p.c.windridge@staffs.ac.uk

CSS Basics

Introduction

The purpose of this tutorial is to introduce you to how you can affect the visual appearance of your web page using Cascading Style Sheets (CSS).

A good visual appearance is important for the majority of visitors to your website. Correctly marked up content with correctly applied CSS rules to affect visual display will also increase the accessibility of your website.

Inline Elements, Block-Level Elements and Meaning

Elements in HTML are either block-level or inline elements. An example of a block-level element is a paragraph *<p>* element. An example of an inline element is an anchor *<a>* element.

Their default behaviour when rendered in a browser differs. A block-level element will span the entire window and the following element will appear underneath it. An inline element will not force a new line but will appear 'inline' with the content that surrounds it.

It is important to realise that this default behaviour is artificially imposed by the browser (or other user agent) that renders them. A paragraph has a distinct meaning within an HTML page that differs from every other element.

This 'HTML meaning' of what it is to be a 'paragraph' does not depend upon or result from the visual appearance of its default appearance when rendered in a particular browser. Its meaning when marked up by HTML should be considered separately to how it will be rendered in a visual browser.

That is not to say that visual appearance isn't important. The impact, effect or emphasis of a paragraph can be strongly influenced by how it is visually displayed. However, the visual appearance is the responsibility of CSS. Visual concerns should therefore be left to CSS as much as is possible.

CSS allows you to change the default behaviour of block-level and inline elements as you see fit.

The Box Model

All elements in HTML (whether block-level or inline) conform to the box model when they are rendered. This means that every element in an HTML page can be considered as a rectangular box as shown in Figure 1.

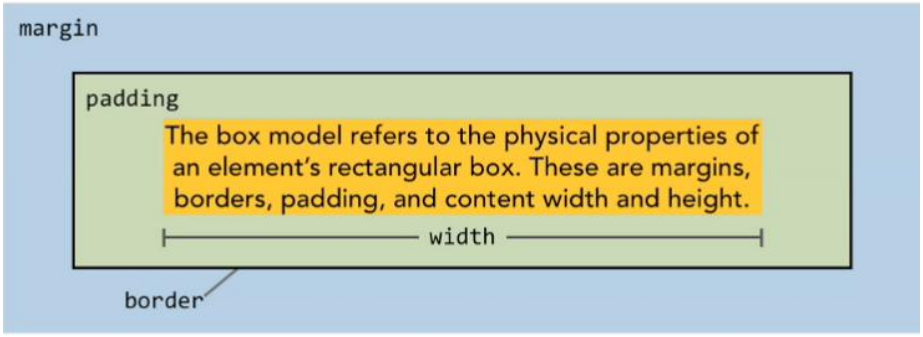

Figure 1: The 'Box Model' (Taken from "CSS: Page Layouts" by James Williamson at Lynda.com)

Each box is comprised of:

- Margin – this is the space between elements
- Border – the boundary line between the margin area and the element
- Padding – the space between border edge and content of the element (this takes the background colour of containing element)
- Width – the total horizontal space taken up by the element on the display (can change depending on how the box model is rendered – see "Example" section)
- Height – the total vertical space taken up by the element on the display (can change depending on how the box model is rendered – see "Example" section)

Example

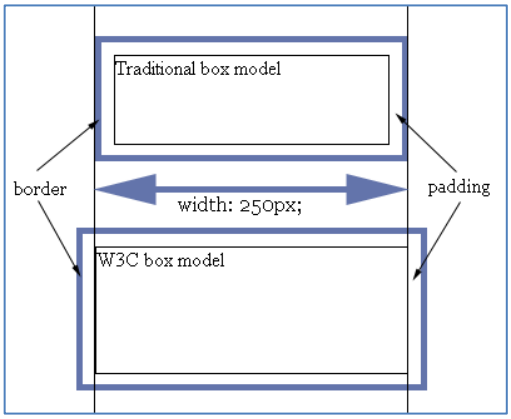

Figure 2: Different box model 'widths' (Taken from: http://quirksmode.org/css/user-interface/boxsizing.html: Accessed 15/09/2013)

Figure 2 shows the 'traditional' box model vs. the 'W3C' box model. Generally browsers will use the 'W3C' approach but there is an obvious danger of your carefully crafted web page appearing differently depending upon the model used. Including *<!DOCTYPE html>* at the start of your page removes this problem.

Following the W3C model and taking the example in Figure 2 as a *<p>* element with the following dimensions:

- Paragraph (content) width: 250px
- Padding: 15px
- Border: 10px
- Margin: 30px

The overall width of the content block is:

$$width + left\,padding + right\,padding + left\,border + right\,border + left\,margin + right\,margin$$

=

$$250px + 15px + 15px + 10px + 10px + 30px + 30px = 360px$$

The amount of horizontal display space occupied by the *<p>* element would be 360px.

