

CM0133 Internet Computing

4. CSS , Stylesheets

Objectives

- Learn the motivation of stylesheets
- Appreciate the advantages of using stylesheets vs representational html elements
- Learn by examples to style a simple web page

Content and Presentation

- Cascading Stylesheets (CSS) have been designed to better **separate the content and presentation** of a web document (coded in HTML).
- CSS uses a model of **selectors** and **declarations** to achieve that.
- Selectors specify to which elements of a document a set of declarations apply.
- Additionally there is a model of how property values are inherited and cascaded.
- The biggest limitation of CSS is that it cannot change the structure of the displayed document.

Representational Aspects

- Aspects of the document presentation include
 - positioning on the page
 - choice of fonts
 - colours and backgrounds
 - Borders
- These have been controlled by HTML attributes (align, color, bgcolor)
- As a consequence HTML became “polluted” by layout information.

CSS

- CSS was introduced as a format for layout information
 - HTML can be kept simple, containing only the structures
 - Layout information can be reused by using separate CSS files
- CSS has been designed for HTML
 - It has been generalized to cover XML
 - HTML heritage is visible in CSS

Styles

- A **style** is a set of formatting instructions that can be applied to a piece of text.
- Styles can be defined
 1. Within a single HTML tag – **Inline styles (redundant inside documents)**
 2. In the **<head>** section, and applied to the whole document – **Global styles (redundant across documents)**
 3. In external files, and can be applied to any document by including the URI of the file – **Stylesheets (best reuse)**
 4. **Any combination is possible**

Inline styles

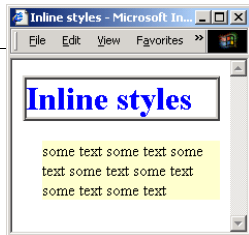
- Every tag has a **style attribute**
- This can be assigned a **style definition**
- A style definition is a list of **property-value** pairs
 - a property is separated from its value by a colon
 - property-value pairs are separated by semi-colons
 - the list is delimited by quotation marks
- Inline styles only affect the text contained in the tag

```
<h1 style="color:#2255ff; border:ridge">Inline styles</h1>
```

Inline styles

```
<body>  
  <h1 style="color:blue; border:ridge">  
    Inline styles</h1>  
  <p style="margin-left:10%; background:#ffffcc">  
    some text . . . some text  
  </p>  
</body>
```

- The heading is boxed with the text displayed in blue
- The paragraph is indented by 10% (from the left) and has a cream background



Global styles

- A style can be defined in the head of the document using the **<style>** tag
- The style declaration is placed inside a **comment** so that it can be ignored by older browsers
- Each style rule consists of the name of an element (**selector**) followed by a list of **property-value pairs** enclosed in curly brackets

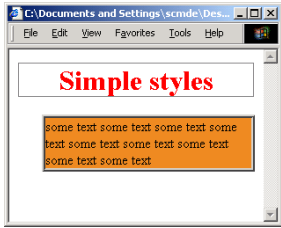
```
<head>  
<title>Styles</title>  
<style>  
<!--  
  h1 {  
    color: red;  
    border: thin groove;  
    text-align:center;  
  }  
-->  
</style>  
</head>  
<body>  
<h1>Simple styles</h1>  
</body>
```

```

<head>
<style>
<!--
  h1 {
    color: red;
    border: thin groove;
    text-align:center;
  }
  p {
    margin-left: 10%;
    border: ridge;
    background: #ee8822;
  }
-->
</style>
</head>
<body>
<h1>Simple styles</h1>
<p>some text . . . </p>
</body>

```

Example



- A global style applies to every instance of the corresponding element in the document

Stylesheets

- Styles can be declared in separate files called [stylesheets](#).
- A stylesheet is linked to a web document by including the following line in the head section

```
<link rel="StyleSheet" type="text/css" href="URL">
```

- [rel](#) specifies the type of link being used
- [href](#) specifies a hyperlink to the stylesheet file
- [type](#) specifies the MIME type of the data
- [text/css](#) describes the "cascading style sheets" type

Multiple stylesheets

- The first stylesheet is included using the [<link>](#) tag
- Any further stylesheets have to be [imported](#)
- The [@import](#) command is placed inside a comment

```

<head>
<title>Stylesheets</title>
<link rel="StyleSheet" type="text/css"
  href="http://www.abc.com/mainstyles.css">
<!--
  @import
  url(http://www.abc.com/deptstyles.css)
  url(mystyles.css)
-->
</head>

```

Cascading stylesheets

- Multiple stylesheets can be included in a document
- Styles defined in the first stylesheet are overridden by corresponding styles defined in the second stylesheet
 - the stylesheets are said to **cascade**
- Example
 - `mainstyles.css` – the company's stylesheet
 - `deptstyles.css` – the department's stylesheet
 - `mystyles.css` – the user's stylesheet
- If the stylesheets are included in this order, the user's style definitions will override the department styles, which in turn will override the company styles

