

98-130

TECHNIQUES FOR WEB INTERFACES

LESSON 4: "LAYOUT & CASE STUDIES"

LECTURE OVERVIEW

- Multi-column layouts
 - Text zoom vs. page zoom
 - Typographic line length
 - Layout types: fixed, liquid, elastic
-

RELATED READING

- Dave Child: *Faux Columns for Liquid Layouts*
<http://www.addedbytes.com/css/faux-columns-for-liquid-layouts/>
 - Mike Cherim: *CSS Layouts: The Fixed. The Fluid. The Elastic.*
<http://green-beast.com/blog/?p=199>
-

MULTI-COLUMN LAYOUTS

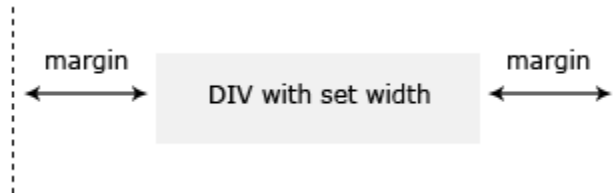
It's incredibly common to have a website laid out in multiple columns. (Facebook has three, the 98-130 site has two, etc.) This has roots in the concept of the grid (which we'll discuss in one of our design-oriented lectures later). Usually there will be a header and a footer, too.

There are a ton of ways that we can layout multiple columns in CSS: the most common methods use absolute positioning or floats. We'll cover the latter, since it's usually the easiest method.

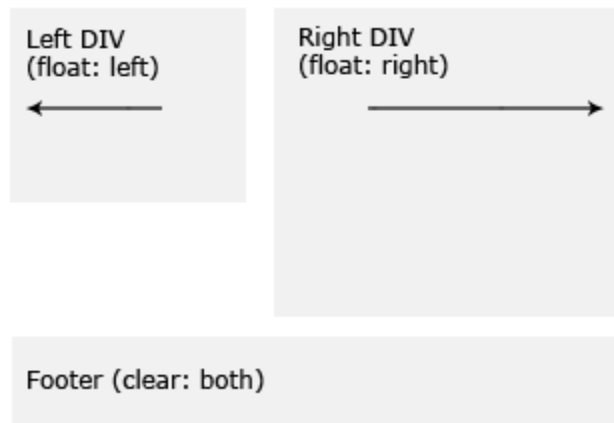
CENTERING USING MARGINS Before we do anything, let's learn how to horizontally center a block-level element. (A common mistake is to think that this is done with `text-align: center`, which actually isn't exactly what we want; that would only center text.) The trick is actually to give the element a width, and then make the left/right margin equal in size and fill to expand the containing element (thus, 'pushing' the centered element towards the center from both sides). For example:

```
width: 500px;
margin: 0 auto;
```

Note that we're using another shortcut version of margin that defines the top and bottom in the first unit provided (0) and the left and right in the second (auto). Auto is a special keyword indicating exactly what we wanted above.



TWO-COLUMN LAYOUT Now say we just want a two-column layout (the common case is one column for navigation and another for content). We can do this by having one `DIV` for the left column (which we float to the left) and another `DIV` for the right column (which we float to the right), while defining widths for each according to how wide we want them to be. We can even throw in a footer by clearing both the floats (which will push the footer down below the floated `DIV`s, as explained before).



Pretty simple. Note that we can contain all of this inside of a “wrapper” `DIV`, give it all a set width, and center it. For example:

```
<div id="wrapper">
  <div id="left">...</div>
  <div id="right">...</div>
  <div id="footer">...</div>
</div>
```

with the CSS:

```
#wrapper { width: 760px; margin: 0 auto; }
#left { float: left; width: 300px; }
#right { float: right; width: 450px; }
#footer { clear: both; }
```

THREE-COLUMN LAYOUT Instead, what if we wanted a three-column layout: navigation, primary content, and secondary content? The easiest way to do this is simply to split our “right” DIV into two columns itself. For example:

```
<div id="wrapper">
  <div id="navigation">...</div>
  <div id="content">
    <div id="primary_content">...</div>
    <div id="secondary_content">...</div>
  </div>
  <div id="footer">...</div>
</div>
```

with additional CSS like:

```
#primary_content { float: left; width: 40%; }
#secondary_content { float: right; width: 60%; }
```

FAUX COLUMNS So that’s all pretty simple. But note that the columns don’t expand to match each other’s heights. For example, in the diagram for two column layouts, the left column’s background won’t expand all the way down the page. Not really much of a column.

