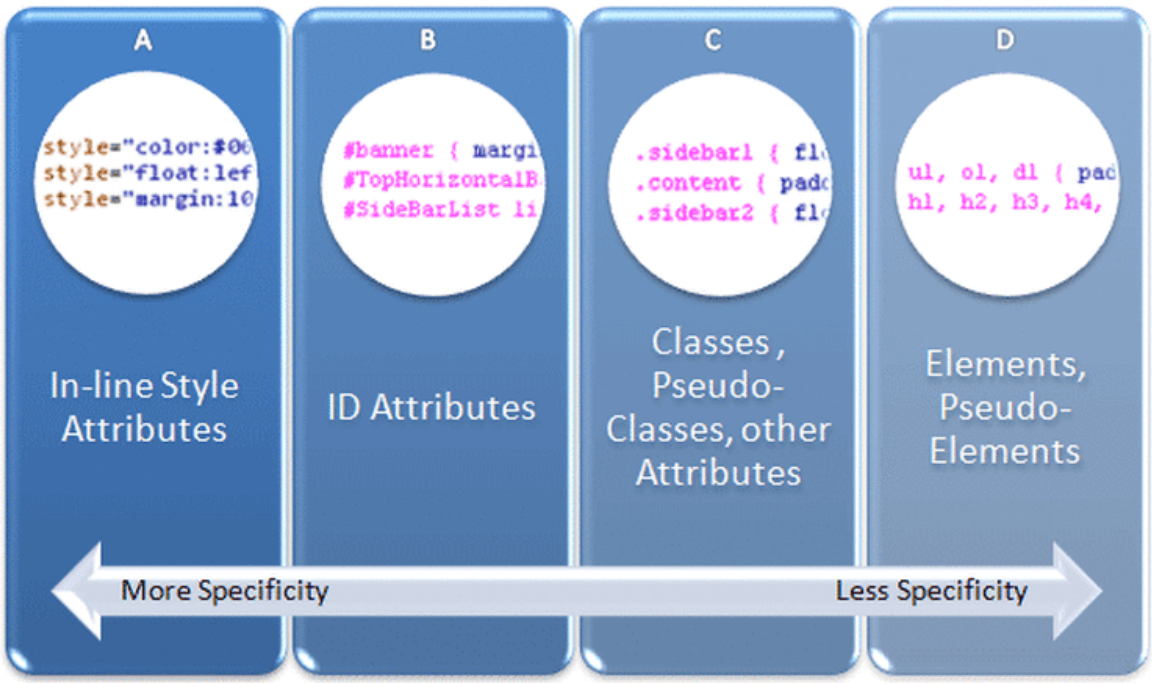
that will be rendered as

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:

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

Example: Consider the style directives

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

together with the HTML markup

```
<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 from `p.blue`

Multiple Classes

Consider the style directives

```
.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

```
<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

- 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
  $\rightsquigarrow$ 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

```
.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

```
<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:

Selector	Example	Selects
element element	`ol a`	All `a` elements inside `ol` elements
element>*element*	`ol>li`	All `li` elements where the parent is an `ol` element
element+*element*	`p+ol`	All `ol` elements placed immediately after a `p` element
element~*element*	`p~ol`	All `ol` elements preceded by a `p` element

Relative Selectors: Example

Consider the style directives

```
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

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

J. Niederst Robbins: Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics (5th ed).
O'Reilly, 2018.
E-book `https://library.liv.ac.uk/record=b5647021`