Arranging Elements

It's time to teach your XHTML elements new tricks. We're not going to let those XHTML elements just sit there anymore – it's about time they get up and help us create some pages with real *layouts*. How? Well, you've got a good feel for the and structural elements and you know all about how the box model works, right? So, now it's time to use all that knowledge to craft some real designs. No, we're not just talking about more background and font colors, we're talking about full blown professional designs using multi-column layouts. This is the chapter where everything you've learned comes together.

Did you do the Super Brain Power?

If you didn't do the **Super Brain Power** at the end of the last chapter, then march right back there and do it. It's required.

Okay, now that we have that out of the way, at the end of the last chapter, we left you with a bit of a cliffhanger. We asked you to move the elixirs ** ** up under the logo, and then add one little property to the elixirs rule in your CSS, like this:

```
float: right;
```

And, wow, what a difference one property can make! All of a sudden the page has gone from a fairly ordinary-looking Web page to a great-looking Web page with two columns. It's immediately more readable and pleasant to the eye.

So what's the magic? How did this seemingly innocent little property produce such big effects? And, can we use this property to do even more interesting things with our pages? Well, of course, this is Head First, after all. But first, you're going to need to learn how a browser lays out elements on a page. Once you know that, we can talk about all kinds of ways you can alter how it does that layout, and also how you can start to position your elements on the page.

Here's the good news: you already know all about block elements and inline elements, and you even know about the box model. These are the real foundations of how the browser puts a page together. Now all you need to know is exactly how the browser takes all the elements in a page, and decides where they go.

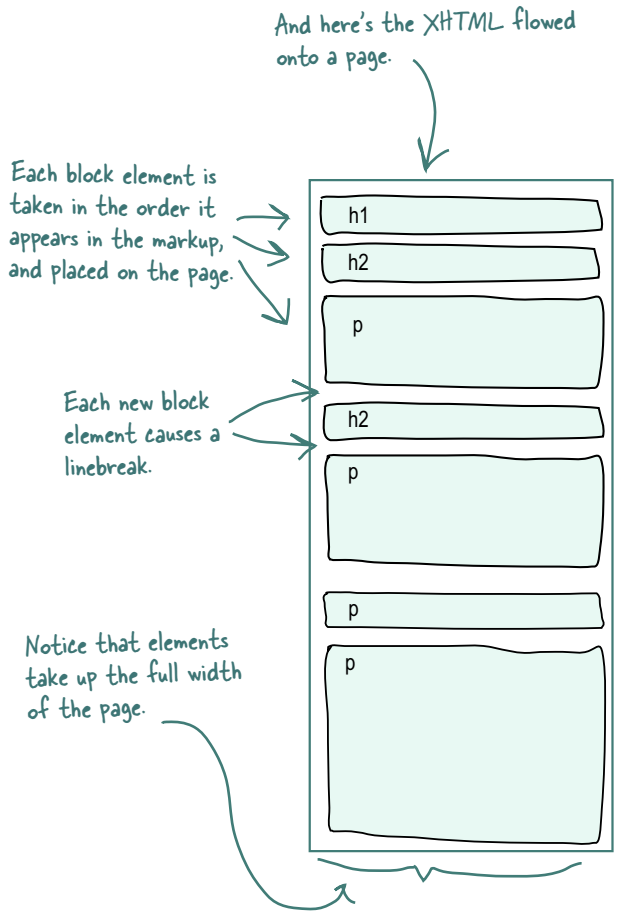
Use the flow, Luke

The Flow is what gives a CSS master his power. It's an energy field created by all living things. It surrounds us and penetrates us. It binds the galaxy together....
Oh, sorry.

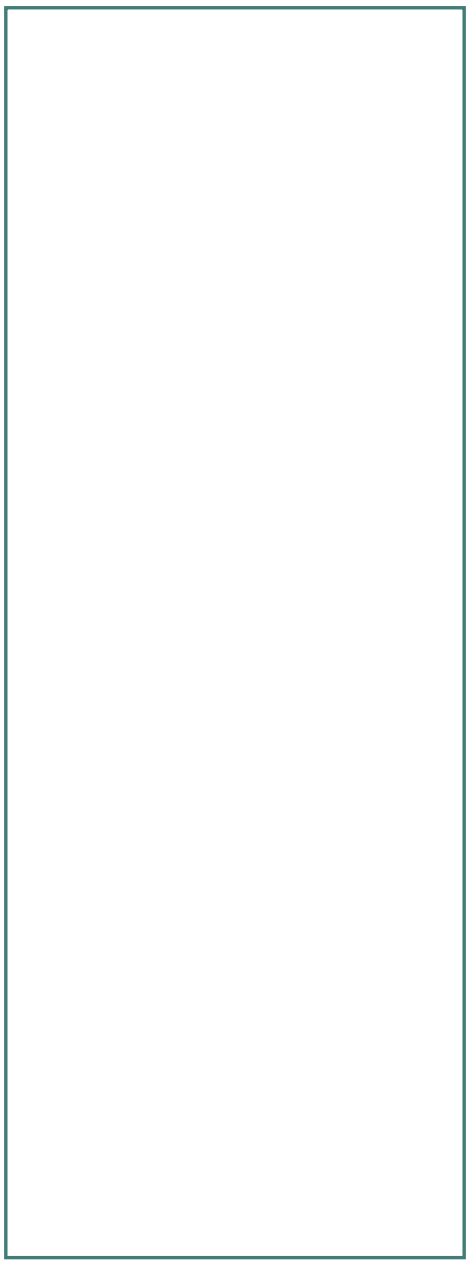
Flow is what the browser uses to lay out a page of XHTML elements. The browser starts at the top of any XHTML file and follows the flow of elements from top to bottom, displaying each element it encounters. And, just considering the block elements for a moment, it puts a linebreak between each one. So the first element in a file is displayed first, then a linebreak, followed by the second element, then a linebreak, and so on, from the top of your file to the bottom. That's flow.

