CSS Basics - Module 2

In this module we'll take a second look at CSS, including some additional selectors, the CSS Box Model - which is important to understand in terms of calculating how much space an element will take up on a page - inline and block-level elements.

We'll also look at two new elements - divs and spans and how we can use these for applying styling and layout. We'll also look at how to adjust the layout of elements in relation to each other by using the float property.

Finally we'll cover specificity and media types.

Review of Previous Module

In the previous module we looked at some additional HTML elements and concepts.

We discussed **meta tags** and their use and then some additional HTML elements that can be used to identify certain types of content, such as **addresses**, **abbreviations** and **quotations**.

Then we moved on to look at a different type of list - the **definition list** - that can be used to define **terms** and associated **descriptions**.

Following that we looked at **images**, **tables** and **forms** - although we'll come back to forms over the coming modules.

Finally we took an initial look at some of the new features of **HTML5**.

Topics for this module

In this module we'll return to CSS and cover the following topics:

- Further CSS selectors – attributes, pseudo-classes and adjacent selectors
- The CSS Box Model
- Margins and padding
- Borders and outlines
- Layout – positioning with floats
- Specificity
- Media types

Review of Exercises

Before we begin on this module we'll take 10 minutes to review any exercises from the previous module.

Further CSS Selectors

In the previous CSS module we looked at the basic types of CSS selectors - elements, classes and ids. In this module we are going to look at some addition selectors:

attributes, pseudo-classes and adjacent selectors

- attribute selectors
- pseudo-class selectors
- adjacent selectors

CSS Selectors - attributes

We already know that HTML elements can have attributes defined in their opening tags. These attributes can be used as CSS selectors:

```
input[type="text"]{
  width: 250px;
  display: block;
  margin-bottom: 10px;
  background-color: #CCC;
}
```

In the example above the CSS declarations will be applied to any elements that have a **type** attribute with a value of "**text**".

CSS Selectors – pseudo classes

In the previous CSS module we said that an anchor element can be in one of four states, and that we can use pseudo classes to incorporate these into selectors.

```
a:link {color:#FF0000;}      /* unvisited */
a:visited {color:#00FF00;}   /* visited */
a:hover {color:#FF00FF;}     /* mouse over */
a:active {color:#0000FF;}    /* active */
```

Note:

In order for these pseudo class selectors to work reliably the a:hover rule must come after a:link and a:visited, and a:active MUST come after a:hover in the CSS

Tip – you can remember this sequence with '**LoVe HAte**'

The first-child pseudo-class

Another pseudo-class is the first-child. This targets the first occurrence of a selector within another element.

```
p:first-child {color:#CCC;}
```

This will select any <p> element that is the first child any other element.

```
<body>
  <p>This will be matched.</p>
  <p>This will not be matched.</p>
</body>
```

:first-child – continued

The :first-child pseudo class can also use a '>' operator to apply to specific elements.

For example, the rule below will target the first <a> element inside any <p> element

```
p > a:first-child {color: #f00;}
```

```
<p>
  <a href="news.html">This will be matched.</a><br>
  <a href="blog.html">This will NOT be matched.</a>
</p>
<blockquote>
  <a href="blog.html">This will NOT be matched.</a>
</blockquote>
```


CSS selectors – adjacent selectors

The + **operator** can be used to specify adjacent selectors.

For example, the following rule will match a p element that is the first element after an h1 element:

```
h1 + p {color: #fff;}
```

Operators can also be combined

```
p > a:first-child + h2 {color: #f00;}
```

What would the rule shown above select?

Exercise

Modify the code for the 'explorer-stats.html' file.

Also use one of the selector types that we have just discussed to target your CSS the first header row.

Use one of the selector types that we have just discussed to target your CSS at the first <td> element on each row.

Inline and block elements

HTML elements fall into one of two categories

- Inline element
- Block elements

Inline elements have no effect on layout or structure of a document and their content.

