Session four: CSS page layout

In this fourth session we shall create a simple CSS page layout, learning more about CSS along the way.

By the end of the session we shall have added layout styles to our existing external style sheet and created a new HTML page with a two column CSS layout.

We shall also learn about the CSS box model; and how to add padding, borders and margins to our page elements.

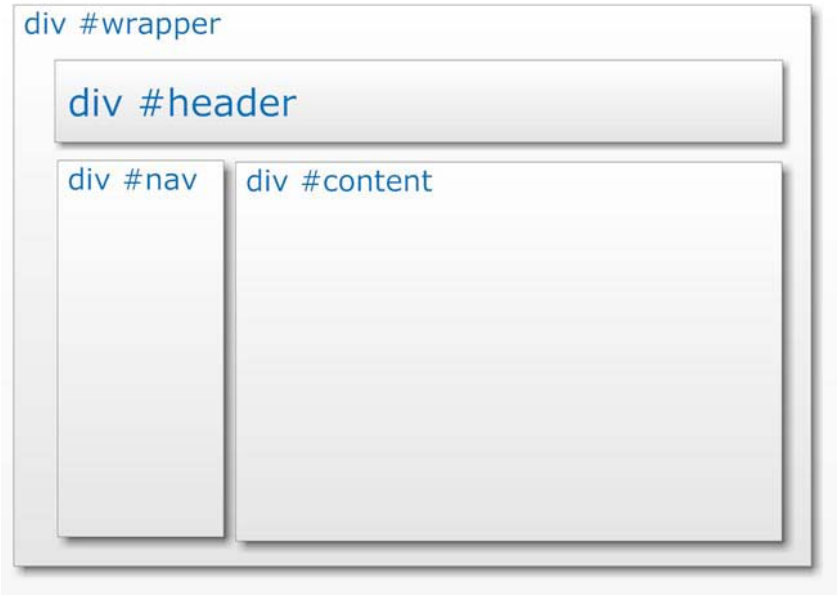
Read through the web pages for this session (links below) and follow the practical exercises to set up your first CSS page layout.

Session four

CSS page layout

Let's look in detail at how we can set up the layout of our page using CSS.

The simplest layout to start off with is a header (often called a banner) section which spans the width of the page, with a left sidebar and main content area beneath the header. A wrapper div encloses all the other divs. It's a good idea to sketch out your layout on paper before you start. Here's a diagrammatic example:



The **wrapper** enables us to apply a width to the overall design, and to centre it in the browser window, however wide the user's browser might be.

To create this layout, first set up the styles in the stylesheet (this is the example as shown in Session three, CSS selectors).

Note that each id name is preceded by a **#** sign.

```
        body {text-align: center;
              }
#wrapper {width: 80%;
          margin: 0 auto;
          text-align: left;
         }
 #header {color: #FFF;
          background-color:#2D73B9;
         }
    #nav {color: #FFF;
          background-color:#00A54E;
          width: 30%;
          float:left;
         }
#content {color: #FFF;
          background-color:#E3372E;
          margin-left:32%;
         }
```

We haven't seen some of these style properties before, so I'll describe their purposes:

body {text-align: center; } : this style declaration centres elements contained within the body tag

#wrapper styles:

width:80%; : this sets the width of the page's contents to 80% of the width of the browser window, whatever that width may be

margin: 0 auto; : sets the top and bottom margin of the wrapper div to zero, and the right and left margins to auto (i.e. the browser will make right and left margins equal, which has the effect of centring the page, when used in combination with the text-align: center property added to the body selector)

text-align: left; : this sets the text-alignment to left, overriding the text-align: center applied to the body selector which we used to centre the #wrapper

#header has only color and background-color style properties added.

#nav, as well as color and background-color, has the following styles applied:

width: 30%; : sets a width for the #nav div; we must give an element a width before we add the float property to it. Setting the width at 30% means that the #nav div will be 30% of the width of the #wrapper div, the element which contains it

float: left; : floats the selected element to the left of elements following it on the page (i.e. the #content div) within the containing element (i.e. the #wrapper)

#content, as well as color and background-color, has the following style applied:

margin-left: 32%; : gives the #content div a left-hand margin so that, rather than moving up behind the floated #nav div, its background and text content are positioned to the right of the left-floated div. (See how this works in the screenshots below.)