Here's a little "abbreviated" XHTML.

```
<html>
   <head>...</head>
   <body>
        <h1>...</h1>
        <h2>...</h2>
        <p>...</p>
        <h2>...</h2>
        <p>...</p>
        <p>...</p>
        <p>...</p>
   </body>
</html>
```



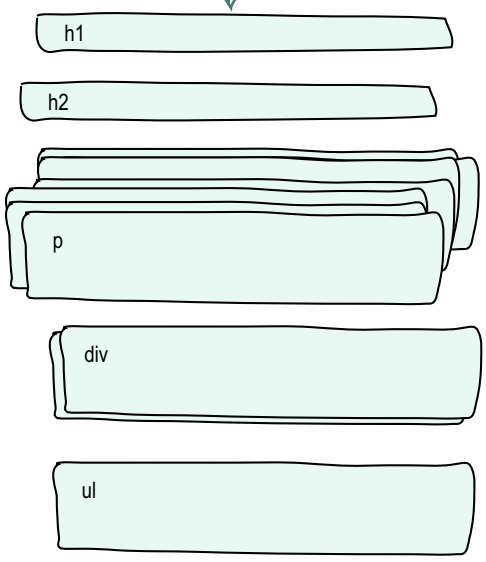
Here's your page. Flow the block elements in "lounge.html" here.



BE the

Open your "lounge.html" file and locate all the block elements. Flow each one on to the page to the left. Just concentrate on the block elements nested directly inside the body element. You can also ignore the "float" property in your CSS because you don't know what it does yet. Check your answer before moving on.

Here are all the block elements you'll need to complete the job.



What about inline elements?

So you know that block elements flow top to bottom, with a linebreak in between each element. Easy enough. What about the inline elements?

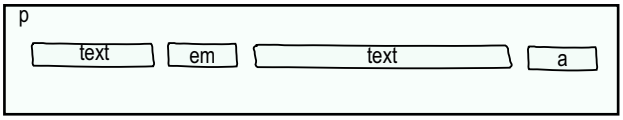
Inline elements are flowed next to each other, horizontally, from top left to bottom right. Here's how that works.

Here's another little snippet of XHTML.

```
<p>
Join us <em>any evening</em> for
these and all our other wonderful <a
href="beverages/elixir.html" title="Head
First Lounge Elixirs">elixirs</a>.
</p>
```

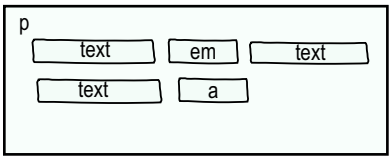
If we take the inline content of this <p> element and flow it onto the page, we start at the top left.

The inline elements are laid next to one another horizontally, as long as there is room on the right to place them.



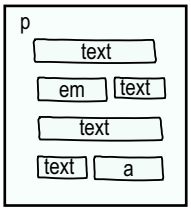
Here, there's room to fit all the inline elements horizontally. Notice that text is a special case of an inline element. The browser breaks it into inline elements that are the right size to fit the space.

So what if we make the browser window a little thinner, or we reduce the size of the content area with the width property? Then there's less room to place the inline elements in. Let's see how this works.