Block level elements are inserted into the document as a rectangular block, or box.

By default they will take up the full width of the element in which they are contained and will have a line break after them.

Inline elements

One inline element that we have already looked at is the anchor element <a>.

```
<p>Hi, why don't you <a href="http://
www.wombats.com">visit our Wombats website?</a><p>
```

This is displayed as:

Hi, why don't you visit our Wombats website?

The only effect of the <a> tags around the text is to enable it as a link and, by default, underline it.

Let's look a some other inline elements.

Confidential

Inline elements

Other inline elements include the following:

**** – used to mark emphasized text – usually be making the text italic

**** – used to identify text as strong, or bold

<i> – used to identify text that should be italicized

<b> – used to identify text that should be displayed in bold

<sub> – used identify text that should be displayed in subscript notation

<sup> – used identify text that should be displayed in subscript notation

Inline elements -

A span is an inline element that has no semantic meaning - it doesn't represent any particular type of content and nor does it have any effect on the layout of a page.

The main reason for using a span is to act as a container element for a piece of content that you wish to target with CSS for styling or layout but which doesn't have a natural container of its own.

```
span.key {color: green;}

<p>Key terms are <span class="key">shown in green text
</span></p>
```

Key terms are shown in green text

Block level elements

Whereas inline elements have little or no effect on the layout of the page, block level elements do.

Heading and paragraphs are examples of block level elements. The browser makes them the full width of their containing element by default and adds a line break immediately before and after them. We can see this by adding a border and background color with CSS.

This is a heading with a border

This paragraph has had a border applied, it has also had a background color.

This is another paragraph. You will also notice that there is some space between the heading and the paragraph and between each paragraph. This is because the browser adds vertical margins by default.

Block level elements – div

Like a span, a div is an element that has no semantic meaning – it doesn't represent any particular type of content. However, unlike a span, a div is a block-level element and so it acts as a rectangular wrapper around the elements that it contains.

The main reason for using a span is to act as a container element for other elements to be targeted with CSS for styling or layout.

```
div.shaded {background-color: yellow;}

<div class="shaded">
 <h1>Level 1 heading</h1>
 <p>This heading and paragraph are contained in a
&lt;div&gt; element with a yellow background color.</p>
</div>
```


Confidential

Block level elements – div


```
div.shaded {background-color: yellow;}

<div class="shaded">
 <h1>Level 1 heading</h1>
 <p>This heading and paragraph are contained in a
&lt;div&gt; element with a yellow background color.</p>
</div>
```

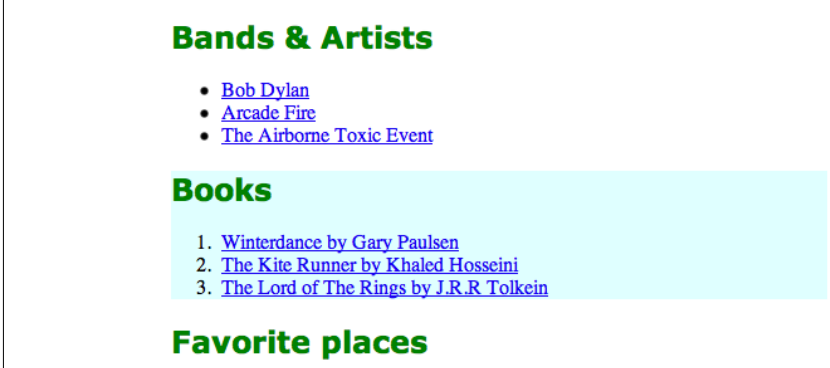

Level 3 heading

This heading and paragraph are contained in a element with a yellow background color.

Exercise

Update your 'My favorite stuff' page so that the heading and list for the Books section have a background color applied.

Make sure that background covers both the heading and the list.



Confidential

The CSS Box Model

Elements act as containers – either for their own content, or for other elements and content.

