Lab 8. 2. Layout with CSS

Prepared: TrangNTT

Lab 8. 2. Layout with CSS 1
8.1. Layout with style 1
8.2. Layout with DIV tag 8
8.3. Exercise 1: Modify 8.1 using DIV tag 21
8.4. Exercise 2: CSS Box Model 21

8.1. Layout with style

Step 1. Writing HTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
<head>
  <title>My first styled page</title>
</head>

<body>

<!-- Site navigation menu -->
<ul class="navbar">
  <li><a href="index.html">Home page</a>
  <li><a href="musings.html">Musings</a>
  <li><a href="town.html">My town</a>
  <li><a href="links.html">Links</a>
</ul>

<!-- Main content -->
<h1>My first styled page</h1>

<p>Welcome to my styled page!

<p>It lacks images, but at least it has style. And it has links, even if they
don't go anywhere&hellip;

<p>There should be more here, but I don't know what yet.

<!-- Sign and date the page, it's only polite! -->
<address>Made 15 October 2009<br>
  by myself.</address>

</body>
</html>
```

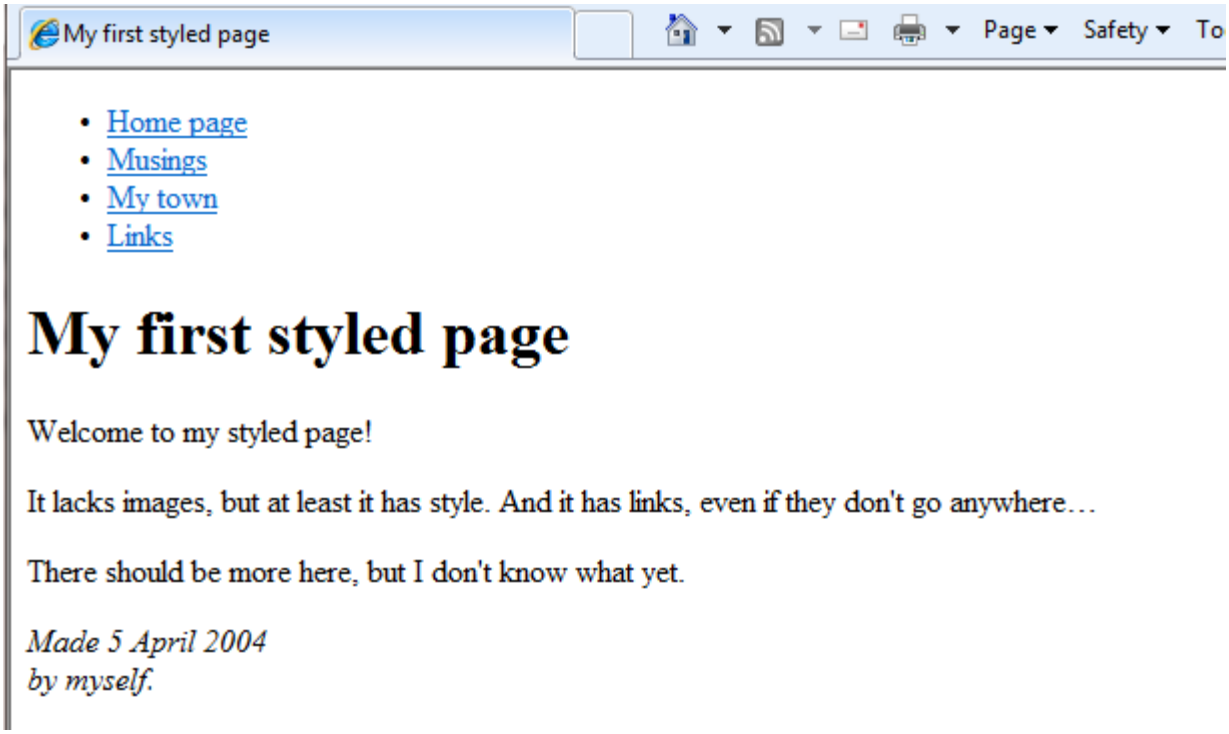
Let's assume that this is going to be one page of a Web site with several similar pages. As is common for current Web pages, this one has a menu that links to other pages on the hypothetical site, some unique content and a signature.

Now select “Save As…” from the File menu, navigates to a directory/folder where you want to put it and save the file as “mypage.html”. Don't close the editor yet, we will need it again.

(If you are using TextEdit on Mac OS X before version 10.4, you will see an option “Don't append the .txt extension” in the “Save as” dialog. Select that option, because the name “mypage.html” already includes an extension. Newer versions of TextEdit will notice the .html extension automatically).

Next, open the file in a browser. You can do that as follows: find the file with your file manager (Windows Explorer, Finder or Konqueror) and click or double click the “mypage.html” file. It should open in your default Web browser. (If it does not, open your browser and drag the file to it).

As you can see, the page looks rather boring…



Step 2. Adding some colors

You probably see some black text on a white background, but it depends on how the browser is configured. So one easy thing we can do to make the page more stylish is to add some colors. (Leave the browser open, we will use it again later.)

We will start with a style sheet embedded inside the HTML file. Later, we will put the HTML and the CSS in separate files. Separate files are good, since it makes it easier to use the same style sheet for multiple HTML files: you only have to write the style sheet once. But for this step, we just keep everything in one file.

We need to add a <style> element to the HTML file. The style sheet will be inside that element. So go back to the editor window and add the following five lines in the head part of the HTML file. The lines to add are shown in red (lines 5 to 9).

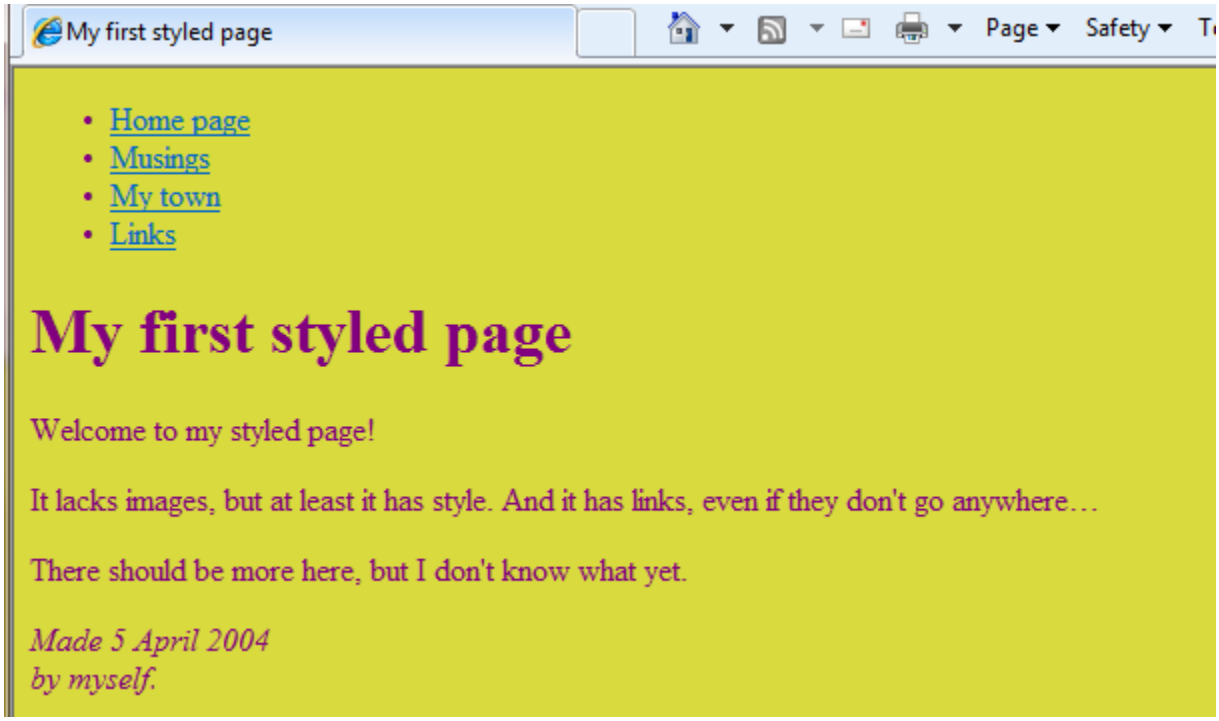
```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
<head>
  <title>My first styled page</title>
  <style type="text/css">
    <!--
        body {
                color: purple;
                background-color: #d8da3d
        }
    -->
  </style>
</head>

