S@ifur's Solutions

69/B Monwara Plaza (Level-5)
East Panthapath
Dhaka-1205
Cell: 01713 432003

Lecture Sheet on "PHP+MySQL"
Day-3: (CSS)

Designing with Style Sheets: With CSS, it is possible to completely change the way elements are presented by a user agent. This can be executed at a basic level with the display property, and in a different way by associating style sheets with a document. The user will never know whether this is done via an external or embedded style sheet, or even with an inline style. The real importance of external style sheets is the way in which they allow authors to put all of a site's presentation information in one place, and point all of the documents to that place. This not only makes site updates and maintenance a breeze, but it helps to save bandwidth since all of the presentation is removed from documents.
To make the most of the power of CSS, authors need to know how to associate a set of styles with the elements in a document. To fully understand how CSS can do all of this, authors need a firm grasp of the way CSS selects pieces of a document for styling, which is the subject of the next chapter.
Basic Rules: As I've stated, a central feature of CSS is its ability to apply certain rules to an entire set of element types in a document. For example, let's say that you want to make the text of all H2 elements appear gray. Using old-school HTML, you'd have to do this by inserting <FONT COLOR="gray">...</FONT> tags in all your h2 elements:

```
<h2><font color="gray">This is h2 text</font></h2>
```

Obviously, this is a tedious process if your document contains a lot of h2 elements. Worse, if you later decide that you want all those H2s to be green instead of gray, you'd have to start the manual tagging all over again. CSS allows you to create rules that are simple to change, edit, and apply to all the text elements you define (the next section will explain how these rules work). For example, simply write this rule once to make all your h2 elements gray:

```
h2 {color: gray;}
```

If you want to change all H2 text to another color say, silver simply alter the rule:

```
h2 {color: silver;}
```

Rule Structure:
To illustrate the concept of rules in more detail, let's break down the structure. Each rule has two fundamental parts, the selector and the declaration block. The declaration block is composed of one or more declarations, and each declaration is a pairing of a property and a value. Every style sheet is made up of a series of rules. Figure 2-1 shows the parts of a rule.

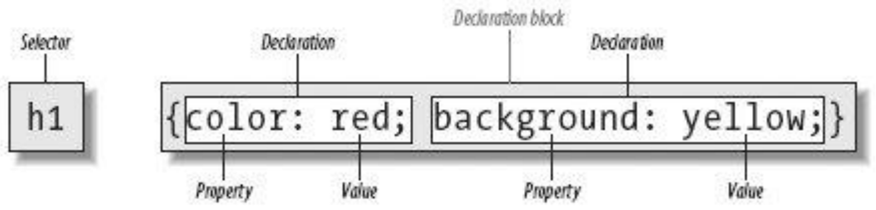


Figure 2-1 The Structure of a ruleThe selector, shown on the left side of the rule, defines which piece of the document will be affected. In Figure 2-1, `h1` elements are selected. If the selector were `p`, then all `p` (paragraph) elements would be selected. The right side of the rule contains the declaration block, which is made up of one or more declarations. Each declaration is a combination

of a CSS property and a value of that property. In Figure 2-1, the declaration block contains two declarations. The first states that this rule will cause parts of the document to have a `color` of `red`, and the second states that part of the document will have a `background` of `yellow`. So, all of the `h1` elements in the document (defined by the selector) will be styled in red text with a yellow background.

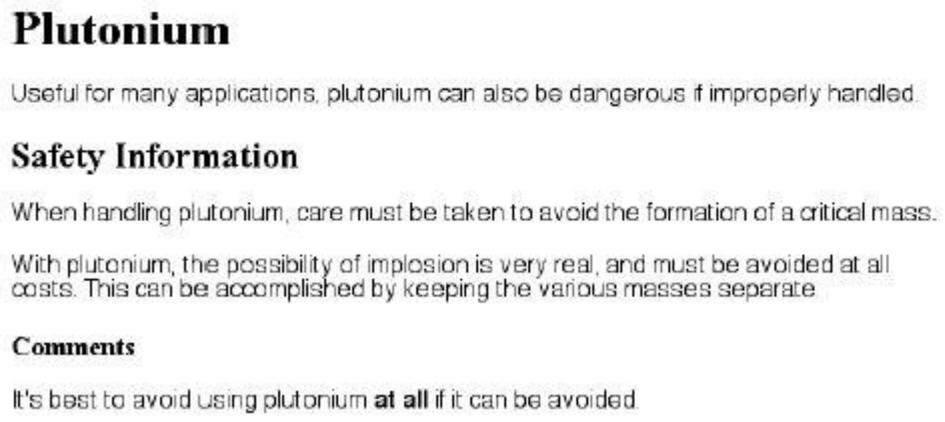
Element Selectors: A selector is most often an HTML element, but not always. For example, if a CSS file contains styles for an XML document, a selector might look something like this:

```
QUOTE {color: gray;}
BIB {color: red;}
BOOKTITLE {color: purple;}
MYElement {color: red;}
```