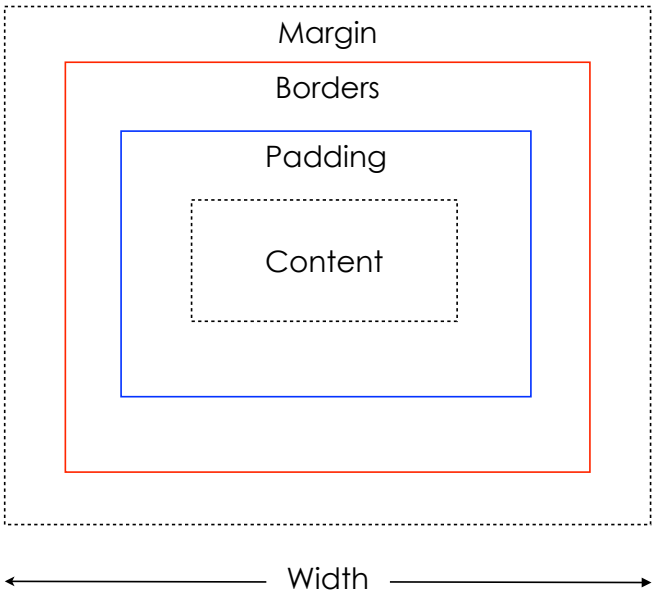
First of all padding can be added around the content. This can either be added around all sides of the content or just on specific sides.

Borders are then applied around the content and its padding.

Finally, a margin can be applied to any side or around the complete container.

By adding together all these components we can determine the overall width of the element.

The CSS Box Model



Padding and margins

Padding and margins are applied to elements with CSS.

Padding surrounds the content itself.

Border and outlines are then added around the padding.

Margins are applied outside any borders and outlines.

Padding

The following CSS properties can be used to apply padding to an element:

- padding-top
- padding-bottom
- padding-left
- padding-right

```
{padding-left: 15px;}

{padding-right: 25px;}
```


Padding

There is also a shortcut property that can be used to apply padding to more than one side at the same time:

- padding

```
{padding: 15px 10px 5px 20px;}
```

The property can take from 1 to 4 values. If 4 values are provided they are applied in this order:

top right bottom left (**TR**ou**BL**e)

Padding

If the shortcut padding property is used with less than 4 values then they are applied as follows:

```
{padding: 15px 10px 5px;}
/* top=15px, right=10px, left=10px, bottom=5px */

{padding: 15px 10px;}
/* top=15px, right=10px, left=10px, bottom=15px */

{padding: 15px;}
/* top=15px, right=15px, left=15px, bottom=15px */
```


Margins

The following CSS properties can be used to apply margins to an element:

- margin-top
- margin-bottom
- margin-left
- margin-right

```
{margin-left: 15px;}

{margin-right: 25px;}
```

Margins

There is also a shortcut property that can be used to apply a margin to more than one side at the same time:

- margin

```
{margin: 15px 10px 5px 20px;}
```

The property can take from 1 to 4 values. If 4 values are provided they are applied in this order:

top right bottom left (**TR**ou**BL**e)

Margins

If the shortcut margin property is used with less than 4 values then they are applied as follows:

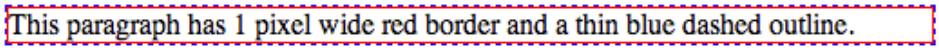
```
{margin: 15px 10px 5px;}
/* top=15px, right=10px, left=10px, bottom=5px */

{margin: 15px 10px;}
/* top=15px, right=10px, left=10px, bottom=15px */

{margin: 15px;}
/* top=15px, right=15px, left=15px, bottom=15px */
```

Borders and Outlines

Elements can have borders and outlines applied with CSS.

Borders are displayed along the outer edges of the element and contain the content and any padding.

Outlines are located outside any borders.



Borders

There are 3 CSS properties that can be set for borders.