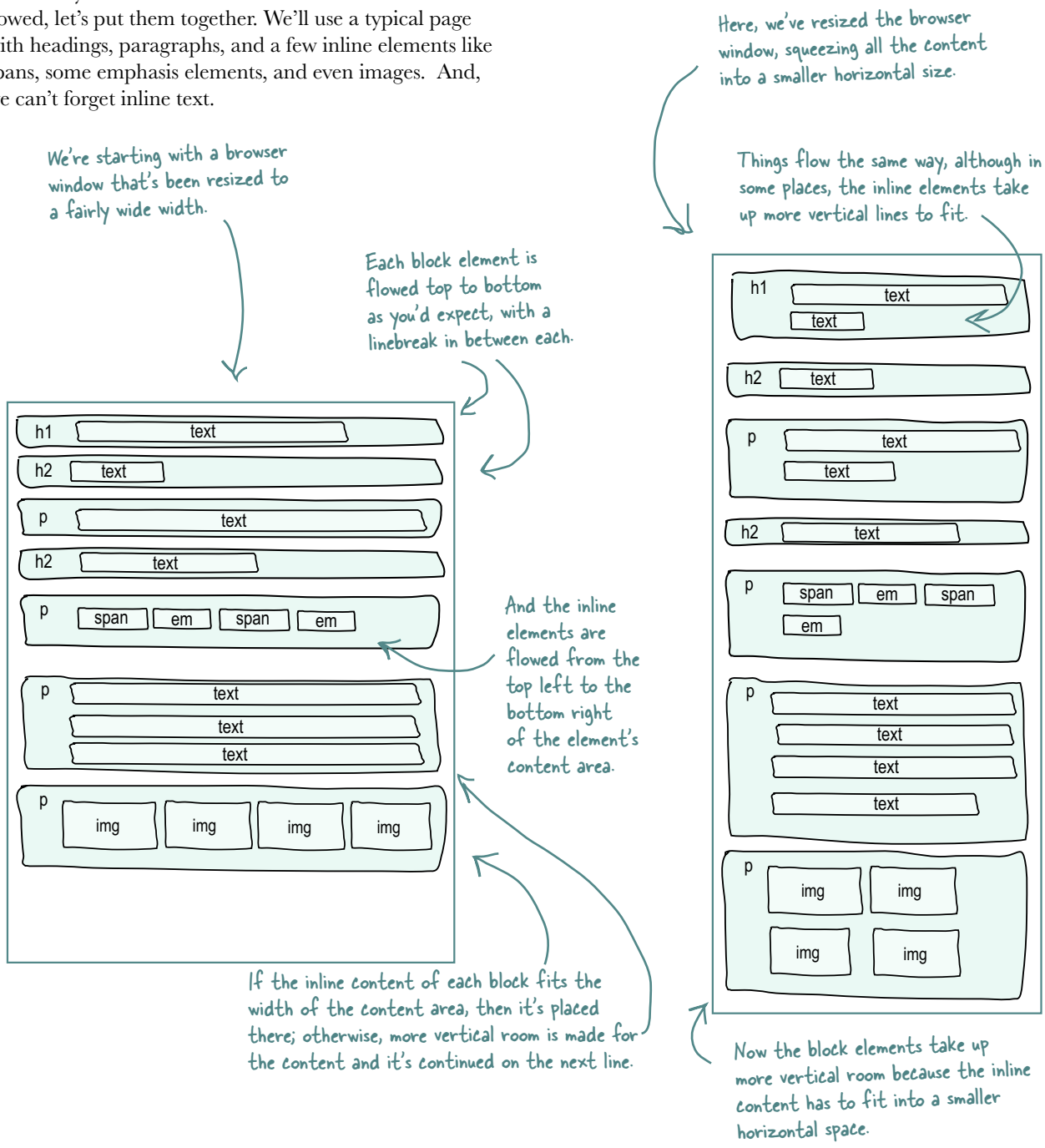
Now the content has been flowed left to right until there's no more room, and then the content is placed on the next line. Notice the browser had to break the text up a little differently to make it fit nicely.

And if we make the content area even thinner, look what happens. The browser uses as many lines as necessary to flow the content into the space.



How it all works together

Now that you know how block and inline elements are flowed, let's put them together. We'll use a typical page with headings, paragraphs, and a few inline elements like spans, some emphasis elements, and even images. And, we can't forget inline text.



One more thing you should know about flow and boxes

Let's zoom in just a bit and look at one more aspect of how the browser lays out block and inline elements. It turns out that the browser treats margins differently depending on which type of element is being placed on the page.

When the browser is placing two inline elements next to each other...

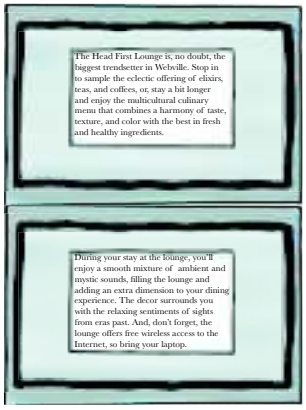
When the browser has the task of placing two inline elements side by side, and those elements have margins, then the browser does what you might expect. It creates enough space between the elements to account for both margins. So, if the left element has a margin of 10 pixels and the right has a margin of 20 pixels, then there will be 30 pixels of space between the two elements.

Here we've got two images side by side. Images are inline elements, right? So, the browser uses both of their margins to calculate the space that goes between them.

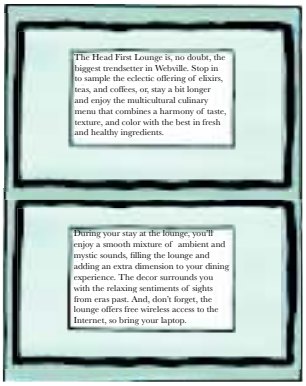
When the browser is placing two block elements on top of each other...

Here's where things get more interesting. When the browser places two block elements on top of each other, it collapses their shared margins together. The height of the collapsed margin is the height of the largest margin.

When the browser places two block elements on top of each other, it collapses their margins.



Their shared margin is the size of the larger of the two margins. Say the top element's bottom margin is 10 pixels, and the bottom element's top margin is 20 pixels. Then the collapsed margin will be 20 pixels.



there are no Dumb Questions

Q: So if I have a block element with a zero margin, and a block element below it with a top margin of 20, the margin between them would end up being 20?

