University of Glasgow

Cascading Style Sheets (part 2)

Web Application Development 2

Specialisation of Presentation

- **Class** and **ID selectors** can be used for finer control
- This involves more planning/effort with document markup
  - But can result in a better user experience
  - It is also very important for manipulating elements in JavaScript
  - The effort also pays off if you use libraries like jQuery
- Class selectors work on a set of specified elements through the **class attribute**
- While ids provide a way to stylise unique elements through the **id attribute**

Class Selectors

- Class selectors allow you to style items with the same (X)HTML element differently
- They work when the **class attribute** of an HTML tag is given a **name**
- The **dot (.)** operator is used to define the class

```
<style>
    .warning {font-weight: bold;}
</style>

<p class="warning">This text will be displayed in bold.</p>
<p>This text will NOT be displayed in bold.</p>
<p class="warning">Bold again here.</p>
```

ID Selectors

- Similar to class but they define a special case for an element
  - IDs are meant to unique and only used once
  - However, browsers are not particularly fussy about enforcing the uniqueness of identifiers
- The **hash symbol (#)** is used to specify a unique ID

```
<style>
    #first-para { font-weight: bold; }
</style>

<p id="first-para">This paragraph will be bold-faced.</p>
<p>This will not be bold.</p>
<p id="third-para">This will not be bold.</p>
```

Descendent Selectors

- Elements that are descended from a particular element are styled according to the rule of the **descendent selector**
  - This means that the rules will be applied to a set of elements in one context but not in another

```
<style>
      p em { color: red; font-weight: bold;  }
</style>
<body>
      <p>this will be the default colour
      <em>this will be red, bold and italics</em>
       back to the default colour</p>
</body>
```

Restricted Class and ID Selectors

- All h2 elements within the class red should be coloured red

```
<style>   .red h2 { color : red; }    </style>
<body>   <div class="red"><h2>I am red</h2> </div>    </body>
```

- All h2 elements whose class is red should be coloured red

```
<style>   h2.red { color : red; }    </style>
<body>   <h2 class="red">I am red</h2>     </body>
```

- All h2 elements within an element with ID red should be coloured red

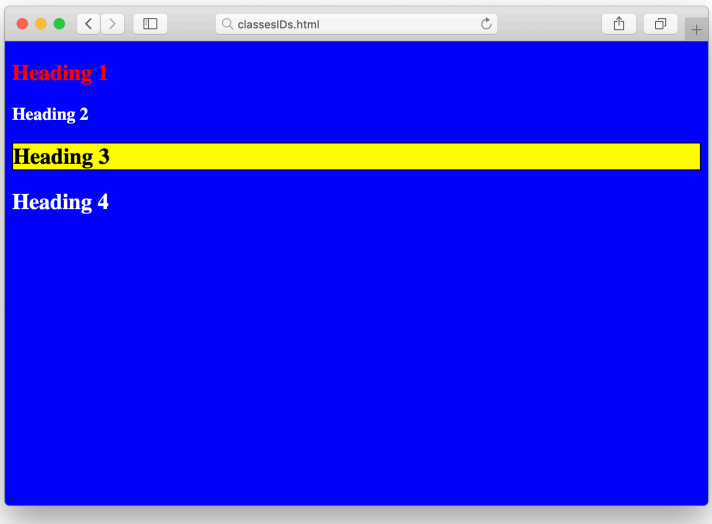
```
<style>   #red h2 { color : red; }    </style>
<body>   <div id="red"><h2>I am red</h2> </div>    </body>
```

Inheritance of Style

- The order of application of styles is through inheritance
- Styles are applied not only to a specified element, but also to its descendants
  - For example, below will inherit the style of its parent <p>

```
<style>
      p { color: red;
          font-weight: bold  }
</style>
<body>
<p>happily red <em>really emphasizing redness</em></p>
</body>
```

Example



classesIDs.html

```
<!DOCTYPE html>
<html>
   <head>
      <link rel="stylesheet" type="text/css" href="stylesheet2.css"/>
      <title>Sample Title</title>
   </head>

   <body>
      <div class="red"><h2>Heading 1</h2></div>
      <div id="red"><h3>Heading 2</h3><h2>Heading 3</h2></div>
      <h2 class="red">Heading 4</h2>
   </body>
</html>
```

stylesheet2.css

```
body {
    background-color: #0000FF;
    color : white
}

#red h2 {
   background-color : yellow;
   border : 1px solid black;
   color : black
}

.red h2 {
   color : red;
}
```