In other words, the elements of the document serve as the most basic selectors. In XML, a selector could be anything, since XML allows for the creation of new markup languages that can have just about anything as an element name. If you're styling an HTML document, on the other hand, the selector will generally be one of the many HTML elements such as `p`, `h3`, `em`, `a`, or even `html` itself. For example:

```
html {color: black;}
h1 {color: gray;}
h2 {color: silver;}
```

The results of this style sheet are shown in Figure 2-2.



Plutonium

Useful for many applications, plutonium can also be dangerous if improperly handled.

Safety Information

When handling plutonium, care must be taken to avoid the formation of a critical mass.

With plutonium, the possibility of implosion is very real, and must be avoided at all costs. This can be accomplished by keeping the various masses separate.

Comments

It's best to avoid using plutonium **at all** if it can be avoided.

Figure 2-2 Simple styling of a simple document

Structuring pages with Tables versus Divs: the question is not really "CSS or no CSS" — who wants to go back to font tags, and hand-editing every page? It's whether, for layout purposes, you want to encase your content in tables (which is easier if you're used to it) or divs (which are tricky at first but offer certain freedoms and lighter code). So, where you see questions like "Should I use CSS or tables?", what they're usually asking is "CSS and tables or CSS and divs?".*

* Side note: For years, the argument was that tables were never "intended" to be used for web page layout (who cares?); unfortunately for those who do care about this argument, someone who was present when tables were introduced says that layout was one of the uses suggested for tables. Anyway, this particular discussion gets rather heated with a fervor that is something to behold. As well, since 1998, the Web Standards Project (webstandards.org) was founded to get browser makers to follow Web Standards (agreed upon by browser makers at the W3C.org meetings) so that we can rely on standard coding to give us standard display — because current browsers don't always display the same. They've gone a good way towards this ideal, although Microsoft is still playing silly with some of it.

Now, while you've learned to work with tables, you've illustrated one excellent argument for using divs/CSS: display for multiple devices, such as a Blackberry. It's thought that the various browsers on dozens of models cell phones, Blackberries, etc. may not display table-based sites properly — and you've demonstrated for yourself that that's true. Given the history of desktop browsers, I'd recommend divs and CSS, as that's where the industry is going anyway.

Lastly, do yourself a favor and write your CSS in a separate page (which makes it an "external stylesheet"), and link to it from within the <head> area of your pages, so that updates require editing one file, not every page. And test/design in Firefox or Opera, as they're more standard than IE (which is buggy and has display issues), then fix your code for IE.

IDs, Classes and HTML tags: CSS styles cascade down the CSS page, meaning that if you define something, and then redefine it later in the CSS page, the latter styling will apply. However, CSS also has two levels in addition to the ability to define how any HTML tag displays:

- any HTML tag can also be styled (e.g., body, h1, p, etc.)
- ID (id="something") which can only be used *once on any page* and has more "weight" than (can override) classes
- class (class="something") which can be used multiple times on a page

body <= an HTML tag
#pagewrap <= an ID; can be used only once on a page
.someclass <= a class; can be used multiple times on page
I use IDs for main layout elements (header, footer, navbar, content) so that they're not inadvertently out-weighed by a class. (Note: while you can name your IDs and classes anything you like, I'd recommend taking care in doing so, or you'll be updating them later when they drive you nuts trying to figure out what you applied them to.)