<body>
[etc.]
```

The first line says that this is a style sheet and that it is written in CSS (“text/css”). The second line says that we add style to the “body” element. The third line sets the color of the text to purple and the next line sets the background to a sort of greenish yellow.

The background of the body element will also be the background of the whole document. We haven't given any of the other elements (p, li, address…) any explicit background, so by default they will have none (or: will be transparent). The 'color' property sets the color of the text for the body element, but all other elements inside the body inherit that color, unless explicitly overridden. (We will add some other colors later.)

Now save this file (use “Save” from the File menu) and go back to the browser window. If you press the “Refresh” button, the display should change from the “boring” page to a colored (but still rather boring)

page. Apart from the list of links at the top, the text should now be purple against a greenish yellow background.



Step 3. Adding fonts

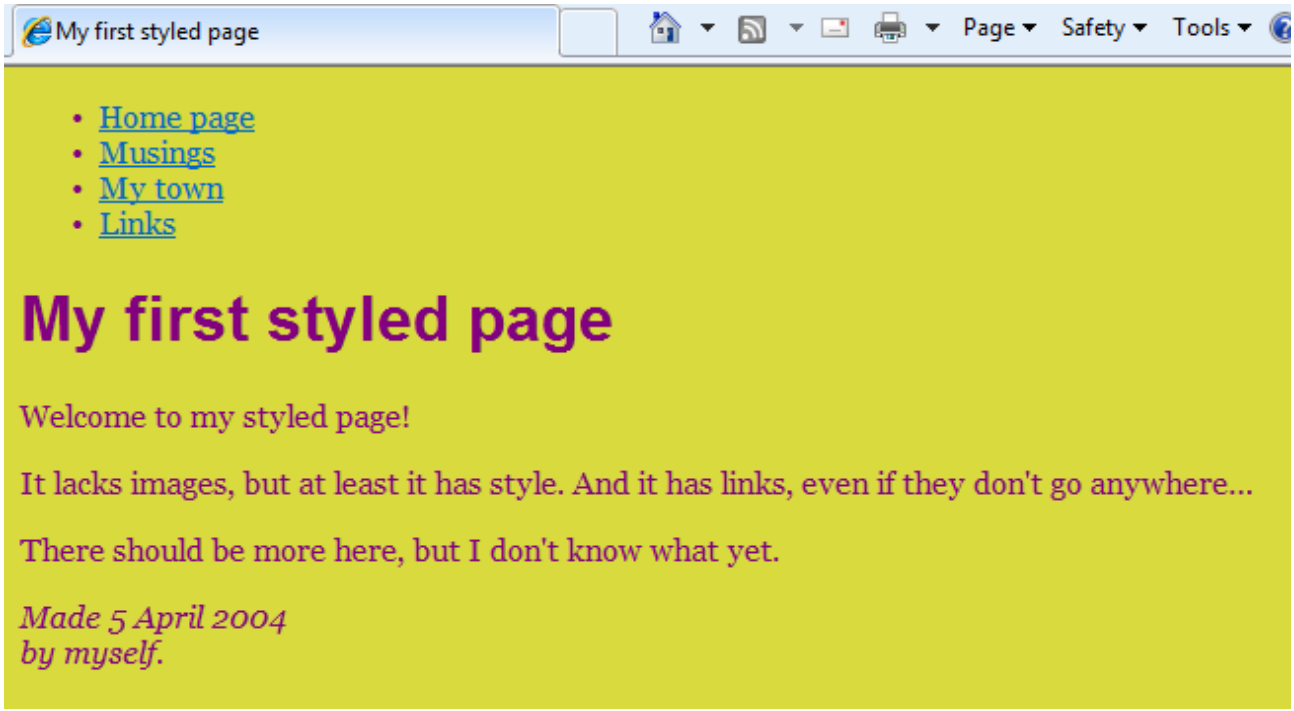
Another thing that is easy to do is to make some distinction in the fonts for the various elements of the page. So let's set the text in the “Georgia” font, except for the h1 heading, which we'll give “Helvetica.”

On the Web, you can never be sure what fonts your readers have on their computers, so we add some alternatives as well: if Georgia is not available, Times New Roman or Times are also fine, and if all else fails, the browser may use any other font with serifs. If Helvetica is absent, Geneva, Arial and SunSans-Regular are quite similar in shape, and if none of these work, the browser can choose any other font that is serif-less.

In the text editor add the following lines (lines 7-8 and 11-13):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
<head>
  <title>My first styled page</title>
  <style type="text/css">
  body {
    font-family: Georgia, "Times New Roman",
          Times, serif;
    color: purple;
    background-color: #d8da3d }
  h1 {
    font-family: Helvetica, Geneva, Arial,
          SunSans-Regular, sans-serif }
  </style>
</head>

<body>
[etc.]
```