Specificity of Style

- Sometimes multiple rules apply to the same element
- CSS uses a weighting scheme to ensure that there are predictable outcomes with conflicts of style
  - For every ID attribute value given in the selector, add 1,0,0
  - For every class attribute value, attribute selection, or pseudo-class given in the selection, add 0,1,0
  - For every element and pseudo-element given in the selector, add 0,0,1
- Ordering then lexicographic, e.g., 1,0,0 beats 0,5,5

```
h1 {color: red;}                  /* specificity = 0,0,1 */
body h1 {color: green;}           /* specificity = 0,0,2 */
#content h2 {color: silver;}      /* specificity = 1,0,1 */
h2.grape {color: purple;}         /* specificity = 0,1,1 */
```

- Inline CSS overrides all these - effectively 1,0,0,0

The Cascade in CSS

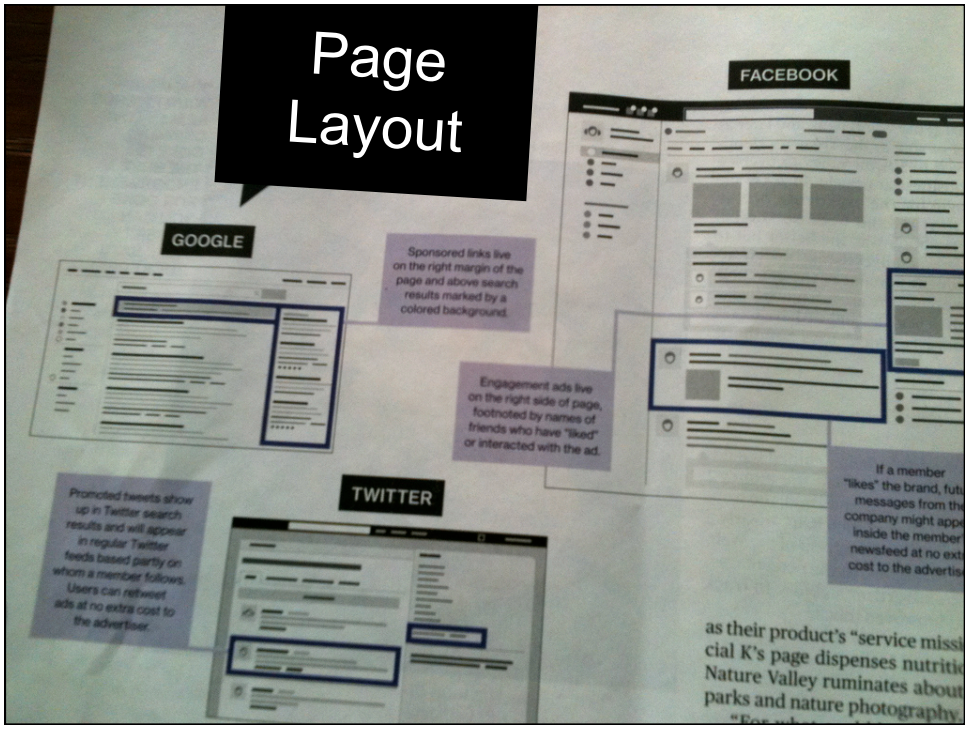
- However, sometimes there is still a conflict between two or more rules.
  - i.e., if they all have the same weight
- CSS is based on a method of causing **styles to cascade together**, which is made possible by combining inheritance, specificity and order
- The purpose of "**cascading**" is to find one winning rule among a set of rules that apply to a given element

Cascading Rules

- Find all rules that contain a selector that matches a given element.
- Sort all declarations by explicit weight applying to the element.
  - Those rules marked important are given higher weight

```
<style>  p { color: gray ! important; }  </style>
```

- Sort all declarations by specificity applying to a given element.
  - Those elements with a higher specificity have more weight than those with lower specificity
- Sort all declarations by order applying to a given element.
  - The later a declaration appears in the style sheet or document, the more weight it is given
  - Declarations that appear in an imported style sheet are considered to come before all declarations within the style sheet that imports them.



Page Layout

- Layout of major elements on a webpage (e.g. columns, navigation bars, sidebars, headers and footers) can be specified using CSS
- In bygone days, tables were heavily used for layout
  - simple to use for simple tasks
  - painful for complex layout
  - tables are meant for content, not layout
- The preferred solution is to divide a page into a collection of (division/section) elements
  - ** … **
  - …

Floating

CSS floating properties allow you to **float** elements horizontally.

Elements can be floated: **left** and **right**, but not up and down!