Style rules

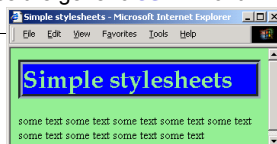
- A **style rule** has two parts
 - a selector (element name) and a set of declarations
- The **selector** associates the style rule with a HTML tag of the same name
- Each **declaration** has two parts:
 - a **property** and a **value**
- For readability, each declaration should be placed on a separate line

```
selector {  
  property: value;  
  property: value;  
  property: value;  
  property: value;  
}
```

Style rules

- Some properties can be given multiple values
 - The browser first looks for the "**Book Antiqua**" font
 - If this is not on the system, it looks for the **Times** font
 - Last resort: the browser uses the generic **serif** font

```
body {  
  background-color: lightgreen;  
}  
h1 {  
  color: lightgreen;  
  background-color: blue;  
  font-family: "Book Antiqua", Times, serif;  
  border: thick groove #9baab2;  
}
```



Properties

- Properties define how elements are formatted
 - They define a specific facet of formatting
 - They may have interdependencies with other properties
 - They can be assigned explicitly
 - They may be defined through **cascading** or **inheritance**
- A property has a name and is used in a name/value-pair
 - The name identifies the property that is being set
 - The value space depends on the property
 - Some properties accept complex values
 - Sets of values:
p { font-family: "Segoe UI", Verdana, Helvetica, Arial, Sans-Serif }
- Property specifications can be grouped
 - .thinboxed {border-width: 1px ; padding: 10 px ; margin : 5px }

Properties and values

Fonts

- **font-family**: <family name> [<generic family>]
- **font-style**: normal|italic|oblique
- **font-weight**: normal|bold|bolder|lighter
- **font-size**: small|medium|large|smaller|larger

Backgrounds and colours

- **color**: <value>
- **background-color**: <value>|transparent
- **background-image**: URL|none

Properties and values

Text

- **text-decoration**: none|underline|overline|line-through
- **text-transformation**: none|capitalize|uppercase|lowercase
- **text-align**: left|right|center|justify
- **text-indentation**: length|percentage

Properties and values

Boxes

- `margin-top`, `margin-right`, `margin-left`, `margin-bottom`: `value` (e.g. `5px`)
- `border-width`: `thin`|`thick`|`medium`|`length`
- `padding`: `length`|`percentage` (e.g. `.5em`)
- `border-color`: `red`|`blue`|`etc..`
- `border-style`: `none`|`dotted`|`dashed`|`solid`|`double`|`groove`|`ridge`
- `border-width`: `thick`|`medium`|`thin`

Properties and values

Position

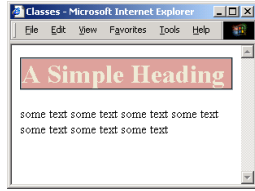
- `location`: `absolute`|`relative`|`fixed`
 - `absolute`: relative to upper left corner of window
 - `relative`: relative to the last item
 - `fixed`: does not move when the page is scrolled
- `left`: distance from left border of window (pixels, %)
- `top`: distance from top border of window (pixels, %)
- These properties are very useful with the *div tag* (see later)

Classes

- Simple style rules change the appearance of **all** instances of the associated element
- A **class** is a style definition that may be applied as and when we choose
 - if we don't want the styles, we don't have to use them
- **Simple classes** are applied to a single type of element
- **Anonymous classes** can be applied to any type of element

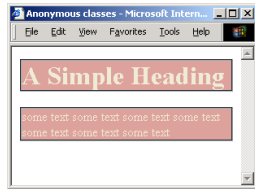
Simple classes

```
</head>
<style>
<!--
  h1.fred {
    color: #eeebd2;
    background-color: #d8a29b;
    border: thin groove #9baab2;
  }
-->
</style>
</head>
<body>
<h1 class="fred">A Simple Heading</h1>
<p>some text . . . some text</p>
</body>
```



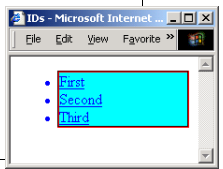
Anonymous classes

```
</head>
<style>
<!--
  .fred {
    color: #eeebd2;
    background-color: #d8a29b;
    border: thin groove #9baab2;
  }
-->
</style>
</head>
<body>
<h1 class="fred">A Simple Heading</h1>
<p class="fred">some text . . . some text</p>
</body>
```



IDs

```
<head>
<style>
<!--
  #list1 {
    color: blue;
    background: cyan;
    text-decoration: underline;
    border: thin groove red;
  }
-->
</style>
</head>
<body>
<ul id="list1">
<li>First</li>
<li>Second</li>
<li>Third</li>
</ul>
</body>
```