Example Stylesheet:

body {text-align:center; font-family:arial,helvetica,sans-serif} <= centers everything on the page that falls within the **<body>** and **</body>** tags, which is everything

#pagewrap {width:780px; text-align:left} <= now we need to center-align the "page" but left-align the content. If you're not setting a page width, I don't think you'll need this.

h1 {color:#f000, font-size:23px}

Floats: CSS/divs relies heavily on what is called a "float" ... meaning you can align or "float" something to the left or right or where it would be if you didn't float it:

```
<div style="float:left">sdfd</div>
<img src="someimage.jpg" style="float:right">
```

However, stuff later on the page *may* still be affected by the float, so you have to clear it from what went on before (you can clear:left, clear:right, or clear:both):

```
<div style="float:left">sdfd</div>
<img src="someimage.jpg" style="float:right">
<p style="clear:both">
```

Let's try it: Now, without writing an entire stylesheet, let me illustrate. Here's code starting right after the **body** tag (remember that we've already made the page center in the body tag):

```
<div id="pagewrap">
<div id="logo">logo and whatever here</div>

<div id="contentarea">
<div float me left; assign a width> Here's my content</div>
<div float me right; assign a width> Here's my navbar</div>
<div clear:both>footer</div>

</div> <!-- closes #contentarea -->
</div> <!-- closes #pagewrap -->
```

Columns with background colors: lastly (really), divs are generally only as tall as their content, which means that (for example) a navbar may not be as tall as the content on a particular page — which is okay if you *don't* want a background color on either one. With divs, if you want (for example) a navbar with a background color that runs to the bottom of the content, make a background *image* for that part of the page (e.g., the whole content/navbar); problem solved.

Create a Tableless Layout

I bet you've heard the coding obsessed web designers getting their knickers in a twist about CSS layouts, yet you don't know the first thing about them, right? Obviously that's why you're here! Well, providing you have a little CSS knowledge and know how to separate style from content, you should find this lesson a doodle!

Tableless Layout Basics

Firstly, let's take a look at your current coding. I bet it probably looks a little like this:

```
<html>
  <head>
    <title> my website </title>
  <meta name="keywords" content="joe bloggs' website, site, pages,">
  <meta name="author" content="joe bloggs">
  <link href="stylesheet.css" rel="stylesheet" type="text/css">
  </head>
  <body>
<table id="main">
<tr><td>
    <img src="images/bla.gif" align="left" alt="layout image 1"><br />
</td></tr>
<tr><td>
  <h1>Welcome to my website</h1>
    <p>This is my wonderful website with funky content and groovy images!</p>
</td></tr>
</table>
```

```
  </body>
</html>
```

Look at those terrible tables. Messy or what?! Anyway, <table>s are designed for tabular data — stuff like numbers and statistics, not for presentation of layouts. This means we have to find an alternative: <div>s — a highly adaptive block-element tag which can work just as well as tables.

A <div> works like most other HTML tags — most importantly you can give it an ID and assign attributes to it using CSS (in fact, this is the key to tableless layouts). <div>s can go inside other <div>s, can be floated, resized, recoloured and positioned. If you can do it to a table, you can do it to a <div>.

Let's redo that page up there, assigning <div>s instead of <table>s (every 'block' of table needs a div):

```
<html>
  <head>
    <title> my website </title>
  <meta name="keywords" content="joe bloggs' website, site, pages,">
  <meta name="author" content="joe bloggs">
  <link href="stylesheet.css" rel="stylesheet" type="text/css">
  </head>
  <body>
<div id="image">
    <img src="images/bla.gif" align="left" alt="layout image 1"><br />
</div>
<div id="main">
  <h1>Welcome to my website</h1>
    <p>This is my wonderful website with funky content and groovy images!</p>
</div>
  </body>
</html>
```

You can now customise those <div>s using normal CSS. As we've given the <div>s in the above example IDs, we customise them like so:

```
#image {
float: left;
width: 500px;
}
```

..where *image* is the name we gave one of the <div>s in the example above. If we had assigned the <div> a class instead of an id, this is how we'd customise it:

```
.image {
float: left;
width: 500px;
}
```

There's not much difference between the two, so be sure to assign the right thing. Now that is the basics of creating a tableless layout — using <div>s and then customising the <div>s with CSS. Now, onto actually creating CSS only templates...

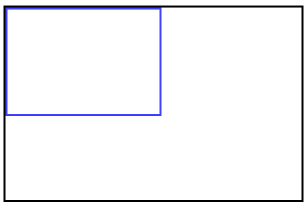
Tableless Templates

So, you've figured out that <div>s are the magic boxes of the CSS world, now what can you do with them? Tons! Although most n00bs would try and make you believe that there's things you can't do with <div>s, they're not giving you the whole truth. For example, let's look at a common table-based layout:

header image	
content column	linkscolumn

A lot of people think you cannot replicate this with CSS. Try coding up three <div>s in an example page and see what you get; anything like that example above? No. This is where you need to learn about an integral part of CSS based layouts: floats.