Elements after the floating element will flow around. So if screen size changes elements will move down.

```
<head>
    <style type="text/css" media="screen">
        .thumbnail {
            float:left;
            width:110px;
            height:90px;
            margin:5px; }
    </style>
</head>

<body>
    <img class="thumbnail" src="clematis_small.jpg">
    <img class="thumbnail" src="clematis2_small.jpg">
    <img class="thumbnail" src="clematis3_small.jpg">
</body>
```

Positioning

CSS positioning properties allow you to **position** an element.

Elements can be positioned using: **top**, **bottom**, **left** and **right** properties.

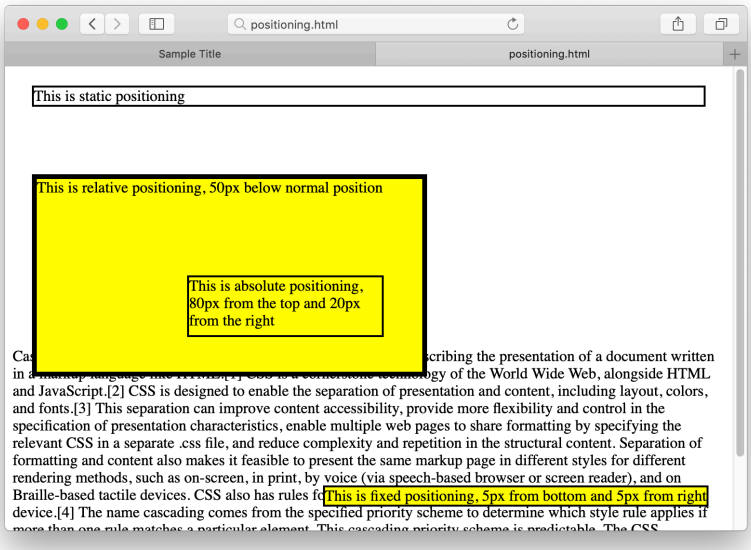
There are four different ways to position: static (default), **fixed**, **relative** and **absolute**.

```
<style type="text/css" media="screen">
    div {
        border: 1px solid #999999;
        margin: 20px;    }

    div.fixed {
        position: fixed;
        top: 30px;
        right: 5px;        }

    div.relative {
        position: relative;
        top: -50px;        }

    div.absolute {
        position: absolute;
        left: 100px;
        top: 150px;        }

</style>
```

Positioning

- **Static Positioning** HTML elements are positioned static by default. A static positioned element is always positioned according to the normal flow of the page. Static positioned elements are not affected by the top, bottom, left, and right properties.
- **Fixed Positioning** An element with fixed position is positioned relative to the browser window. No matter how you scroll or resize. Might mean has to overlap other content
- **Relative Positioning** A relative positioned element is positioned relative to its normal position
- **Absolute Positioning** An absolute position element is positioned relative to the *first parent element that has a position other than static*

Positioning: example

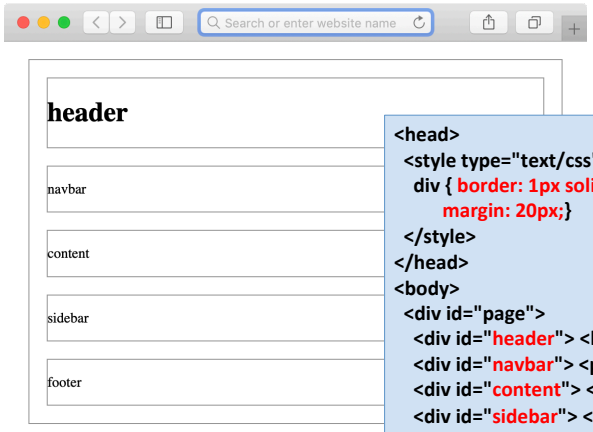


Positioning: example

```
<style type="text/css">
 div {
        border: 2px solid black;
        margin: 20px; }

div.fixed {
        position: fixed;
        bottom: 5px; right: 5px;
        background: yellow}

div.relative {
        position: relative;
        top: 50px;
        width: 400px; height: 200px;
        border: 5px solid black;
        background: yellow}

div.absolute {
        position: absolute;
        top: 80px; right: 20px;
        width: 200px; height: 60px;}
 </style>
```

```
<body>

        <div>This is static positioning</div>

        <div class=fixed>This is fixed positioning, 5px
from bottom and 5px from right</div>

        <div class=relative>This is relative positioning,
50px below normal position

                <div class=absolute>This is absolute
positioning, 80px from the top and 20px from the
right</div>
        </div>

        Cascading Style Sheets (CSS) is a style sheet
language used for describing the presentation of a
document written in a markup language like HTML.
[1] CSS is a cornerstone technology of the World
Wide Web, alongside HTML and JavaScript.[2]
...
</body>
```