A: Right. If one of the margins is bigger, then the margin becomes the larger of the two, even if one margin is zero. But if the margins are the same, say, 10 pixels, then they just get collapsed together to 10 pixels total.

Q: Can inline elements really have margins?

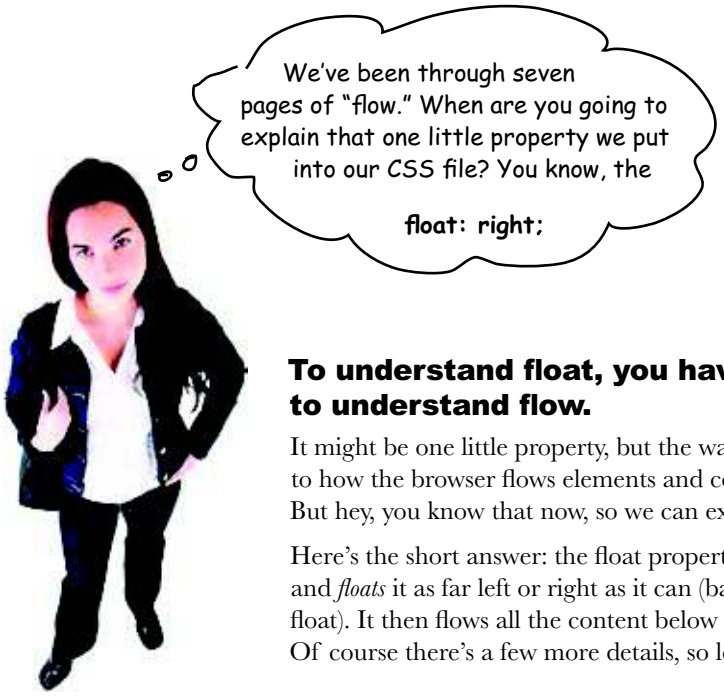
A: They sure can, although you won't find that you set the margins of inline elements often. The one exception is images. It is very common to not only set margins but also borders and padding on images. And while we aren't going to be setting any inline element margins in this chapter, we will be setting the border on one a little later.

Q: What if I have one element nested inside another and they both have margins? Can they collapse?

A: Yes, that can happen. Here's how to figure out when they will: whenever you have two vertical margins touching, they will collapse, even if one element is nested inside the other. Notice that if the outer element has a border, the margins will never touch, so they won't collapse. But if you remove the border, they will. This is sometimes puzzling when you first see it happen, so put it in the back of your mind for when it occurs.

Q: So how exactly does text work as an inline element since its content is not an element?

A: Even if text is content, the browser needs to flow it onto the page, right? So the browser figures out how much text fits on a given line, and then treats that line of text as if it were an inline element. The browser even creates a little box around it. As you've seen, if you resize the page, then all those blocks may change as the text is refit within the content area.



To understand float, you have to understand flow.

It might be one little property, but the way it works is closely tied to how the browser flows elements and content onto the page. But hey, you know that now, so we can explain float.

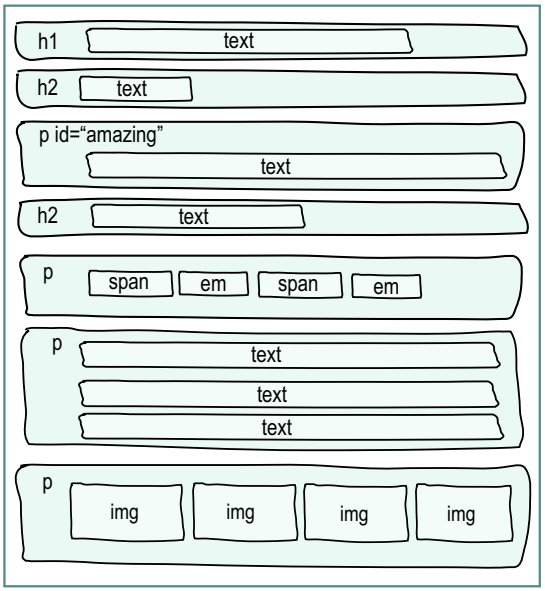
Here's the short answer: the float property first takes an element and *floats* it as far left or right as it can (based on the value of float). It then flows all the content below it around the element. Of course there's a few more details, so let's take a look...

How to float an element

Let's step through an example of how you get an element to float, and then we'll look at what it does to the flow of the page when you do.

First, give it an identity

Let's take one of these paragraphs and give it an id. We'd like to call it the "amazing floating paragraph", but we'll just call it "amazing" for short.