Viewing the web page as boxes

Save the following HTML page on your 'T' drive as "Boxes.html" and view it in a browser.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Form Example</title>
    <style>
        * {border: solid #ff0000 1px;}
```

```
    </style>
</head>
<body>
<form id="order" method="get" action="Process.php">
    <fieldset>
        <legend>Order Form</legend>
        <p><label>First name: <input type="text"
name="fname"></label></p>
        <p><label>Last name: <input type="text"
name="lname"></label></p>
    </fieldset>
    <fieldset>
        <legend>Pick your colour</legend>
        <p><label><input type="checkbox" name="colour" value="red">
red</label></p>
        <p><label><input type="checkbox" name="colour"
value="green"> green</label></p>
        <p><label><input type="checkbox" name="colour"
value="blue"> blue</label></p>
    </fieldset>
    <p><label>Choose file: <input type="file"
name="file"></label></p>
    <p>
        <label>Choose your town:
            <select name="town">
                <option value="">Please Select ...</option>
                <option value="Swindon">Swindon</option>
                <option value="London">London</option>
                <option value="Stafford">Stafford</option>
            </select>
        </label>
    </p>
    <p><label>Message: <textarea
name="message"></textarea></label></p>
    <p><input type="submit"></p>
    <p><input type="reset"></p></form>
</body>
</html>
```

Can you see what is causing your form elements to appear outlined in red?

CSS Rules

CSS enables a web site developer to write rules that determine how the content of an element should be displayed. The rules are associated with HTML elements of the mark-up.

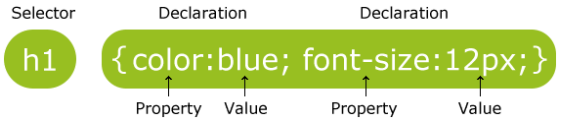

Figure 3: CSS Declaration (taken from http://www.w3schools.com/css/css_syntax.asp: Accessed 15/09/2013

Figure 3 shows two declarations for the *<h1>* (heading 1) element. The entire line is referred to as a 'rule'. This rule says that all *<h1>* elements should appear in blue font and have a size of 12px.

Rule structure

Every CSS rule has two parts: the selector and the declaration.

The selector identifies the element or elements to which the rule will apply. The declaration provides the style to be applied in the form of a property/value pair.

The declaration is enclosed in curly braces. You can include as many declarations as you wish for a given selector.

Linking CSS to your HTML

There are three ways to use CSS with your HTML:

- Inline style – attached to individual elements
- Internal style sheet – contained within (and applying to) a single page
- External style sheet – contained in a separate file and linked to one or more HTML pages

Inline style

A CSS rule applied directly to an element will override any other styles that may be applied using either an internal or external style sheet.

The following is an example of an inline style applied to a *<h1>* element.

```
<h1 style="color: blue;">Example</h1>
```

The main disadvantage of using this approach is that it mixes up display information with content mark-up. It has the effect of making your webpages confusing, difficult to change and larger than necessary.

Internal style sheet

A CSS rule in an internal style-sheet will override a rule in an external one but will be overridden by an inline style.

The example web page in "Viewing a web page as boxes" above used an internal style-sheet to place a red border around the page elements.

Internal style-sheets are defined in the *<head>* element of a web page as a *<style>* element. The following is an internal style-sheet that sets the font colour of all *<h1>* elements in the web page to blue.

```
<head>
  <meta charset="utf-8">
  <title>Example</title>
  <style>
```

```
    h1 {color: blue;}
  </style>
</head>
```

You can include as many rules as you wish in the internal style-sheet.

The main disadvantage of an internal style-sheet is the same as the main disadvantage of the inline style though mitigated by the isolation of the internal style-sheet within the *<head>* element.

The difficulty of changing a web site appearance and the extra bytes of data needed to communicate style information for the page becomes more pronounced when dealing with a web site of hundreds of pages.

External style-sheet

An external style-sheet is a text file with a ".css" extension. The content of the file is made up of one or more CSS rule.

A CSS rule in an external style-sheet may be overridden by a rule in an internal style-sheet or an inline style.

The following is an example of the contents of an external style-sheet.

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

If this were to be saved as "Style.css" in the same directory as a web page the following would be required in the *<head>* element of the web page to link it to the external style-sheet.

```
<head>
  <meta charset="utf-8">
  <title>Example</title>
  <link rel="stylesheet" type="text/css" href="Style.css">
</head>
```

The main disadvantages of the inline style and the internal style-sheet are overcome by this approach. A web site becomes much more scalable.

Inheritance and Cascading

Many CSS property values are inherited. This means that a child element can take on a blue font that has been specified for their parent. For example, in the following CSS the *<body>* element has its font family and colour specified. All elements within the body tag (in other words every element to be displayed) will take on these property values.

Inheritance can be enforced for a particular property. The example below is adapted from http://www.w3.org/TR/CSS2/cascade.html#inheritance: Accessed 16/09/2013. It sets the font colour for the *<body>* element as black and then sets all elements to inherit the colour property value.