If you save the file again and press “Refresh” in the browser, there should now be different fonts for the heading and the other text.



Step 4. Adding a navigation bar

The list at the top of the HTML page is meant to become a navigation menu. Many Web sites have some sort of menu along the top or on the side of the page and this page should have one as well. We will put it on the left side, because that is a little more interesting than at the top…

The menu is already in the HTML page. It is the <ul> list at the top. The links in it don't work, since our "Web site" so far consists of only one page, but that doesn't matter now. On a real Web site, there should not be any broken links, of course.

So we need to move the list to the left and move the rest of the text a little to the right, to make room for it. The CSS properties we use for that are 'padding-left' (to move the body text) and 'position', 'left' and 'top' (to move the menu).

In the editor window, add the following lines to the HTML file (lines 7 and 12-16):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
<head>
  <title>My first styled page</title>
  <style type="text/css">
  body {
    padding-left: 11em;
    font-family: Georgia, "Times New Roman",
          Times, serif;
    color: purple;
    background-color: #d8da3d }
  ul.navbar {
    position: absolute;
    top: 2em;
    left: 1em;
    width: 9em }
  h1 {
    font-family: Helvetica, Geneva, Arial,
          SunSans-Regular, sans-serif }
  </style>
</head>

<body>
[etc.]
```

If you save the file again and reload it in the browser, you should now have the list of links to the left of the main text. That already looks much more interesting, doesn't it?

Step 5. Styling the links

The navigation menu still looks like a list, instead of a menu. Let's add some style to it. We'll remove the list bullet and move the items to the left, to where the bullet was. We'll also give each item its own white background and a black square. (Why? No particular reason, just because we can.)

We also haven't said what the colors of the links should be, so let's add that as well: blue for links that the user hasn't seen yet and purple for links already visited (lines 13-15 and 23-33):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
<head>
  <title>My first styled page</title>
  <style type="text/css">
  body {
    padding-left: 11em;
    font-family: Georgia, "Times New Roman",
          Times, serif;
    color: purple;
    background-color: #d8da3d }
  ul.navbar {
    list-style-type: none;
    padding: 0;
    margin: 0;
    position: absolute;
    top: 2em;
    left: 1em;
    width: 9em }
  h1 {
    font-family: Helvetica, Geneva, Arial,
          SunSans-Regular, sans-serif }
  ul.navbar li {
    background: white;
    margin: 0.5em 0;
    padding: 0.3em;
    border-right: 1em solid black }
  ul.navbar a {
    text-decoration: none }
  a:link {
    color: blue }
  a:visited {
    color: purple }
  </style>
</head>

<body>
[etc.]
```

If you save the file again and reload it in the browser, you should now have a real menu:

Step 6. Adding a horrizontal line

The final addition to the style sheet is a horizontal rule to separate the text from the signature at the bottom. We will use 'border-top' to add a dotted line above the <address> element (lines 34-37):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
<head>
  <title>My first styled page</title>
  <style type="text/css">
  body {
    padding-left: 11em;
    font-family: Georgia, "Times New Roman",
          Times, serif;
    color: purple;
    background-color: #d8da3d }
  ul.navbar {
    list-style-type: none;
    padding: 0;
    margin: 0;
    position: absolute;
    top: 2em;
    left: 1em;
    width: 9em }
  h1 {
    font-family: Helvetica, Geneva, Arial,
          SunSans-Regular, sans-serif }
  ul.navbar li {
    background: white;
    margin: 0.5em 0;
    padding: 0.3em;
    border-right: 1em solid black }
  ul.navbar a {
    text-decoration: none }
  a:link {
    color: blue }
  a:visited {
    color: purple }
  address {
    margin-top: 1em;
    padding-top: 1em;
    border-top: thin dotted }
  </style>
</head>

<body>
[etc.]
```

Now our style is complete. Next, let's look at how we can put the style sheet in a separate file, so that other pages can share the same style.

Step 7. Putting the style sheet in a separate file

We now have an HTML file with an embedded style sheet. But if our site grows we probably want many pages to share the same style. There is a better method than copying the style sheet into every page: if we put the style sheet in a separate file, all pages can point to it.

To make a style sheet file, we need to create another empty text file. You can choose “New” from the File menu in the editor, to create an empty window. (If you are using TextEdit, don't forget to make it plain text again, using the Format menu.)

Then cut and paste everything that is inside the <style> element from the HTML file into the new window. Don't copy the <style> and </style> themselves. They belong to HTML, not to CSS. In the new editor window, you should now have the complete style sheet:

```
body {
  padding-left: 11em;
  font-family: Georgia, "Times New Roman",
        Times, serif;
  color: purple;
  background-color: #d8da3d }
ul.navbar {
  list-style-type: none;
  padding: 0;
  margin: 0;
  position: absolute;
  top: 2em;
  left: 1em;
  width: 9em }
h1 {
  font-family: Helvetica, Geneva, Arial,
        SunSans-Regular, sans-serif }
ul.navbar li {
  background: white;
  margin: 0.5em 0;
  padding: 0.3em;
  border-right: 1em solid black }
ul.navbar a {
  text-decoration: none }
a:link {
  color: blue }
a:visited {
  color: purple }
address {
  margin-top: 1em;
  padding-top: 1em;
  border-top: thin dotted }
```

Choose “Save As…” from the File menu, make sure that you are in the same directory/folder as the mypage.html file, and save the style sheet as “mystyle.css”.

Now go back to the window with the HTML code. Remove everything from the <style> tag up to and including the </style> tag and replace it with a <link> element, as follows (line 5):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
<head>
  <title>My first styled page</title>
  <link rel="stylesheet" href="mystyle.css">
</head>

<body>
[etc.]
```

This will tell the browser that the style sheet is found in the file called “mystyle.css” and since no directory is mentioned, the browser will look in the same directory where it found the HTML file.

If you save the HTML file and reload it in the browser, you should see no change: the page is still styled the same way, but now the style comes from an external file.

8.2. Layout with DIV tag

Most Web page designers use a table-based layout to achieve a coherent and consistent look. There's a different way to achieve the same look. Using CSS and DIV tags reduces markup code, speeds up page downloads, separates content from its visual presentation, and brings your code closer to Web standards compliance--all while making your website more appealing to search engine spiders. Alejandro Gervasio explains how it's done, with copious examples.

Step 1. Introduction

We see it every day. As we surf the huge network of the Web, we are able to appreciate several page layouts through different websites. They run the gamut from classical but highly effective two and three-column designs, to those exotic and certainly uncommon designs which seem not to fall into a specific category.

For many years, Web designers have been sticking firmly to table-based layouts, achieving coherent and consistent looks by the use of complex table nesting techniques, gif spacers and other well-known design processes. Currently, table layout still continues to have a strong presence on the Web, since it provides wide cross-browser compatibility. Tables are pretty well supported elements for browsers, and that’s an extremely powerful reason to use them for Web page layouts. But this strong compatibility comes with an extra cost. Most of the time, nested tables introduce lots of additional markup, and increase file size, download times and server requests when using graphics as spacers, even if they are cached later.

Extra HTML pushes the important content farther down in the source code, making it less likely that a search spider will give Web pages a high relevance score for the keywords. With the use of CSS and DIV tags, we can achieve the same table-based layout effects, reduce our markup code noticeably, get faster page downloads, and separate content from its visual presentation. We are getting closer to standards-compliant code, and our Web pages are more appealing to search engine spiders.

Now, is everything good about CSS? No, unfortunately. The major drawback with CSS is that browsers don’t display reliably some CSS rules because most of them are not completely compliant with Web standards. Still, the advantages of using CSS and DIV tags for page layout are numerous, and certainly are appealing to many Web designers, as modern browsers add more reliable support.

With the pros and cons in our hands, let’s see the basics of how table design works.

Step 2. The table-based approach

One of the most common page layouts using tables is easy to do with the ALIGN attribute. This is a left-hand navigation bar with a larger content section, directly to the right of it, building only two tables.