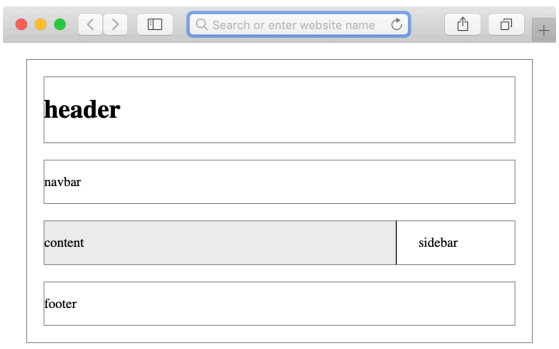
Floating & Positioning




```
<head>
 <style type="text/css" media="screen">
  div { border: 1px solid #999;
        margin: 20px;}
 </style>
</head>
<body>
 <div id="page">
  <div id="header"> <h1>header</h1> </div>
  <div id="navbar"> <p>navbar</p> </div>
  <div id="content"> <p>content</p> </div>
  <div id="sidebar"> <p>sidebar</p> </div>
  <div id="footer"> <p>footer</p> </div>
 </div> <!-- end of page -->
</body>
```

Floating and Positioning

We want content occupying most of width; a sidebar to the right

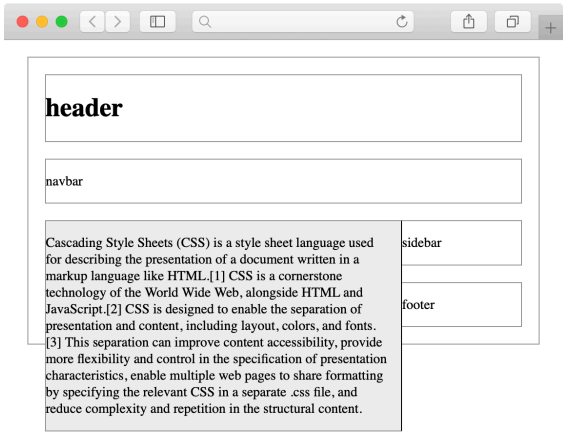



```
#content {
    float: left;
    width: 67%;
    background: #eee;
    margin-top: 0;
    margin-right: 1.67em;
    border-right: 1px solid black;
    padding-top: 0;
    padding-right: 1em;
    padding-bottom: 0;
  }
```

Floating and Positioning

Add text to content div

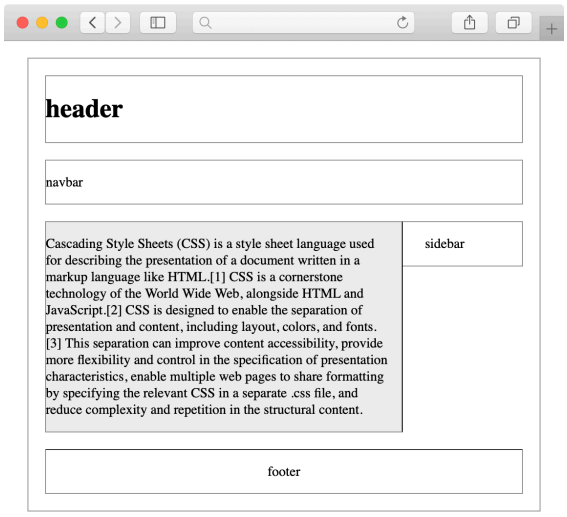
Not what we want: due to the natural flow layout, the footer block has appeared in the incorrect location



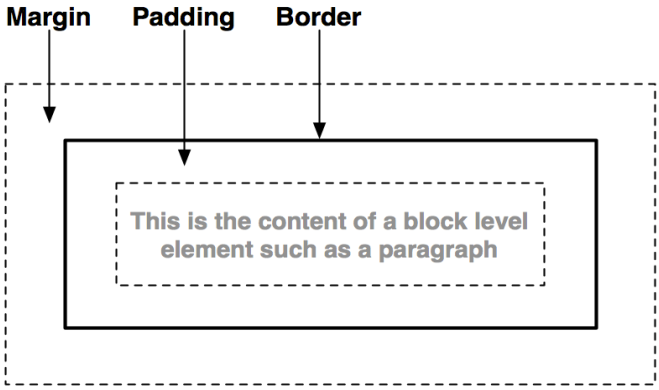
Floating and Positioning

```
#footer {
    clear: both;
    padding-bottom: 1em;
    border-top: 1px solid #333;
    text-align: center;
   }
```