- border-width
- border-style
- border-color

```
{border-width: 1px;}

{border-style: solid;}
/*can be solid, dotted, dashed or double */

{border-color: green;}
```

Borders

There is also a shortcut property than can be used to apply a combination of properties in one go:

- border

```
{border: 1px solid red;}
```

The properties are usually applied in the following order:

border-width border-style border- color

Any of the properties can be omitted:

```
{border: 1px red;} /* use existing value for style */
```


Outlines

There are 3 CSS properties that can be set for outlines.

- outline-width
- outline-style
- outline-color

```
{outline-width: 1px;}

{outline-style: solid;}
/* see http://www.w3schools.com/cssref/
pr_outline-style.asp for options*/

{outline-color: green;}
```

Outlines

There is also a shortcut property than can be used to apply a combination of properties in one go

- outline

```
{outline: green solid 1px;}
```

The properties are usually applied in the following order:

outline-color outline-style outline-width

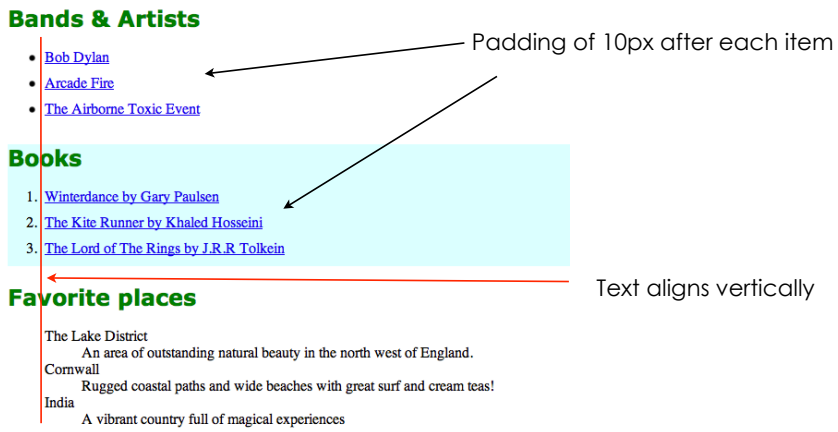
Any of the properties can be omitted:

```
{outline: green 1px;} /*use existing value for style*/
```


Exercise

Update your 'My favorite stuff' page so each item in the ordered and unordered lists has 10 pixels of padding applied at the bottom.

Also, make the definition list indented so that the text of the definition terms align vertically with the text of the other two lists.



CSS Layout - Floats

So far we have concentrated on how CSS can be used for the styling and formatting of elements and content.

The other use of CSS is to control the layout or positioning of elements on a page.

The browser adds each element to the page in the order that they occur in the underlying HTML document. Any in-line elements are appended to the previous element with applying line breaks and they only occupy the amount of space needed for their content.

For block-level elements the browser will add a line break before and after the element and will make the element's width that of its containing (parent) element.

CSS Layout - Floats

Two block level elements will therefore appear one under the other.

```
<ul>
  <li>Widgets</li>
  <li>Wombats</li>
</ul>

<ol>
  <li>Happy</li>
  <li>Sad</li>
</ol>
```

Each list is a block-level element that occupies the full width and has line breaks before and after. A border has been applied to show the width of each element.

- Widgets
- Wombats

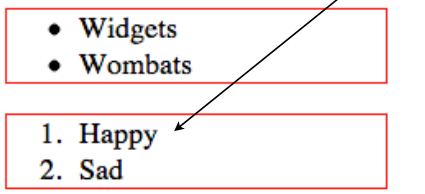
1. Happy
2. Sad

CSS Layout – Floats

There may be times however when we want to display the elements next to each other. Perhaps we could just make the elements narrower – say 40% of the page width and see if that works.

```
{border: solid 1px red; width: 40%;}
```

The lists are now only 40% of the page width, but still appear one below the other.