Here is the code:

```
<table align=”left” width=”20%” height=”100%”>
<tr>
<td valign=”top”>Navigation links are placed here</td>
</tr>
</table>

<table align=”right” width=”80%” height=”100%”>
<tr>
<td valign=”top”>Main content is placed here</td>
</tr>
</table>
```

We build this simple two-column layout, placing the tables as mentioned above: one is located to the left and the other is located to the right. The ALIGN attribute helps us to turn complex tables into simpler ones.
We can use this technique with nested tables too, to keep our layout easier to maintain and display, while it’s still quite appealing to search engines spiders.
A note about this: the attributes ALIGN, WIDTH and HEIGHT are deprecated elements in HTML 4.01 (they are marked for deletion in future versions), but they aren’t likely to disappear any time soon. Currently, all browsers recognize these attributes.
From this point on, we could attach an external style sheet that contains all the proper style declarations for each table, continuing to improve the code like this:

```
<table id=”content” width=”80%” align=”right” height=”100%”>
<tr>
<td valign=”top”>Main content is placed here</td>
</tr>
</table>

<table id=”leftnavigation” width=”20%” height=”100%”>
<tr>
<td valign=”top”>Navigation Links are placed here</td>
</tr>
</table>
```

The layout is the same as before, but note the ID attribute for each table. Assigning ID contextual selectors from an external style sheet allow us to build the visual appearance for data contained into each table.
So far, this is a simple sample for table-based layout. It can be easily expanded to more complex layouts, adding nested tables’ techniques and so. However, I would strongly recommend using DIV tags and CSS for page layout, as we shall see in a moment. Trust me.

Step 3. The mighty DIV tag

According to the O’Reilly HTML Reference, “the DIV element gives structure and context to any block-level content in the document. Each DIV element becomes a generic block-level container for all content within the required start and end tags.” As we can see, the DIV tag is a powerful generic element well suited for being used as a container within our Web page. This turns it into a good candidate for creating sections or divisions (hence "DIV") of Web documents.
The concept is very intuitive. Instead of using tables as layout elements, we are going to take advantage of DIVS as excellent content containers to build our page layout. In conjunction with several CSS declarations, DIVS based layout are pretty easy to get.
Before we explain any further, two general categories used for Web page design must be clearly differentiated: “fixed” and “liquid" design. When referring to “fixed” design, we are talking about a Web page that has “fixed” dimensions. Widths (and optionally, heights) for each container element are assigned normally in pixels.
The final result of this approach is that Web pages are displayed with a determined “fixed” size, and they don’t “expand” to cover all the space in the computer monitor. Many “big” websites currently use this technique for achieving a consistent look and size across several user screen resolutions.

On the other hand, Web pages built over a “liquid” design principle will display as “elastic” documents, expanding as needed according to different monitor resolutions. Normally, dimensions for any container element are expressed in a percentage. “Liquid design,” when used properly, looks much more professional and takes one step further the definition that conceives a Web page as an “elastic” element, not a desktop document.

With all of these basic concepts out of the way, we will see different approaches for page layouts, depending on whether we follow a “fixed” design or a “liquid” design pattern.

Step 4. Fixed layout design with fixed boxes

Now, it’s time to build a basic two-column layout with fixed sizes.

Here is the code for styling our DIVS:

```
<style type="text/css">
<!--
#header {
  background: #0f0;
  position: absolute;
  top: 0px;
  left: 0px;
  width: 800px;
  height: 100px;
}
#leftcol {
  background: #f00;
  position: absolute;
top: 100px;
  left: 0px;
  width: 150px;
  height: 500px;
}
#content {
  background: #fff;
  position: absolute;
  top: 100px;
left: 150px;
  width: 700px;
  height: 500px;
}
#footer {
  background: #0f0;
  position: absolute;
  top: 500px;
  left: 0px;
  width: 800px;
  height: 100px;
}
-->
</style>
```

And here’s the HTML code:

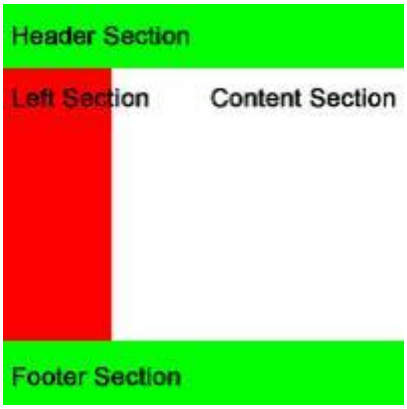
```
<div id="header">Header Section</div>
<div id="leftcol">Left Section</div>
<div id="content">Content Section</div>
<div id="footer">Footer Section</div>
```

The complete code is as follows:

```
<html>
<head>
<title>TWO-COLUM FIXED LAYOUT WITH FIXED BOXES</title>
<style type="text/css">
<!--
```

```
#header {
  background: #0f0;
  position: absolute;
  top: 0px;
  left: 0px;
  width: 800px;
  height: 100px;
}
#leftcol {
  background: #f00;
  position: absolute;
  top: 100px;
left: 0px;
  width: 150px;
  height: 500px;
}
#content {
  background: #fff;
  position: absolute;
  top: 100px;
left: 150px;
  width: 650px;
  height: 500px;
}
#footer {
  background: #0f0;
  position: absolute;
  top: 500px;
  left: 0px;
  width: 800px;
  height: 100px;
}
-->
</style>
</head>
<body>
<div id="header">Header Section</div>
<div id="leftcol">Left Section</div>
<div id="content">Content Section</div>
<div id="footer">Footer Section</div>
</body>
</html>
```

Let's see what's happening here. We have given fixed dimensions and absolute positions to all our DIVS. Widths and heights are expressed in pixels, as well as top and left coordinates. For the sake of this article, CSS code is within the same Web page, but it should always be attached as an external file. Please remember, separating content from visual presentation is a key concept that makes websites easily maintainable and accessible. The visual output for the code is the following:



Step 5. Three-column fixed layout

With little additions to the code, we can easily build a three-column fixed layout. Let's see how it works:

```
<style type="text/css">
<!--
#header {
  background: #0f0;
  position: absolute;
  top: 0px;
  left: 0px;
  width: 800px;
  height: 100px;
}
#leftcol {
  background: #f00;
  position: absolute;
  top: 100px;
left: 0px;
  width: 150px;
  height: 500px;
}
#rightcol {
  background: #f00;
  position: absolute;
  top: 100px;
left: 650px;
  width: 150px;
  height: 500px;
}
#content {
  background: #fff;
  position: absolute;
  top: 100px;
left: 150px;
  width: 500px;
  height: 500px;
}
#footer {
  background: #0f0;
  position: absolute;
  top: 500px;
  left: 0px;
  width: 800px;
  height: 100px;
}
-->
</style>
```

The HTML code is the following:

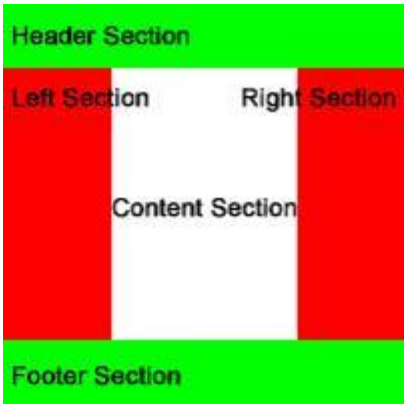
```
<div id="header">Header Section</div>
<div id="leftcol">Left Section</div>
<div id="content">Content Section</div>
<div id="rightcol">Right Section</div>
<div id="footer">Footer Section</div>
```

Here is the complete code:

```
<html>
<head>
<title>THREE-COLUMN FIXED LAYOUT WITH FIXED BOXES</title>
<style type="text/css">
<!--
#header {
  background: #0f0;
  position: absolute;
  top: 0px;
  left: 0px;
  width: 800px;
  height: 100px;
```

```
}
#leftcol {
  background: #f00;
  position: absolute;
  top: 100px;
  left: 0px;
  width: 150px;
  height: 500px;
}
#rightcol {
  background: #f00;
  position: absolute;
  left: 650px;
  top: 100px;
  width: 150px;
  height: 500px;
}
#content {
  background: #fff;
  position: absolute;
  top: 100px;
left: 150px;
  width: 500px;
  height: 500px;
}
#footer {
  background: #0f0;
  position: absolute;
  top: 500px;
  left: 0px;
  width: 800px;
  height: 100px;
}
-->
</style>
</head>
<body>
<div id="header">Header Section</div>
<div id="leftcol">Left Section</div>
<div id="content">Content Section</div>
<div id="rightcol">Right Section</div>
<div id="footer">Footer Section</div>
</body>
</html>
```

Here is the visual output:



As stated above, with minor modifications to the code, we can easily add another column to our existing layout, keeping widths and heights expressed in pixels. These simple examples show clearly that once we have defined our general layout, we are able to add more DIVS into the main containers or other elements, building more complex layouts with little or no effort. In a moment, we'll see another interesting approach for styling DIVS: floating boxes.

Step 6. Fixed layout design with floating boxes

Since we have developed a couple of examples giving absolute positions to each DIV, the next step is building the same schema, this time using the "float" property. Originally, the float property was not intended for building Web page layouts. It's commonly used to make text float around images or similar page structures. Possible values for float are: left (element floats to left), right (element floats to the right), none (element doesn't float), and inherit (element inherits from its parent). However, we can apply this property to our DIVS, and get the same effect as with fixed boxes. Let's see how it works:

```
<style type="text/css">
<!--
body {
  margin: 0px;
  padding: 0px;
}
#header {
  background: #0f0;
  width: 800px;
  height: 100px;
}
#leftcol {
  background: #f00;
  float: left;
  width: 150px;
  height: 500px;
}
#content {
  background: #fff;
  float: left;
  width: 650px;
  height: 500px;
}
#footer {
  background: #0f0;
  clear: both;
  width: 800px;
  height: 100px;
}
-->
</style>
```

The corresponding HTML is like this:

```
<div id="header">Header Section</div>
<div id="leftcol">Left Section</div>
<div id="content">Content Section</div>
<div id="footer">Footer Section</div>
```

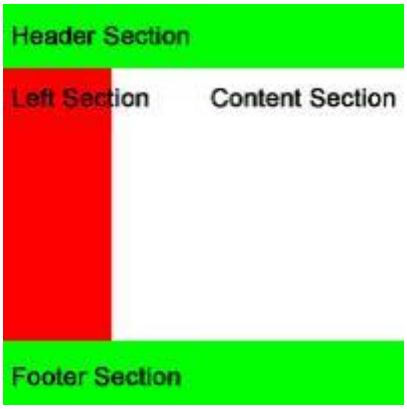
Again, we present the complete code:

```
<html>
<head>
<title>TWO-COLUMN FIXED LAYOUT WITH FLOATING BOXES</title>
<style type="text/css">
<!--
body {
  margin: 0px;
  padding: 0px;
}
#header {
  background: #0f0;
  width: 800px;
  height: 100px;
}
#leftcol {
```

```
  background: #f00;
  float: left;
  width: 150px;
  height: 500px;
}
#content {
  background: #fff;
  float: left;
  width: 650px;
  height: 500px;
}
#footer {
  background: #0f0;
  clear: left;
  width: 800px;
  height: 100px;
}
-->
</style>
</head>
<body>
<div id="header">Header Section</div>
<div id="leftcol">Left Section</div>
<div id="content">Content Section</div>
<div id="footer">Footer Section</div>
</body>
</html>
```

There is something really interesting going on. We have applied the float property to the “leftcol” DIV, making it float to the left. Then, we made the content DIV float to the left too, sticking the two boxes to each other. The same layout effect has been achieved, making the boxes float. Please note, all absolute position declarations were removed, as well as top and left coordinates properties, so the whole file size has been decreased.