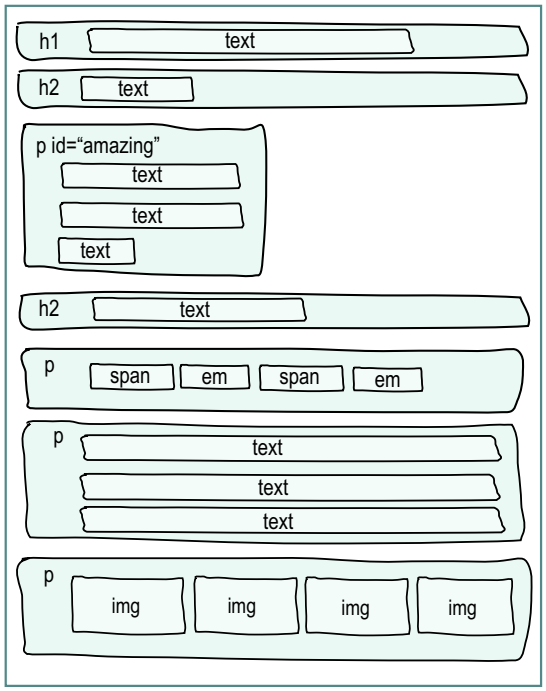


Now give it a width

A requirement for any floating element is that it have a width. We'll make this paragraph 200 pixels wide. Here's the rule:

```
#amazing {
   width: 200px;
}
```

Now the paragraph is 200 pixels wide, and the inline content contained in it has adjusted to that width. Keep in mind, the paragraph is a block element, so no elements are going to move up beside it because all block elements have linebreaks before and after them.

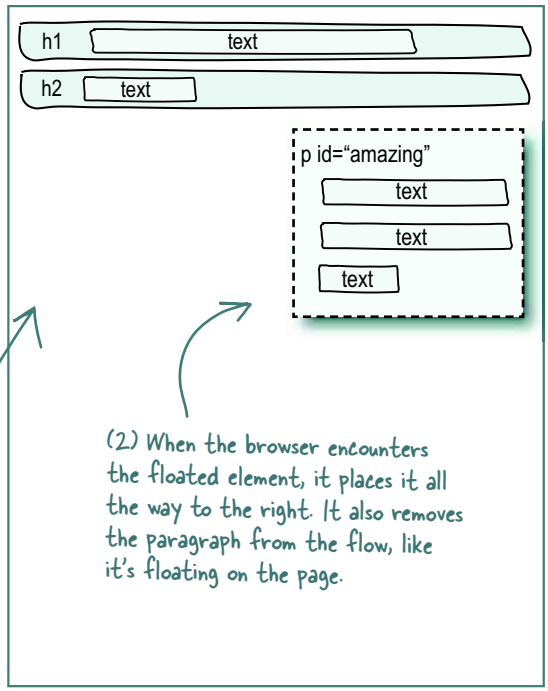


Now float it

Now let's add the **float** property. The **float** property can be set to either left or right. Let's stick with right:

```
#amazing {
   width: 200px;
   float: right;
}
```

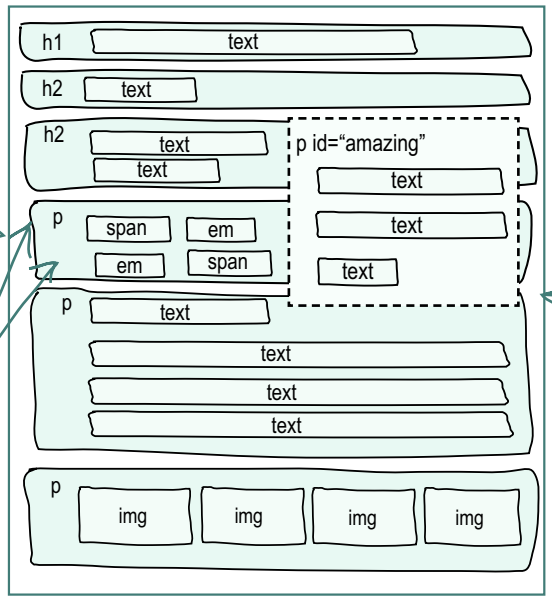
Now that we've floated the "amazing" paragraph, let's step through how the browser flows it and everything else on the page.



(1) First the browser flows the elements on the page as usual, starting at the top of the file and moving towards the bottom.

(2) When the browser encounters the floated element, it places it all the way to the right. It also removes the paragraph from the flow, like it's floating on the page.

(3) Because the floated paragraph has been removed from the normal flow, the block elements are filled in, like the paragraph isn't even there.



Notice that the block elements are positioned under the floated element. That's because the floated element is no longer part of the normal flow.

(4) But when the inline elements are positioned, they respect the boundaries of the floated element. So they are flowed around it.

However, when the inline elements are flowed within the block elements, they flow around the borders of the floating element.

Behind the scenes at the lounge

Now you know all about flow and how floated elements are placed on the page. Let's look back at the lounge and see how this all fits together.