```
body {
  color: black;
  background: white;
```

```
}

* {
  color: inherit;
  background: transparent;
}
```

Application of CSS rules 'cascades' according to a weight that is applied to each rule. In very general terms the cascade works as follows:

1. Find all declarations for an element
2. Sort in ascending order of precedence according to where/how they are set and whether or not they are marked up as "!important"
3. Sort rules with the same origin and importance according to their specificity
4. Sort in order of occurrence in the style-sheet.

Selectors

Selectors are a versatile way of identifying (to name a few):

- an individual element (using the *id* attribute)
- all elements of a particular type (for example *<p>* elements)
- all elements of a particular class (using the *class* attribute)
- all elements on the page (using the universal selector)
- the child elements of a given element (for example all the *<p>* elements within a *<form>* element)

You can think of a selector as a pattern that you can use to match against your HTML page. Where there is a conflict between elements identified by a particular pattern, for example a rule for all *<h1>* elements and a rule for a particular *<h1>* element, the more general rule will be overridden.

http://www.w3schools.com/cssref/css_selectors.asp: Accessed 16/09/2013 provides a good reference to the different kinds of selector patterns that are available for you to use. For even more details on selectors see http://www.w3.org/TR/CSS2/selector.html#type-selectors: Accessed 16/09/2013.

The following are examples of the more commonly used ones.

Universal Selector

The universal selector (*) is implemented when you want a rule to apply to all elements:

```
* {
   color: Red;
   font-family: arial, verdana, sans-serif;
}
```

The universal element is top-level, which means all other rules are more specific and so override it.

Type Selector

A type selector simply matches the name of an element in mark-up:

```
body {
   font-family: arial;
   color: blue;
}
```

When the same rule is to be applied to more than one element, multiple selectors can be grouped together, separated by commas:

```
p, h1, h2 {
   font-family: arial, verdana, sans-serif;
   color: blue;
   font-weight: bold;
}
```

Class Selector

The class selector (.) enables you to apply a rule to all elements with a *class* attribute that has a given value.

```
.copyright {
   color: black, font-family: "Times New Roman";
}
```

This CSS rule states that any element that has an attribute *class="copyright"* will have black text and use the Times New Roman font.

You can also match to element types with a given class attribute. The following rule will be applied to all *<p>* elements with that have an attribute *class="copyright"*.

```
p.copyright {
   color: Black, font-family: "Times New Roman";
}
```

ID Selector

The ID selector (*#*) can be used to apply a style to a unique element.

```
#copyright {
   color: Silver;
   font-weight: bold;
}
```

This CSS rule will be applied to the element that has the *id* attribute *id="copyright"*.

Pseudo Class Selectors

There are some pseudo class elements that are defined in CSS that are useful:

:link – Applies to hyperlinks that have not been visited

:visited - Applies to hyperlinks that have been visited

:hover – Applies to when an element is hovered over

:active – Applies when an element is being activated

:focus – Applies when an element has the focus

An example of this is shown below

```
a:link    { color: red }    /* unvisited links */
a:visited { color: blue }   /* visited links   */
a:hover   { color: yellow } /* user hovers     */
a:active  { color: lime }   /* active links    */
```

CSS Positioning

Like everything else to do with display, layout should be done using CSS – it should not be part of the HTML mark-up. Fortunately CSS offers a number of properties that give the developer a high level of control over the positioning of elements on the page.

Normal flow

In normal flow the element 'boxes' conform to their block-level or inline context. In a block-level context they occur one after the other vertically, in an inline context they occur one after the other horizontally. Once a box has been laid out it can be shifted 'relative' to its position which is referred to as relative positioning (for example: *position: relative; left: -5px*).

Float

A box can be 'floated' left or right on its current line within its bounding container. Content can then flow around the box to the right (if floated left) or to the left (if floated right). When an element is 'floated' it is taken out of normal flow.

The clear property provides a mechanism for switching any preceding float property off. The effect of clearing an element is to move it to the next line and resume normal flow. See http://www.w3.org/TR/CSS2/visuren.html#floats: Accessed 16/09/2013.

Absolute positioning

When using *position: absolute* the box is explicitly positioned with respect to its containing block. An absolutely positioned element is taken out of normal flow. If it is 'fixed' (*position: fixed*) it is still absolutely positioned but will not move even when the display is scrolled.

Positioning the navigation and content in columns using *float*

It is common to see navigation sections to the left of a web page and the main content to the right (as a two column layout). The following CSS achieves this effect using the *float* property.

```
nav {
  float: left;
  width: 20%;
  margin: 10px;
}
section {
  width: 70%;
  float: left;
  margin: 10px;
}
```

Note the use of the *width* property to make the *float* visible.

Were the page to include a footer the following CSS would be used to clear the floated elements and return the page to normal flow.

```
footer {
  clear: left;
}
```