Finally, as to making our footer display correctly, we have “cleared” the float properties previously declared. Pretty good, isn’t it? The visual output is almost identical to that achieved with fixed boxes.



Step 7. Adding a right column

As you can guess, with minor additions, we can add a right column to our layout:

```
<style type="text/css">
<!--
body {
  margin: 0px;
  padding: 0px;
}
#header {
  background: #0f0;
  width: 800px;
```

```
  height: 100px;
}
#leftcol {
  background: #f00;
  float: left;
  width: 150px;
  height: 500px;
}
#rightcol {
  background: #f00;
  float: left;
  width: 150px;
  height: 500px;
}
#content {
  background: #fff;
  float: left;
  width: 500px;
  height: 500px;
}
#footer {
  background: #0f0;
  clear: left;
  width: 800px;
  height: 100px;
}
-->
</style>
```

The HTML code is the following:

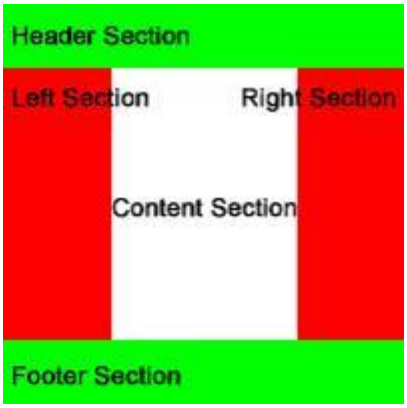
```
<div id="header">Header Section</div>
<div id="leftcol">Left Section</div>
<div id="content">Content Section</div>
<div id="rightcol">Left Section</div>
<div id="footer">Footer Section</div>
```

And, the complete code is listed below:

```
<html>
<head>
<title>THREE-COLUMN FIXED LAYOUT WITH FLOATING BOXES</title>
<style type="text/css">
<!--
body {
  margin: 0px;
  padding: 0px;
}
#header {
  background: #0f0;
  width: 800px;
  height: 100px;
}
#leftcol {
  background: #f00;
  float: left;
  width: 150px;
  height: 500px;
}
#rightcol {
  background: #f00;
  float: left;
  width: 150px;
  height: 500px;
}
#content {
  background: #fff;
```

```
  float: left;
  width: 500px;
  height: 500px;
}
#footer {
  background: #0f0;
  clear: left;
  width: 800px;
  height: 100px;
}
-->
</style>
</head>
<body>
<div id="header">Header Section</div>
<div id="leftcol">Left Section</div>
<div id="content">Content Section</div>
<div id="rightcol">Left Section</div>
<div id="footer">Footer Section</div>
</body>
</html>
```

The visual output is almost the same as for fixed boxes:



In this case, the same principle has been applied to make our DIVS float to left. Since the left section floats to the left, so does the content section and the right section. Actually, all of they are sticking to the left, achieving the desired result. Again, this technique saves code as compared to using fixed boxes.
Lastly, but not least, will see the “Holy Grail” for page layouts: “liquid” design with floating boxes. Let’s go there now.

Step 8. Liquid design with floating boxes

Any well-designed Web page must flow properly, expanding the visual content to fit at any screen resolution. The professional look is evident, and extremely elegant, when it’s reached. However, as with everything in life, it has some limitations. These include many user screen options, monitor resolutions, different hardware and so on. Keeping in mind these conditions, “liquid” design is still the definitive way to go for truly professional websites.
Since we are talking about this issue, let’s see some code to get our “liquid” effect done.

```
<style type="text/css">
<!--
body {
  margin: 0px;
  padding: 0px;
}
#header {
  background: #0f0;
  width: 100%;
}
#leftcol {
```

```
  background: #f00;
  float: left;
  width: 19%;
  height: 500px;
}
#content {
  background: #fff;
  float: right;
  width: 80%;
  height: 500px;
}
#footer {
  background: #0f0;
  clear: both;
  width: 100%;
}
-->
</style>
```

And the HTML code is the following:

```
<div id="header">Header Section</div>
<div id="leftcol">Left Section</div>
<div id="content">Content Section</div>
<div id="footer">Footer Section</div>
```

The complete code is listed below:

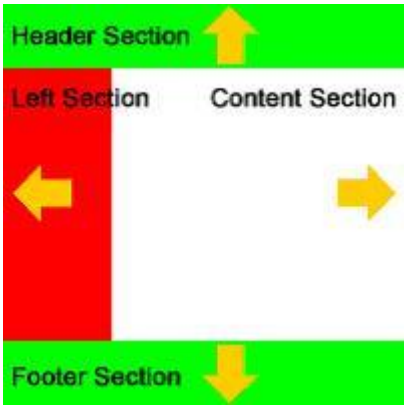
```
<html>
<head>
<title>TWO-COLUMN LIQUID LAYOUT WITH FLOATING BOXES</title>
<style type="text/css">
<!--
body {
  margin: 0px;
  padding: 0px;
}
#header {
  background: #0f0;
  width: 100%;
}
#leftcol {
  background: #f00;
  float: left;
  width: 19%;
  height: 500px;
}
#content {
  background: #fff;
  float: right;
  width: 80%;
  height: 500px;
}
#footer {
  background: #0f0;
  clear: both;
  width: 100%;
}
-->
</style>
</head>
<body>
<div id="header">Header Section</div>
<div id="leftcol">Left Section</div>
<div id="content">Content Section</div>
<div id="footer">Footer Section</div>
```

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

As we see in our new developed layout, there are many noteworthy things to explain in detail. All of the DIVS containers' widths are expressed in percentages, so they will expand to fit the whole screen space. For the sake of this example, we have declared heights in pixels for our DIVS. This is a bad practice! When applied to real world design, DIVS heights will expand accordingly to the content included in them. Sure, this is a well-known subject for experienced designers, but it could be a very important issue to be considered for beginners.

Back to the code, we make our “leftcol” DIV float to the left, and its width is set to 19%. The “content” DIV is floated to the right and its width is set to 80%, covering the remaining space. Header and footer sections widths are set to 100%.

Visual representation for the output is appreciated in the following image:



As depicted above, the containers will expand to fill the screen, filling the complete space, and exposing a very consistent look for different screen resolutions.

Step 9. Three-column liquid layout with floating boxes

Finally, we’ll build a three-column liquid layout with floating boxes. Without a doubt, this is the most used layout model on the Web.
Here is the code:

```
<style type="text/css">
<!--
body {
  margin: 0px;
  padding: 0px;
}
#header {
  background: #0f0;
  width: 100%;
}
#leftcol {
  background: #f00;
  float: left;
  width: 20%;
  height: 500px;
}
#rightcol {
  background: #f00;
  float: right;
  width: 20%;
  height: 500px;
}
#content {
  background: #fff;
```

```
  float: left;
  width: 59%;
  height: 500px;
}
#footer {
  background: #0f0;
  clear: both;
  width: 100%;
}
-->
</style>
```

The HTML code is as follows:

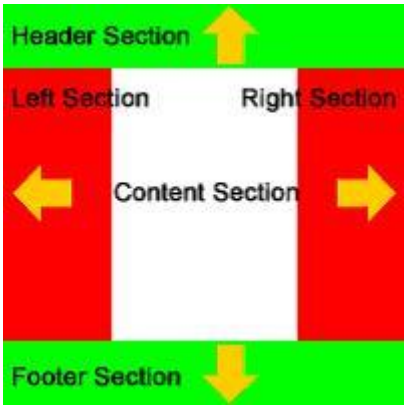
```
<div id="header">Header Section</div>
<div id="leftcol">Left Section</div>
<div id="rightcol">Right Section</div>
<div id="content">Content Section</div>
<div id="footer">Footer Section</div>
```

The complete code is listed below:

```
<html>
<head>
<title>THREE-COLUMN LIQUID LAYOUT WITH FLOATING BOXES</title>
<style type="text/css">
<!--
body {
  margin: 0px;
  padding: 0px;
}
#header {
  background: #0f0;
  width: 100%;
}
#leftcol {
  background: #f00;
  float: left;
  width: 20%;
  height: 500px;
}
#rightcol {
  background: #f00;
  float: right;
  width: 20%;
  height: 500px;
}
#content {
  background: #fff;
  float: left;
  width: 59%;
  height: 500px;
}
#footer {
  background: #0f0;
  clear: both;
  width: 100%;
}
-->
</style>
</head>
<body>
<div id="header">Header Section</div>
<div id="leftcol">Left Section</div>
<div id="rightcol">Right Section</div>
<div id="content">Content Section</div>
<div id="footer">Footer Section</div>
```

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

The corresponding visual representation is shown below:



Let's explain in detail the code for this layout. As in the previous example, we make our "leftcol" DIV float to left and its width is set to 20%. Our "content" DIV is floated to left and its width is set to 59%, while our recently added "rightcol" DIV is floated to the right, and its width is set to 20% too. Header and Footer DIVS are kept to the same width of 100%.

Finally, we have reached our main goal, having a Web page layout that expands nicely, filling the whole screen space flawlessly. The "elastic" concept is well deserved here.

One final consideration is worth noting: since DIVS containers will expand down as we insert more elements into them, sooner or later, we will end up with different heights for each column of the Web page. How can we make them have the same heights? With a little bit of JavaScript, we can achieve the desired result. That will be considered as homework, since it's out of the scope of this article.

Wrapping Up

As seen on the examples presented here, DIV layout with CSS is a very powerful technique, well-suited to face different design approaches and styles, and a big improvement for our background as Web designers. Web pages are faster to download, avoid useless markup, look more appealing to search engines, and keep us closer to Web standards. There is a lot to gain in migrating from table-based to div-based layout. Just give it a try. Good luck!

8.3. Exercise 1: Modify 8.1 using DIV tag

Modify the example in 8.1 to use DIV tag for layout to get the same result.

8.4. Exercise 2: CSS Box Model

Follow the instruction in the slides to make the layout of your webpage which includes 4 areas: header, main, side-bar and footer. Create other web pages (2-3 web pages) applying this layout and style.