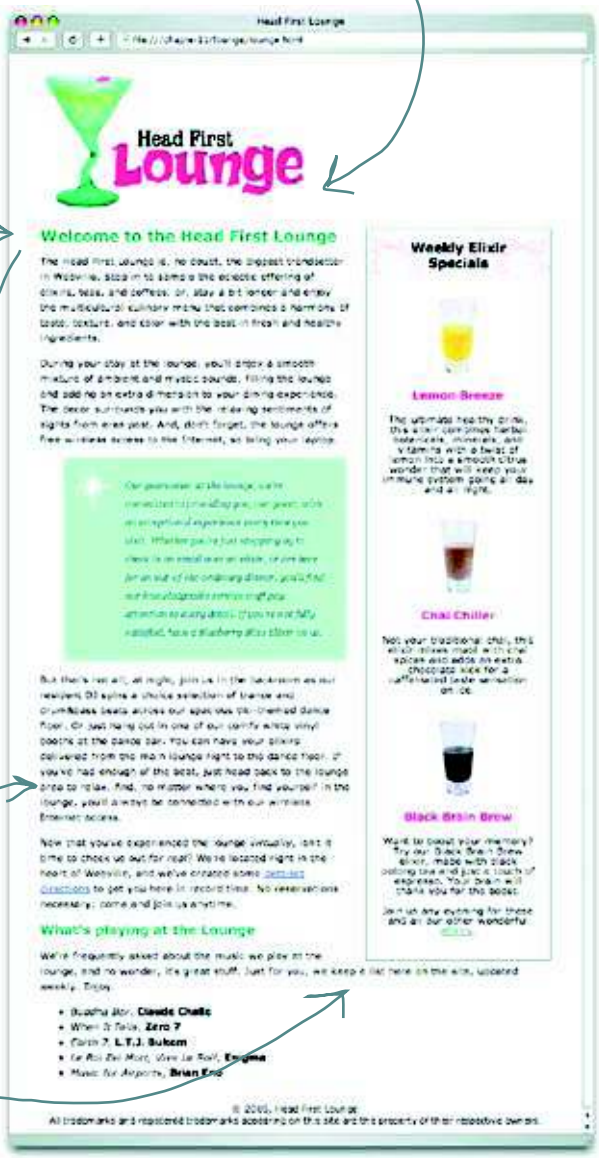
Remember, in addition to setting the elixirs to float right, we also moved the up just below the logo at the top of the page.

Moving the allowed us to float it to the right and then have the entire page flow around it. If we had left the elixirs below the music recommendations, then the elixirs would have been floated right after most of the page had been placed.

All these elements follow the elixirs in the XHTML, so they are flowed around it.

Remember that the elixirs is floating on top of the page. All the other elements are underneath it, but the inline content respects the elixirs' boundaries when they are flowing into the page.

Also notice that the text wraps around the bottom of the elixirs, because the text is contained in a block element that is the width of the page. If yours doesn't wrap, try narrowing your browser window until the text wraps underneath the elixirs.



Move the elixirs back to its original place below the music recommendations, then save and reload the page. Where does the element float now? Check your answer in the back and then put your elixirs back underneath the header.

It looks like you've got a new assignment. Starbuzz really could use some improvement. Sure, you've done a great job of creating the typical top to bottom page, but now that you know flow, you should be able to give Starbuzz Coffee a slick new look that is more user-friendly than the last design.

We do have a little secret though... we've been working on this one a bit already. We've created an updated version of the site. Your job is going to be to provide all the layout. Don't worry, we'll bring you up to speed on everything we've done so far – it's nothing you haven't seen before.

The new Starbuzz

Let's take a quick look at what we've got so far, starting with the page as it looks now. Then we'll take a peek at the markup and the CSS that's styling it.

We've got a header now with a new spiffy Starbuzz logo and the company mission statement. This is actually just a GIF image.

We've got four sections: the header, a main content section, a section advertising something new called the "Bean Machine," and a footer.

Each section is a that can be styled independently.

It looks like we've got one background color for the page as a whole, and then each is using an image as a background.

Here's the "Bean Machine" area. This links to a new area of Starbuzz Coffee where you can order your coffee beans online. This link doesn't work just yet because you're going to build the Bean Machine in an upcoming chapter.

Here's the footer. It doesn't use a background image, just a background color.

Notice that we've styled the links in an interesting way, with dotted underlines...

A look at the markup

Now let's take a look at the new Starbuzz markup. We've taken each of the logical sections and placed it into a ** **, each with its own **id**. Beyond the ** **s and ****s, there's really nothing here that you hadn't already seen by about Chapter 5. So, take a quick look and get familiar with the structure, and then turn the page to check out the CSS style.

Here's all the usual XHTML administravia.

Followed by a for the header and a for the main content area.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en" >
<head>
        <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
        <title>Starbuzz Coffee</title>
        <link type="text/css" rel="stylesheet" href="starbuzz.css" />
</head>
<body>

        <div id="header">
                <img src="images/header.gif" alt="Starbuzz Coffee header image" />
        </div>