You have seen the HTML code for this page before, but here it is again, as a reminder of how the id selectors are applied using tags:

```
<!DOCTYPE HTML public "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/HTML4/loose.dtd">
<html>
<head>
<title>id selectors </title>
<meta http-equiv="content-type" content="text/HTML; charset=UTF-8">
<link href="css/styles.css" rel="stylesheet" type="text/css">
</head>
```

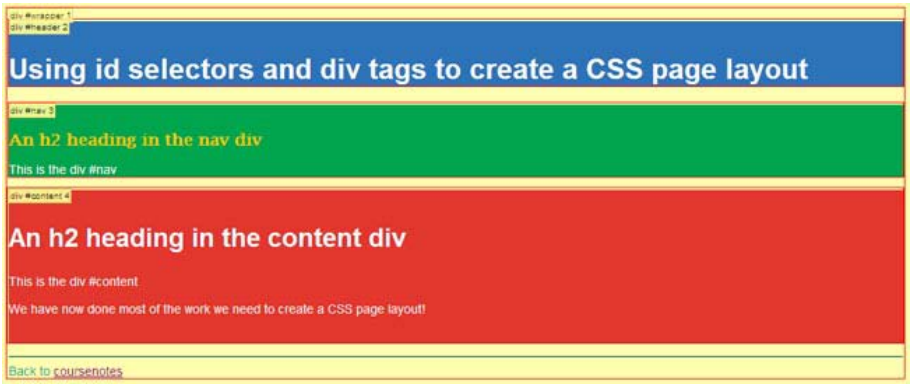
```
<body>

<div id="wrapper">

<div id="header">
<h1>Using id selectors to create a CSS page layout </h1>
</div><!--closes header div-->

<div id="nav">This is the div #nav</div><!--closes nav div-->

<div id="content">
<p>This is the div #content</p>
<p>We have now done most of the work we need to create a CSS page layout!</p>
</div><!--closes content div-->

<hr class="green">
Back to <a href="../selectors.html#back2">coursenotes</a>

</div><!--closes wrapper div-->
</body>
</html>
```

View the page which has these styles applied.

Screenshots

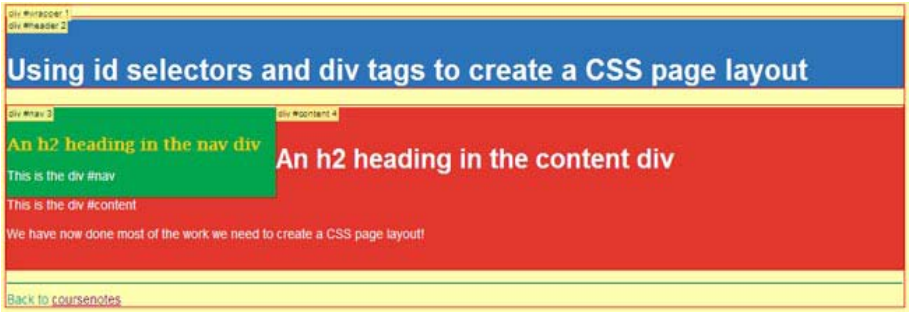
The following screenshots show how the styles are applied. View the screenshots and read the explanatory text, then have a go at creating your own version.

Screenshot one, below, shows the four divs created, but with no width, float or margin styles added. Notice that the divs are displayed in the order in which they are written in the HTML code.



Note: To see div names, and their HTML order number, select **Information > Display Div Order** in the Firefox web developer toolbar.

In **screenshot two**, below, width and float properties have been added to the #nav div. Notice how the the #content div moves up into the space left vacant. The background of the #content div slides underneath the #nav div, but its text content does not.



In **screenshot three**, below, we see the completed page layout. Adding the margin-left property to the #content has moved its background to the right of the #nav div, creating a neat space between the two page elements.

Create your own CSS page layout

To make your own page with CSS layout:

- create a new HTML document
- add content to identify the following page sections, and in the following order:
    1. header or banner section (for the website name and, later, logo)
    2. sidebar which will contain site-wide navigation links
    3. main content section
- surround each section of content with   tags
- assign an id attribute and value to each section's tag, for example
- ensure you have added the <link> tag to the head section of your HTML document to link it to your stylesheet
- save the HTML file as **layout.html**

- in the stylesheet you have already made, add the styles as shown above (but with more pleasing colour choices!) and save the file
- view your **layout.html** file in the browser

This simple page layout is reasonably easy to create, and is accessible to all users (unlike older-style table layouts). We can add additional styling to make the page look more pleasing (which we'll come to later on in the course), but the basic CSS layout is complete.

As you progress, you will come across highly sophisticated CSS page layouts which are complex to construct. The basic principles we have covered in this session lay the foundation for your understanding of such uses of CSS, and for your own further self-study in this exciting and creative area of web design.