- Widgets
- Wombats

1. Happy
2. Sad

Confidential

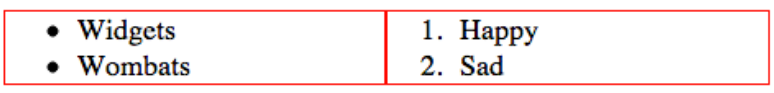
CSS Layout – Floats

The solution is to apply a float property to the elements, in this case we float the elements left.

```
{border: solid 1px red; width: 40%; float: left;}
```

The float property allows an element to be floated left or right (not up or down) If an element is floated then other elements will flow around it.

Now the lists appear side by side.



- Widgets
- Wombats

1. Happy
2. Sad

Specificity

We have already seen that CSS rules are applied in the order in which they occur in the cascade, and that the rule closest to the final content wins.

However, there may be times when the browser decides that an earlier rule is more relevant and that it will not override it.

Specificity is a term that is used to describe the relevance that a browser will attach to a particular element.

The browser calculates this in the form of a numeric value for each selector and the selector with the highest specificity will be the one that is applied.

Specificity

If the selectors are the same then the latest one will always take precedence. For example, if you had:

```
p { color: red; }
p { color: blue; }
```

In this example p elements would be colored blue because that rule came last.

You won't usually have identical selectors with conflicting declarations on purpose (because there's not much point).

Conflicts quite legitimately come up, however, when you have nested selectors.

Specificity

You won't normally have identical selectors with conflicting declarations. More likely you may have a case where the same selector is used in different rules.

```
div p { color: red; }
p { color: blue; }
```

It might seem that p elements within a div element would be colored blue, as a rule to color p elements blue comes last, but they would actually be colored red due to the specificity of the first selector.

The more specific a selector, the more preference it will be given when it comes to conflicting styles.

Specificity

The specificity of nested selectors is calculated as follows:

- Every id selector has a value of 100
- Every class selector has a value of 10
- Every HTML selector has a value of 1.

Then the values are added to give the specificity value. So, in the previous example:

```
div p { color: red; }
p { color: blue; }
```

The first rule has a specificity of 2 whereas the second has a value of 1 – so the first rule wins.

Specificity - Examples

`h1` has a specificity of 1 (1 HTML selector)

`div h1` has a specificity of 2 (2 HTML selectors; 1+1)

`.note` has a specificity of 10 (1 class selector)

`div p.note` has a specificity of 12 (2 HTML selectors and a class selector; 1+1+10)

`#content` has a specificity of 100 (1 id selector)

`body #content .article p` has a specificity of 112 (HTML selector, id selector, class selector, HTML selector; 1+100+10+1)

So if all of these examples were used, then `body #content .article p` would take priority over all of them, regardless of the order in the cascade.

CSS - Media Types

In the majority of cases we use CSS to apply styles to HTML pages.

The media at-rule allows CSS to be applied to other media types.

```
@media print {
    body {
        font-size: 10pt;
        font-family: times new roman, times, serif;
    }

    #navigation {display: none;}
}
```

The CSS above will be applied when a page is printed.

CSS – Media Types

The syntax of the media-at rule is as follows:

```
@media media-type { /* rules for media type /* }
```

Example media-types are:

- **all** – for every media under, over, around and in the sun.
- **aural** – for speech synthesizers.
- **handheld** – for handheld devices.
- **print** – for printers.
- **screen** – for computer screens.

A full list of media types can be found at:

http://www.w3schools.com/css/css_mediatypes.asp

Module Summary

In this module we looked at some additional CSS selectors, such as the first-child pseudo class and also at attribute selectors.

We discussed inline and block-level elements, the CSS box model, borders and outlines and padding and margins.

We also covered specificity and media types.

Labs

Your instructor will upload the labs for this module to Basecamp after the session ends.

Questions

Do you have any questions before we move on to the next module

?

Don't worry, if you think of something later then just ask!