        <div id="main">
                <h1>QUALITY COFFEE, QUALITY CAFFEINE</h1>
                <p>
                   At Starbuzz Coffee, we are dedicated to filling all your caffeine needs through our
                   quality coffees and teas. Sure, we want you to have a great cup of coffee and a great
                   coffee experience as well, but we're the only company that actively monitors and
                   optimizes caffeine levels. So stop by and fill your cup, or order online with our new Bean
                   Machine online order form, and get that quality Starbuzz coffee that you know will meet
                   your caffeine standards.
                </p>
                <p>
                   And, did we mention <em>caffeine</em>? We've just started funding the guys doing all
                   the wonderful research at the <a href="http://buzz.headfirstlabs.com"
                   title="Read all about caffeine on the Buzz">Caffeine Buzz</a>.
                   If you want the latest on coffee and other caffeine products,
                   stop by and pay them a visit.
                </p>
                <h1>OUR STORY</h1>
                <p>
                    "A man, a plan, a coffee bean". Okay, that doesn't make a palindrome, but it resulted
                    in a damn good cup of coffee.  Starbuzz's CEO is that man, and you already know his
                    plan: a Starbuzz on every corner.
                </p>
                <p>
                    In only a few years he's executed that plan and today
                    you can enjoy Starbuzz just about anywhere. And, of course, the big news this year
                    is that Starbuzz teamed up with Head First readers to create Starbuzz's Web presence,
                    which is growing rapidly and helping to meet the caffeine needs of a whole new set of
                    customers.
                </p>
                <h1>STARBUZZ COFFEE BEVERAGES</h1>
                 <p>
                        We've got a variety of caffeinated beverages to choose
```

This is more of the main content area continued over here.

```
                from at Starbuzz, including our
                <a href="beverages.html#house" title="House Blend">House Blend</a>,
                <a href="beverages.html#mocha" title="Mocha Cafe Latte">Mocha Cafe Latte</a>,
                <a href="beverages.html#cappuccino" title="Cappuccino">Cappuccino</a>,
                and a favorite of our customers,
                <a href="beverages.html#chai" title="Chai Tea">Chai Tea</a>.
        </p>
        <p>
                We also offer a variety of coffee beans, whole or ground, for you to
                take home with you.  Order your coffee today using our online
                <a href="form.html" title="The Bean Machine">Bean Machine</a>,
                and take the Starbuzz Coffee experience home.
        </p>

</div>

<div id="sidebar">
        <p class="beanheading">
                <img src="images/bag.gif" alt="Bean Machine bag" />
                <br />
                ORDER ONLINE
                with the
                <a href="form.html">BEAN MACHINE</a>
                <br />
                <span class="slogan">
                        FAST <br />
                        FRESH <br />
                        TO YOUR DOOR <br />
                </span>
        </p>
        <p>
            Why wait?  You can order all our fine coffees right from the Internet with our new,
            automated Bean Machine.  How does it work?  Just click on the Bean Machine link,
            enter your order, and behind the scenes, your coffee is roasted, ground
            (if you want), packaged, and shipped to your door.
        </p>
</div>

<div id="footer">
        &copy; 2005, Starbuzz Coffee
        <br />
        All trademarks and registered trademarks appearing on
        this site are the property of their respective owners.
</div>

</body>
</html>
```

Here's the for the Bean Machine. We've given it an id of "sidebar". Hmm, wonder what that could mean?

And finally, we have the that makes up the footer of the page.

And a look at the style

Let's get a good look at the CSS that styles the new Starbuzz page. Step through the CSS rules carefully. While the new Starbuzz page may look a little advanced, you'll see it's all just simple CSS that you already know.

```
body {
        background-color: #b5a789;
        font-family:      Georgia, "Times New Roman", Times, serif;
        font-size:        small;
        margin:           0px;
}

#header {
        background-color: #675c47;
        margin:           10px;
        height:           108px;
}

#main {
        background: #efe5d0 url(images/background.gif) top left;
        font-size:  105%;
        padding:    15px;
        margin:     0px 10px 10px 10px;
}

#sidebar {
        background: #efe5d0 url(images/background.gif) bottom right;
        font-size:  105%;
        padding:    15px;
        margin:     0px 10px 10px 10px;
}

#footer {
        background-color: #675c47;
        color:            #efe5d0;
        text-align:       center;
        padding:          15px;
        margin:           10px;
        font-size:        90%;
}

h1 {
        font-size: 120%;
        color:     #954b4b;
}

.slogan { color: #954b4b;}

.beanheading {
        text-align:  center;
        line-height: 1.8em;
}
```

First we just set up some basics in the body: a background color, fonts, and we also set the margin of the body to 0. This makes sure there's no extra room around the edges of the page.

Next we have a rule for each logical section. In each, we're tweaking the font size, adding padding and margins and also – in the case of main and the sidebar – specifying a background image.

Next we set up the fonts and colors on the headings.

And then some colors on the class called slogan, which is used in the sidebar . And likewise with the beanheading class, which is used there as well.