For the purposes of this course, creating pages with a page layout similar to the one demonstrated here is perfectly adequate.

Adding a footer

To complete the page layout we shall add a footer. The footer section of a web page often contains secondary navigation links and links to utilities pages such as 'about us' and 'privacy policy.' Very deep footers have been fashionable for the last couple of years, often containing tangential information and social networking links.

In our example page layout the <hr> will be removed, replaced by a div with the id **footer**:

```
<!DOCTYPE HTML public "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/HTML4/loose.dtd">
<html>
<head>
<title>id selectors </title>
<meta http-equiv="content-type" content="text/HTML; charset=UTF-8">
<link href="css/styles.css" rel="stylesheet" type="text/css">
</head>
<body>

<div id="wrapper">

<div id="header">
<h1>Using id selectors to create a CSS page layout </h1>
```

```
</div><!--closes header div-->

<div id="nav">This is the div #nav</div><!--closes nav div-->

<div id="content">
<p>This is the div #content</p>
<p>We have now done most of the work we need to create a CSS page layout!</p>
</div><!--closes content div-->

<div id="footer">
Back to <a href="../selectors.html#back2">coursenotes</a>
</div><!--closes footer div-->
</div><!--closes wrapper div-->
</body>
</html>
```

The following style declaration is added to the stylesheet **styles.css**, which applies a solid, 1 pixel wide, red top border to the div #footer:

```
#footer {border-top: solid 1px #E3372E;
        }
```

More content is added to the left navigation div, and the resulting page looks like this:

Notice that the the height of the #nav div has increased due to its additional content.

As the #nav div has already been floated left, the #footer div moves up into the vacant space to the right of the #nav div, directly beneath the #content div.

This isn't what we want. The footer needs to be below all of the other page elements, to finish off the page neatly.

Clear property

To ensure the footer is pushed below any floated elements (either right or left floated) we add the clear:both property to the #footer declaration block:

```
#footer {border-top: solid 1px #E3372E;
         clear: both;
        }
```

This is how the page looks when clear:both is added to the style declaration:

Due to the clear property, the footer has had to leave a break after the floated element, and has taken its rightful place beneath the nav and content divs.

Min-height property

A div will increase in height to accommodate more content as it is added. Also, a fixed-width div will grow taller if a user increases the page's text size. For these reasons it is unusual to fix a div's height. However, we may not want to see a short div, such as the content div in the example above.

To ensure a div doesn't shrink below a specified height, we can add the min-height property to it, as in this example (the same min-height is also added to the #nav div):

```
#content {color: #FFF;
          background-color:#E3372E;
          margin-left:32%;
          min-height: 350px;
         }
```

Producing the following result:

The page layout is basically finished. To add further refinements, we need to understand the CSS Box Model, next.

The CSS box model

It is important to understand the **CSS box model** when applying styles.

CSS considers each HTML element to be a box, which consists of:

- content
- padding
- border
- margins

Each element can have its own border, padding and margin settings. The settings for each can be different for the top, right, bottom and left sides of the box.

The **box model** allows elements to be spaced in relation to other elements, have borders around them, with space between the content and the border.

The image below illustrates the box model:

Margin

Margins clear an area around the border, creating distance between the box and other boxes around it, or the edge of the div or page. Margins can be set for all sides at once, or individually for each side. Margins are transparent; the background colour will show through margins.

Border

The border sits between the margin and padding. It is transparent by default, but can be given a style, width and colour to make it visible, to define or decorate the boundaries of the box. As with margins, you can set borders for all sides at once, or for each side individually.

Padding

Padding clears an area between the inside edge of the border and the box's content. Padding takes the background color of the box. As with margins and borders, padding can be set for all sides at once, or for each side individually.

