

Creating layouts with CSS

Note: the demos for these sections don't work in MS Word, and I didn't have time to take screenshots; to see these concepts in action, visit the CSS tutorial at www.sjhwebdesigns.com/resources/csstutorial/

Border, margin and padding

(demos for this section can be found at www.sjhwebdesigns.com/resources/csstutorial/bordermarginandpadding/)

borders

CSS can be used to add borders to HTML elements; you can also specify the border style, its width and its colour.

border-width

Sets the thickness of the border. You can use any unit for this, but it's usually best to stick to pixels (px). For example:

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

sets the border-width to 1 pixel around all paragraphs.

You can also set the border to have a different thickness on each side using multiple values.

`border-width: 1px 2px` - sets the top and bottom borders to 1px, and left and right borders to 2px.

`border-width: 1px 2px 3px 4px` - sets the top border to 1px, the right border to 2px, the bottom border to 3px and the left border to 4px. If you want to specify widths for all four sides, they should always be written in that order: top right bottom left. So the widths are specified in a **clockwise** direction from the top.

If this is too confusing, you could alternatively use the separate properties **border-left-width**, **border-top-width** etc. However, learning the shorthand can save a lot of time and space.

Note: to create a border, it is not enough to define its width! You will also need to specify a **border-style**:

border-style

Sets the style of the border. There are many different styles, such as solid, dashed and dotted. You can also set different styles for the different sides in the same way as for border-width, eg `border-style: solid dotted solid dashed`.

border-color

The colour of the border, which you can define using hex colour codes. Again, you can specify different colours for the different sides, eg

```
border-color: #a00 #0a0 #00a #aaa.
```

If this property is not set, the border will have the same colour as the element text (which can be specified using the **color** property).

If the width, style and colour is the same for all sides, you can specify them all within the **border** property, like this:

```
border: 1px solid #0a0;
```

You can also use **border-top**, **border-left** etc to set width, style and colour to the different sides:

```
border-bottom: 3px dashed;  
border-left: 1px solid #0a0;
```

margins

The margin is the space outside the element. Surrounding elements are pushed outside of this margin. Any unit can be used. For example:

```
p {  
    margin: 1em;  
}
```

sets a one em margin around every paragraph. As with borders, you can specify different margins for the different sides using shorthand:

```
margin: 1em 0 - sets top and bottom margins to 1em, left and right margins to 0
```

```
margin: 0 1em 2em 3em - sets the top margin to 0, right margin to 1em, bottom margin to 2em and left margin to 3em.
```

You can also use **margin-top**, **margin-left**, etc.

padding

Padding is the space in between the content of the element and its border. Again, any units can be used, and you can set different paddings on different sides in the same way as for margins.

Note: avoid applying vertical margins or padding to **inline** elements; surrounding elements will not respect the vertical margins, and will overflow under or on top of the vertical padding.

Browser inconsistencies

All graphical browsers apply a certain amount of padding and/or margin to many page elements by default; for example, list items are indented, and headings and paragraphs have small vertical margins. However, different browsers use different mixes of these properties, in different amounts, which results in pages displaying, well, differently.

To ensure consistency across browsers, we can use the **universal selector** - represented as *****. This selector specifies **every** element within the page. So we can get rid of all default margins and padding like this:

```
* {  
    margin: 0;  
    padding: 0;  
}
```

Now we can specify margins and/or padding for individual elements, and the result will be (pretty much) the same in all browsers:

```
h1, h2, h3, h4, h5, h6 {
    margin: 1em 0 0.5em 0;
}
p {
    margin: 1em 0;
}
li {
    margin-left: 1em;
}
```

And so on for other elements. While this is somewhat laborious, it is the only way to be sure of consistent results, and once you have a set of default margin/padding styles, you can reuse it on any other sites you create.

Boxes

(demos for this section can be found at www.sjhwebdesigns.com/resources/csstutorial/boxes/)

width and height

An element can be given a specific width and height by using the **width** and **height** properties (surprise!). These properties only work on **block-level** elements (but see below). For example:

```
#content {
    width: 760px;
}
```

Avoid applying a width **and** height to an element unless you're sure how much space its content is going to take up. If the element content is larger than can fit in the specified width and height, the content will overflow its container.