Can set **clear** to **none**, **left**, **right**, or **both**

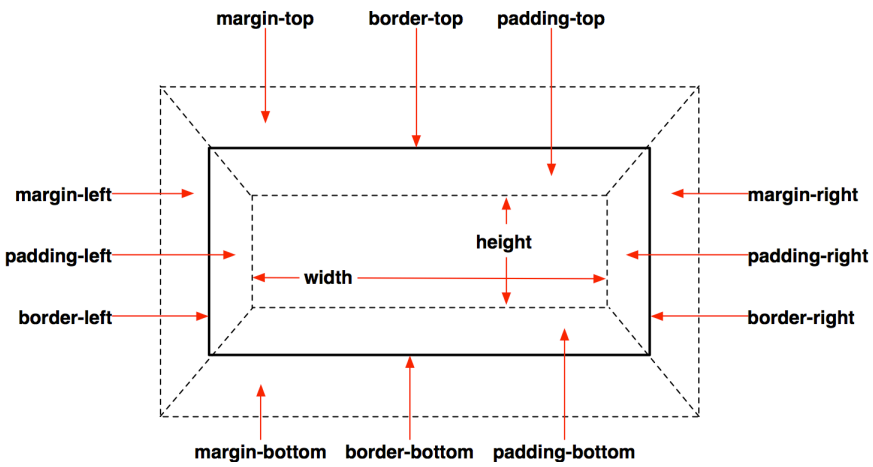


The Box Model

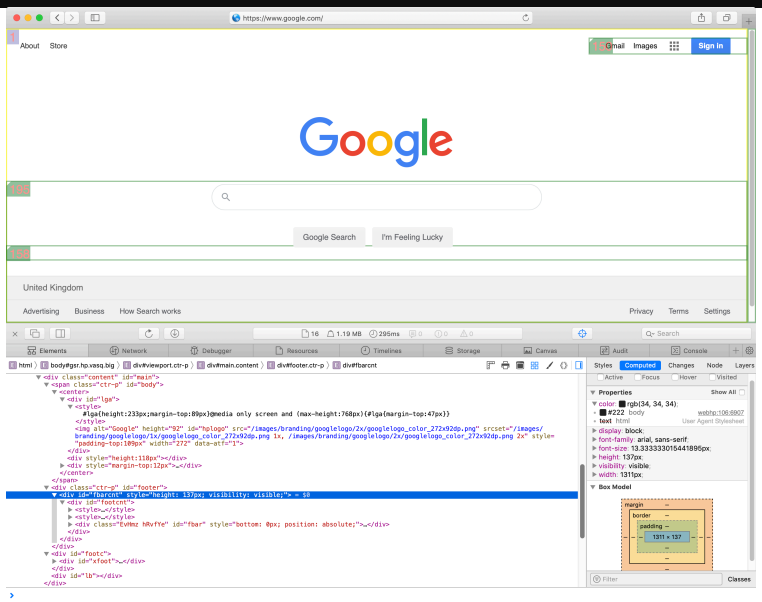


Every element generates one or more rectangular **element boxes** which houses the content. The element box is surrounded by optional amounts of **padding**, **borders**, and **margins**.

Box Model Properties



Developer Tools



Benefits of CSS

- Cascading Style Sheets (CSS) is a method of **separating a document's structure and content from its presentation**
- CSS allows for a much **richer document appearance** than (X)HTML alone
- CSS can save time as the appearance of the entire document can be **created and changed in just one place**
- CSS can improve load times as it **compactly stores the presentation concerns** of a document in one place instead of being repeated throughout the document

Summary

- Separation of concerns is a good principle to adopt
  - simple examples usually don't benefit
  - effort is worth it as complexity increases
- CSS is a powerful method of specifying the style of web pages
  - separates presentation from structure and content

CSS Properties

- The full list of properties can be found: http://www.w3.org/TR/CSS22/propidx.html
- Compact CSS Cheatsheets are useful: http://www.lesliefranke.com/files/reference/csscheatsheet.html

background-color	**border-width**	**font-family**	**height**	**size**
text-align	**width**	**color**	**font-size**	**margin**
padding	**list-style**	**position**	**text-decorations**	**~100+ more!**