- Classes specify styles for particular instances of an element
- IDs specify the style of a single element
 - IDs allow the element to be identified by other elements on the page

Divisions and spans

- Rather than applying styles to an element itself, we wrap the element in
 - a **division** element (usually for block elements), or
 - a **span** element (usually for inline elements)
- Any required formatting can then be applied to the `<div>` or `` element.
- Division and span elements become part of the document
 - In particular, each can have `class` and `id` attributes

Divisions

```
<head>
<style>
<!--
.myclass {
  color: blue;
  background: cyan;
  text-decoration: underline;
  border: thin groove red;
}
-->
</style>
</head>
<body>
<div class="myclass">
  <h2>A Simple Heading</h2>
  <p>some text . . . </p>
</div>
</body>
```

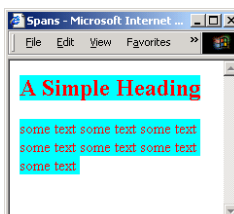
- Styles can be applied to blocks of HTML code using **divisions**



Spans

```
<head>
<style>
<!--
.myclass {
  color: red;
  background: cyan;
  text-decoration: none;
}
-->
</style>
</head>
<body>
<span class="myclass">
  <h2>A Simple Heading</h2>
  <p>some text . . . </p>
</span>
</body>
```

- **spans** are similar to divisions



Layers

- The browser maintains a stack of layers, each containing text, images etc. The browser displays layers on top of each other (in order).
- The `<div>` tag has the following attributes
 - **z-index**: the number of the layer containing the division
 - **left** and **top** : the location (top-left corner) of the division in pixels
 - **width** and **height**: the size of the division in pixels
 - **position**: **absolute** or **relative**

Layers

- Layers can be manipulated using JavaScript to create Dynamic HTML pages
- Layers can also be used to organise page content

BACKGROUND STUFF
THIS STUFF IS ON TOP

```
<body>
<div style="z-index:2; left:100px; top:50px; position:absolute;
background-color:red; font-size:30">
<p>THIS STUFF IS ON TOP</p>
</div>

<div style="z-index:1; left:10px; top:10px; position:absolute;
background-color:yellow; font-size:56">
<p>BACKGROUND STUFF</p></div>
</body>
```

Cascading - Mechanics

- Stylesheets may have three different origins
 1. Page authors associate CSS with their pages
 2. Users configure their browser to use some CSS
 3. User agents (browsers) have built-in CSS how to style content
- Conflicts must be resolved using the following algorithm
 1. Find all matching declarations (matching media type and selector)
 2. Sort according to importance (browser < user < author)
 3. Same importance must be sorted by specificity (more specific selectors)
 4. Finally, sort by order in which they were specified
- !important rules can influence the algorithm
 - They are interpreted in step2 (sorting by importance)
 - Browser < user < author < author (important) < user (important)

adapted from <http://dnet.net/lectures/web-spring09/2009-02-04-css.pdf>

Inheritance

- Properties are often inherited by children
 - Setting a table's color sets the color for all contents
 - Without inheritance, CSS would have to be very large
- Inheritance is mostly intuitive but the reality is a bit more complex: A **specified value** comes from the content model/style sheet or is a default value, computed is an absolutized or inherited version of the specified value, used is the **computed value** in light of the actual layout of the document and the **actual value** deals with limitations of the output device or media.

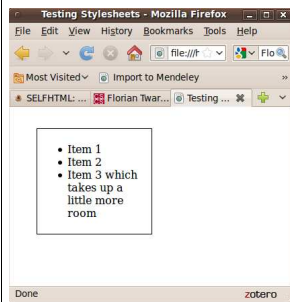
Strategies

- CSS should never show up in your HTML
- Classes should reflect content or formatting
- Containers can restrict styles to a context
- Context can be nested
 - Orthogonal concepts should be represented in nested classes
 - e.g., pages for "staff" and "faculty" and "current" and "past" as classification
 - Different levels of formatting sophistication can be implemented with CSS only
- Avoid redundant CSS code
 - Whenever appropriate, inherit properties
 - For invisible links use a {color : inherit ;}

Lists in CSS

```
#base {
  border: 1px solid #000;
  margin: 2em;
  width: 10em;
  padding: 5px;}

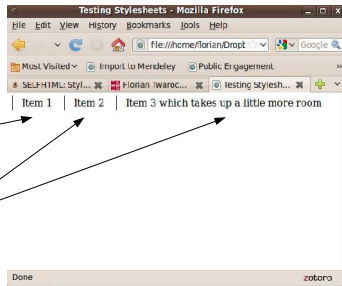
...
<div id="base">
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3 which takes up a
  little more room</li>
</ul>
</div>
...
```