At least it does in standards-compliant browsers. Versions of Internet Explorer prior to 7 will assume that you don't know what you're doing, and will extend the box to fit its content. This may seem like a help at first, but allowing content to overflow its container can actually be very useful, and also prevents the page layout from breaking on larger text sizes. IE7, meanwhile, actually hides much of the overflowing content. Hmmm.

The box model

When applying borders, margins and padding to an element with defined dimensions, you need to be aware of the **box model**, which defines how page elements should be displayed by graphical browsers. The box model states that the width and height specified for an element will define the size of the **content area** for that element; padding goes outside of this content area, borders go outside the padding, and margins go outside the border. What is important to remember is that any padding, borders or margins are in **addition** to the width specified for the box. So a box with a width of **80px** and padding of **40px** on both sides would have a total width of **160px**; adding a **40px** margin and a **40px** border would increase the total width to **320px**.

Display

(demos for this section can be found at www.sjhwebdesigns.com/resources/csstutorial/display/)

Maybe you want to apply width and height, and/or vertical margins and padding, to an inline element - or maybe you want to make multiple block-level elements sit side by side rather than taking up a whole line to themselves. You can achieve this by using the **display** property. This lets you change inline elements to display like block elements, and vice versa.

Applying **display: block** to an inline element will make it clear a space before and after it, and will allow you to define dimensions, and vertical margin/padding to it - just like a block element. Remember that this is purely presentational - it does not allow you to place block elements within that inline element.

Applying **display: inline** to a block element will disable any dimensions, or vertical margins/padding. The element will then only take up the space occupied by its content. One application of this is creating horizontal lists.

You can also make an element disappear altogether by applying **display: none** to it. This can allow you to show or hide different elements when the user interacts with the page.

Note: the **display** property has many other possible values; however, **block**, **inline** and **none** are the only ones widely supported by current browsers.

Positioning and floating

(demos for this section can be found at www.sjhwebdesigns.com/resources/csstutorial/positioningandfloating/)

By now we've achieved a great deal of control over the appearance of our page elements, but our pages are still stuck in a linear layout - each element simply follows after the next in the same order as they are written in the HTML. However, CSS allows us to create sophisticated and versatile page layouts using two properties: **position** and **float**.

position

The **position** property controls how an element is positioned on the page. It can take the following values:

static: the element stays within the normal flow of the document: it appears on the page according to where it is placed in the HTML, and surrounding elements clear a space for it. This is the default behaviour.

relative: this keeps the element within the flow of the document, but it can be shifted relative to its normal position by specifying a horizontal or vertical distance, using any of the properties **top**, **left**, **bottom** and **right**.

The surrounding elements still clear a space for the relatively positioned element, *but they clear the space that it would have occupied before it was moved*. This means that it can overlap other elements if you shift it more.

Note: when positioning elements, you can use **top** or **bottom**, but not both. Similarly with **left** or **right**.

absolute: this takes the element out of the normal flow of the document, and surrounding elements will behave as if it wasn't there. The location of absolutely positioned elements can be specified using the properties **top**, **left**, **bottom** and **right**.

If the absolutely positioned element is inside an element which is positioned absolutely or relatively, it will be positioned relative to this container. Otherwise, it will be positioned relative to the page itself.

fixed: similar to absolute, except that the element is positioned relative to the browser window, and so stays in the same place, even when the user scrolls up and down the page. Very cool. Not supported in Internet Explorer 6 and earlier. 😞

While absolute and relative positioning can be very useful in laying out page elements, they also run the risk of causing elements to be plonked on top of each other. This can be avoided by setting margins for the surrounding elements.

float

Floating allows you to shift elements to the left or right of their container. Surrounding elements will then flow around them.

clear

The **clear** property prevents elements from flowing around floated boxes. It can take the following values:

- **left** - the element clears any left-floated boxes
- **right** - the element clears any right-floated boxes
- **both** - the element clears any floated boxes
- **none** - the element flows around floated boxes. This is the default behaviour.

Unfortunately, many of Internet Explorer's most bizarre and inexplicable bugs seem to reveal themselves in more complex floated layouts, and several of these bugs have not been fixed in version 7: elements within or in the vicinity of floats can reposition themselves, disappear entirely, jump to the left when a neighbouring element is hovered over, or otherwise go wrong in many other weird and wonderful ways. For this reason, you should always make sure to test any floated layouts in IE as well as a standards-compliant browser.