It turns out that the easiest way to solve this problem is actually just to fake it. Using a technique called *faux columns*, we take the #wrapper DIV and give it a vertically-tiling background image that lines up right behind the left DIV, so it *looks* like the left DIV extends all the way down.

Clearly, this doesn’t work well if we don’t have a fixed dimension for our column (we can make a background image that is 300px wide, but we can’t make one that is 30% wide). We won’t get further into that, but if you’d like, check out the link in the related readings section for a solution.

TEXT ZOOM VS. PAGE ZOOM

Before we move on, let's talk briefly about two big concepts: full-page zoom and text zoom. These aren't really CSS topics at all. Instead, they refer to how browsers implement zooming.

Prior to a few years ago, all browsers used *text-only zoom*. A user could increase and decrease the size of text on the page. This made ems even more handy, because measuring an entire site in ems meant that when the user resized the text, the entire site's proportion and structure would resize accordingly, so Grandma and Grandpa wouldn't have to look at a really ugly site just to read it.

Now, however, browsers have gotten much better at *full page zoom* and most offer it as a default. By default, for instance, Firefox will zoom in just by actually, well, zooming in. It'll magnify the entire page: images, text, everything. Your proportions are automatically kept intact without having to do anything. So are ems useless now? Nope. Imagine a special-needs user that has his text size bumped up by default; in fact, this kind of person is one of the reasons CSS was created, so it's very easy for him to do so. An em-based design will accommodate him very well.

Take-home idea? There really is none, except that em-based design was a great idea (and still is). It's just very important to know this concept, especially before we talk about elastic layouts.

TYPOGRAPHIC LINE LENGTH

As lesson one in typography, know that the "ideal" line length is about 66 characters for most serif fonts (though 40-50 characters works for multiple-column pieces like you'd find in a newspaper). Too many or too few will have an adverse effect on readability. Keep this in mind!

LAYOUT TYPES: FIXED, LIQUID, ELASTIC

This discussion will revolve around how what we use for our units will influence the layout of the page.

FIXED LAYOUTS Fixed layouts are achieved by defining our dimensions in terms of absolute, non-changing units: most commonly, this means the px. The layout is referred to as *fixed* because it doesn't scale when you resize the browser or the text: a 600px wide column will always be 600px wide. This is probably the most common type of layout, simply because it's easy and gives the designer the most control; it's also very simple to get a great line length if

you can fix the dimensions.

However, it means that, in some cases, a lot of the screen real estate is going to be wasted. Think about a 700px wide layout on a wide-screen monitor. Magnifying this issue is the fact that Windows users are especially used to maximizing their windows all the time (accept this habit; it's been hardcoded into their practices over the years by the Windows GUI).

LIQUID LAYOUTS Liquid layouts are usually defined in units of percentage (%). For example, one column is 33% of the browser window and the other is 66%. Thus, when you resize the browser window, the layout resizes as well.

This is good because it maximizes screen space. However, it can have crippling effects on line length if you're not careful. Take a look at Wikipedia!

Nevertheless, liquid layouts can be done well. Amazon's home page, for instance, shows horizontal thumbnails of products in categories. If you resize the browser window, they actually show more or less, depending on how many they can fit on one line! This is an awesome idea (and is technically accomplished with the `overflow` property, if you're interested).

ELASTIC LAYOUTS Elastic layouts are defined using the `em`. As previously mentioned, this means that they will scale according to the current text size. Actually, this gives us the best control over line length, but it's the most complicated of our layout techniques and involves quite a bit of math. Additionally, what happens if the user zooms in *too much*? If he keeps zooming, our design will keep expanding to accommodate him, until it creates a horizontal scrollbar.

We can actually fix this by using a *hybrid layout*, where we give a `max-width` in % (let's say 100%) to our layouts too. This means that the page will keep expanding as long as it is less than 100% of the window of the browser window; then it won't get any bigger, though the text can still zoom in. If you're interested in this idea, feel free to check out the last part of the layouts article in the Related Reading section, or just look at the source of the 98-130 site, which uses this exact technique.

NEXT LECTURE

Next lecture, we'll cover some tricks with backgrounds, links, and tabbed navigation techniques. (We'll also do a Facebook interview question!) This

will conclude the “core” of our front-end engineering section of the course, and then we’ll start to slowly transition into design topics.