Content

The content of the box, where text and images appear

Width and Height of an Element

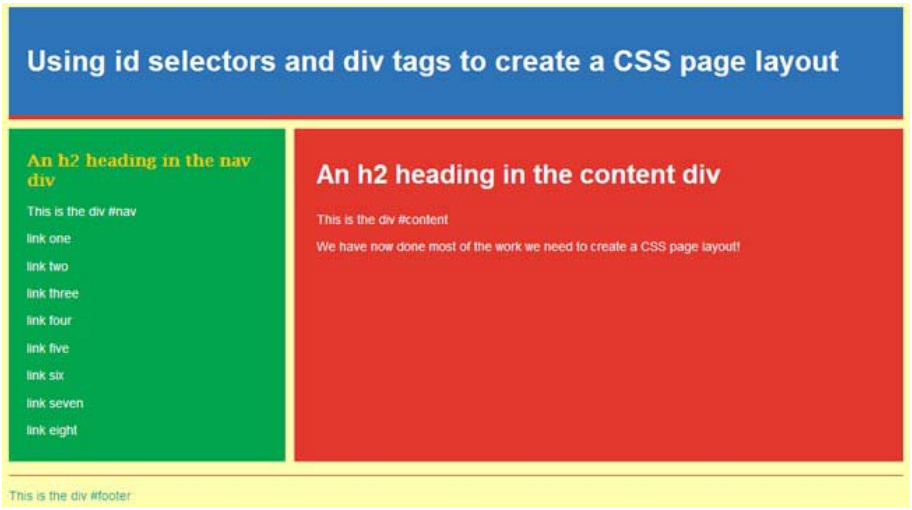
It is important to remember that the width and height specified for an element refers to the **content area only**. To calculate the full size of the element, you must add in the padding, border and margin values.

Adding padding, border and margins

Experiment with adding these properties to the divs in your CSS page layout.

Padding, border and margins are often specified using pixels (px) as the unit of measurement. As you have seen in our example, it is possible to set a margin value in percent (%).

Here's how our example page looks with some padding, margins and a border added:



To add padding to each side of the #nav div individually you could write each declaration individually:

```
#nav {color: #FFF;
        background-color:#00A54E;
        width: 28%:
        float: left;
        min-height: 350px;
        padding-top: 10px;
        padding-right: 10px;
```

```
 padding-bottom: 10px;
 padding-left: 20px;
}
```

Alternatively, you can write:

```
#nav {color: #FFF;
 background-color:#00A54E;
 width: 28%:
 float: left;
 min-height: 350px;
 padding: 10px 10px 10px 20px;
}
```

This shorter method sets padding for all four sides of the box, starting at the top, and working clockwise (top, right, bottom, left), just as the example above, but with less typing.

If values for the top and bottom padding are the same, and for left and right (but different from top and bottom), the declaration can be written:

```
#nav {color: #FFF;
 background-color:#00A54E;
 width: 28%:
 float: left;
 min-height: 350px;
 padding: 10px 20px;
}
```

The declaration above sets the padding top and bottom to 10px, and left and right padding to 20px.

If all four sides are to be the same (in this example, 10px), write this:

```
#nav {color: #FFF;
 background-color:#00A54E;
 width: 28%:
 float: left;
 min-height: 350px;
 padding: 10px;
}
```

This form of abbreviation works in the same way for setting margins.

Setting border properties can be long-winded if each border is to be styled differently, as you have options for style, width and colour. Here styles are set for top and right border (bottom and left borders could be added in the same way):

```
#nav {color: #FFF;
        background-color:#00A54E;
        width: 28%:
        float: left;
        min-height: 350px;
        padding: 10px;
        border-top-style: solid;
        border-top-width: 1px;
        border-top-color: #33C;
        border-right-style: dotted;
        border-right-width: 2px;
        border-right-color: #009B8F;
       }
```

These declarations can be shortened to:

```
#nav {color: #FFF;
        background-color:#00A54E;
        width: 28%:
        float: left;
        min-height: 350px;
        padding: 10px;
        border-top: solid 1px #33C;
        border-right: dotted 2px #009B8F;
       }
```

Note that there is a space but no punctuation between the style, width and colour values.

If all four borders are to have the same styling, the declaration can be written:

```
#nav {color: #FFF;
        background-color:#00A54E;
        width: 28%:
        float: left;
        min-height: 350px;
        padding: 10px;
        border: solid 1px #33C;
       }
```

Remember, you can see the complete list of CSS properties at:
www.w3schools.com/css/css_reference.asp

Session four summary

At the end of Session four you should know that:

- we can use CSS for page layout, using the tag
- we use the float property with s to create page layout
- the clear property is very useful to use in conjunction with float
- in CSS, each HTML element is treated as a box, having content, padding, borders and margins

You will have added layout styles to your external style sheet, and made a new page with a CSS layout including a header and two columns. You may also have experimented with adding padding, border and margins to your page elements.

Creating a CSS page layout is certainly an achievement: well done!

Self-study tasks

It is always a good idea to practise new skills: to reinforce what you've already learnt, and give you confidence to tackle the next new topic.

Here are my suggestions:

- make more pages using your basic CSS page layout and experiment with adding different colours, margins, padding and borders to your divs and other page elements