Firstly, one thing you need to realise about floats is that once they're assigned, the element you've assigned them to will 'stick' to the furthest side they can reach. For example, look at the example below. It is a <div> (represented in blue) floated to the left of a page/browser (black) with no other elements.



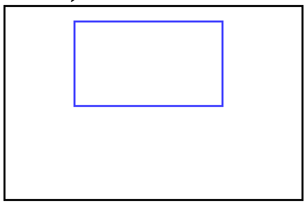
If you've got a fancy pants layout set up in/with that <div>, you probably won't want it resting again against the left hand side of the browser. Even if you use the centering cheat the <div> will stick to that left hand side — this is where a container <div> comes in handy. A container <div> is just a normal <div>, but you assign the properties which position the main 'bits' to this <div>. First, let's stick the container in the page, between the <body> tags:

```
  ..etc..
  <body>
<div id="container">
</div>
  </body>
  ..etc..
```

Now we just customise it in out stylesheet like we would a normal <div> (the code below demonstrates an example of the centering cheat; this would center the container):

```
body {
text-align: center; /* center things in pre-IE6 */
}
#container {
margin: 35px auto;
width: 500px;
text-align: left;
}
```

Our page should now look a bit like this; the container is represented in blue:



You're probably thinking something like *"well that's all fine and dandy, but it doesn't look anything like the table layout at the top of this page!"* — this is where we start adding more <div>s along with some handy floats. The first <div> we add will need to replace the header. This is the easiest part, we just add a <div> inside the container, and give it an id of "header" (at the moment, just so we know which is which):

```
  ..etc..
  <body>
<div id="container">
    <div id="header">    </div>
</div>
  </body>
  ..etc..
```

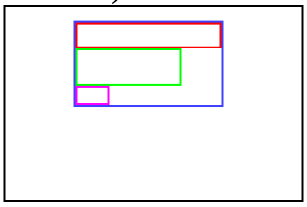
In this header, you'd probably stick your top image or any fiddly text you want to flash up your style. That's the easiest bit, and doesn't require any floats or special properties. Now for the columns:

```
  ..etc..
  <body>
<div id="container">
    <div id="header">    </div>
    <div id="content">  blabla content  </div>
    <div id="sidebar">  blabla links  </div>
</div>
  </body>
  ..etc..
```

First decide how big you want this content column: as I set the container to 500 pixels above, 380 pixels would be a sufficient width. This leaves 120 pixels for the sidebar (links) column. Let's specify these in our stylesheet (I've left the other stuff in so you know where we're up to):

```
#container {
margin: 35px auto;
width: 500px;
}
#header {
width: auto;
}
#content {
width: 380px;
}
#sidebar {
```

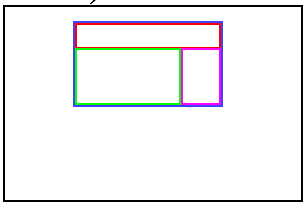
width: 120px;
}
However, even with these <div>s declared, your page will still look something like this (red: header / green: content / pink: sidebar):



We need to give our columns some floats. As the content column is on the left, it'll need a left float. The sidebar column is on the right, so that'll need a right float. We assign these using CSS like so:

```
#content {
float: left;
width: 380px;
}
#sidebar {
float: right;
width: 120px;
}
```

That's pretty much it. Your <div>s should now be correctly positioned, and should look a bit like this (without all the pretty colours):



The two columns should float against the correct sides of the container side-by-side without problem, as long as the total of both columns does **not** exceed the width of the container. That means if you assign padding, margin or a border to either of the columns, you must subtract that total number from the width of the column you've assigned it to. If you're still a bit clueless and need a 'live' example, you can download a basic header and two-column template for testing and inspiration.

Help! My container background is not appearing properly?

Certain browsers will not show the background behind floated divs unless you 'clear' them. To get past this little problem we need to add another div before we close our container, a footer div:

```
  ..etc..
    <div id="sidebar">  blabla links  </div>
    <div id="footer">    </div>
</div>
  </body>
  ..etc..
```

Then we clear it in our stylesheet, by adding this to the bottom:

```
#footer {
clear: both;
}
```

As simple as that!