Inline lists

- Lists are normally vertically orientated
- In CSS you can make them horizontal “inline”

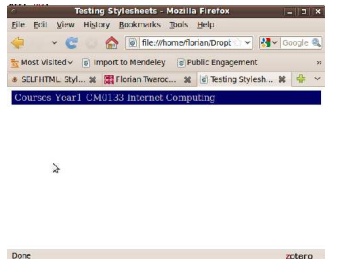
```
#pipe ul {
margin-left: 0;
padding-left: 0;
display: inline;
}
#pipe ul li.first {
margin-left: 0;
border-left: none;
list-style: none;
display: inline;
}
#pipe ul li {
margin-left: 0;
padding: 3px 15px;
border-left: 1px solid #000;
list-style: none;
display: inline;
}
```



Breadcrumb Trails

- Many sites now have an inline list showing the route you have navigated to end up at the current page
- It shows the hierarchy of the web site you followed

```
#bread {
color: #ccc;
background-color: #006;
padding: 3px;
margin-bottom: 25px;
}
#bread ul {
margin-left: 0;
padding-left: 0;
display: inline;
border: none;
}
#bread ul li {
margin-left: 0;
padding-left: 2px;
border: none;
list-style: none;
display: inline;
}
```



Inline hyperlinked lists

- These lists are ok, but for navigation you can make them hyperlinked lists

```
<div id="button">
<ul>
<li><a href="#">Home</a></li>
<li><a href="http://www.bbc.co.uk">BBC</a></li>
<li><a href="#">About Us</a></li>
</ul>
</div>
```

Set up the space for the menu

```
#button {
width: 12em;
border-right: 1px solid #000;
padding: 0 0 0 0;
margin-bottom: 1em;
font-family: 'Trebuchet MS', 'Lucida Grande', Geneva, Helvetica, Arial, sans-serif;
background-color: #90bade;
color: #333;
}
```

Make it look more like a menu!

- Remove the bullets and any borders from the list

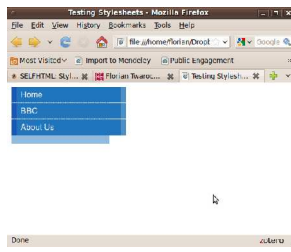
```
#button ul {
  list-style: none;
  margin: 0;
  padding: 0;
  border: none;
}
#button li {
  border-bottom: 1px solid #90bade;
  margin: 0;
}

#button li a {
  display: block;
  padding: 5px 5px 5px 0.5em;
  border-left: 10px solid #1958b7;
  border-right: 10px solid #508fc4;
  background-color: #2175bc;
  color: #fff;
  text-decoration: none;
  width: 100%;
}
```

Make all
hyperlinks
pretty!

The Resulting Menu

```
#button {
  width: 12em;
  border-right: 1px solid #000;
  padding: 0;
  margin: 0;
  border: none;
  background-color: #90bade;
  color: #333;
}
#button ul {
  list-style: none;
  margin: 0;
  padding: 0;
  border: none;
}
#button li {
  border-bottom: 1px solid #90bade;
  margin: 0;
}
#button li a {
  display: block;
  padding: 5px 5px 5px 0.5em;
  border-left: 10px solid #1958b7;
  border-right: 10px solid #508fc4;
  background-color: #2175bc;
  color: #fff;
  text-decoration: none;
  width: 100%;
}
```



CSS events

- We can use CSS code to catch events
- We can then assign CSS code to mouse events
 - CSS code allows you to access, images, layers, position, visibility etc
 - Use of these to create mouse over events for menus
- We can use CSS to make dynamic menus
- We can also use CSS to create all the buttons in our menus from just a text input
- This can save time when altering menus

Mouse Events

- We can catch the mouse over event from a hyperlink and use that to change our menu
- This will alter the colour of the block around the list item when the mouse is over it (the hyperlink)

```
#button li a:hover {  
  border-left: 10px solid #1c64d1;  
  border-right: 10px solid #5ba3e0;  
  background-color: #2586d7;  
  color: #fff;  
}
```

Why use CSS for this?

- CSS allows for neat and concise code to be written
- Any updates over multiple pages are easy to carry out
- Just alter the CSS code and all pages/tags calling that will alter
- Updating menus with new items, you just need to add a
- No need to go to photoshop and create a new set of images
- Overheads are less, so image sizes and code are kept low

CSS coding

- Web pages are becoming much more interactive and therefore complex
- The lack of intra browser similarity causes problems
- Many effects created for the latest browser X wont behave the same in browser Y
- How will these work in browser X version 2011
- How will they work when viewed with a PDA/Mobile/Tablet

- Code needs to be written so that it is
 - Neat
 - Easily maintained
 - Degrades well

Outlook

- CSS 2.0
- Layout with CSS