```
a:link {
        color:                  #b76666;
        text-decoration:        none;
        border-bottom:          thin dotted #b76666;
}
a:visited {
        color:                  #675c47;
        text-decoration:        none;
        border-bottom:          thin dotted #675c47;
}
```

And for the last two rules in the Starbuzz CSS we use the a:link and a:visited pseudo-classes to style the links.

Notice that we're getting a nice dotted underline effect on the links by using a dotted bottom border instead of an underline. This is a great example of using the border property on an inline element.

We're setting the border-bottom as a shortcut.

Let's take Starbuzz to the next level

Here's the goal: to turn Starbuzz Coffee into the site on the right. To do that, we need to move the Bean Machine sidebar over to the right so we've got a nice two-column page. Well, you've done this once already with the lounge, right? So, based on that, here's what you need to do:

1. Give the element you're going to float a unique name using an **id**. That's already done.
2. Make sure the element's XHTML is just below the element you want it to float under; in this case, the Starbuzz header.
3. Set a width on the element.
4. Float the element to the left or the right. It looks like you want to float it right.

Let's get started. In a few simple steps, we should have the Starbuzz CEO sending a few Chai Teas over on the house.

We've got a nice two-column look here, with discrete columns.

Move the sidebar just below the header

It's a fact of life that when you float an element, you need to move the XHTML for the element directly below the element that you want it to float below. In this case, the sidebar needs to come under the header. So, go ahead and locate the sidebar ** ** in your editor and move the entire ** ** to just below the header ** **. You'll find the XHTML in the file "index.html" in the "chapter12/starbuzz" folder. After you've done that and saved, reload the page.

Now the sidebar should be on top of the main content area.

Set the width of the sidebar and float it

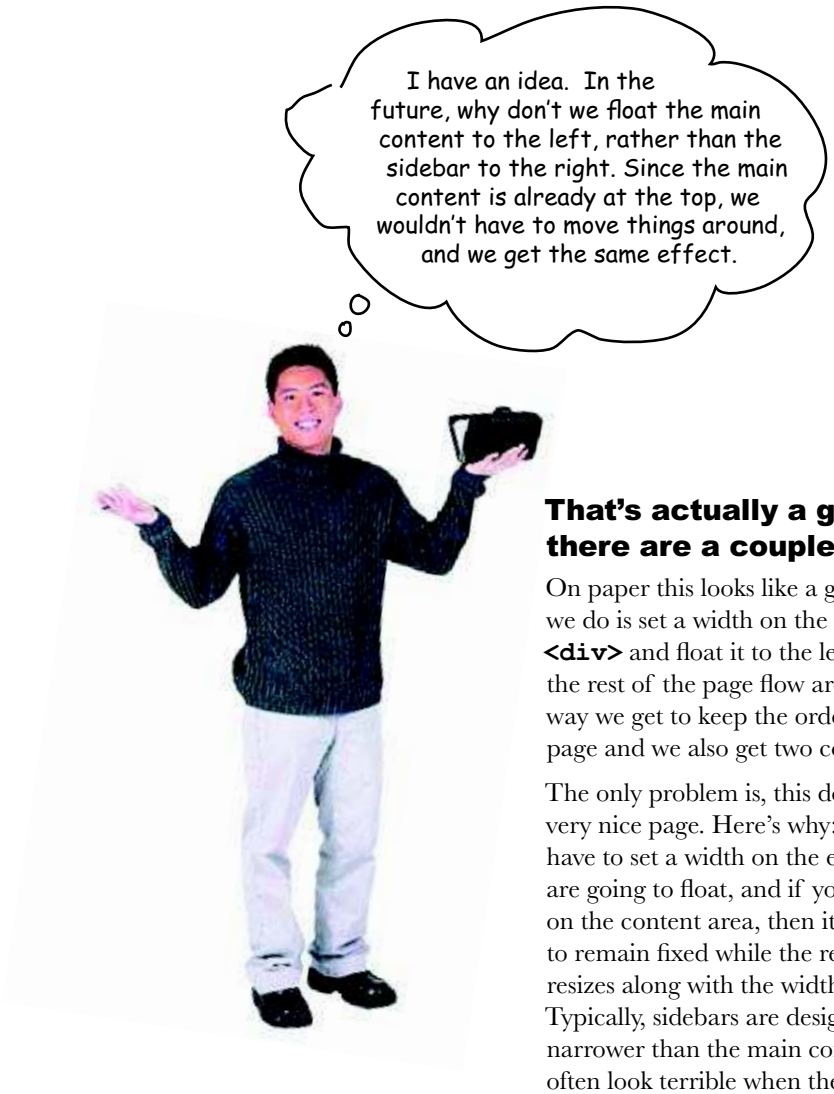
Let's set the width of the sidebar to 280 pixels. And to float the sidebar, add a float property, like this:

We're using an id selector to select the element with the id "sidebar", which we know is the for the sidebar.

```
#sidebar {
        background: #efe5d0 url(images/background.gif) bottom right;
        font-size:  105%;
        padding:    15px;
        margin:     0px 10px 10px 10px;
        width:      280px;
        float:      right;
}
```

We're setting the width of the content area to 280 pixels.

And then we're floating the sidebar to the right. Remember, this moves the sidebar as far right as possible below the header, and it also removes the sidebar from the normal flow. Everything else below the sidebar in the XHTML is going to move up and wrap around it.



That's actually a great idea, but there are a couple of issues.

On paper this looks like a great idea. What we do is set a width on the main content ** ** and float it to the left, and then let the rest of the page flow around it. That way we get to keep the ordering of the page and we also get two columns.

The only problem is, this doesn't result in a very nice page. Here's why: remember, you have to set a width on the element that you are going to float, and if you set a width on the content area, then its width is going to remain fixed while the rest of the page resizes along with the width of the browser. Typically, sidebars are designed to be narrower than the main content area, and often look terrible when they expand. So, in most designs, you want the main content area to expand, not the sidebar.

But we are going to look at a way to use this idea that works great. So hang on to this idea. We'll also talk a little more about why you'd even care what order your sections are in.

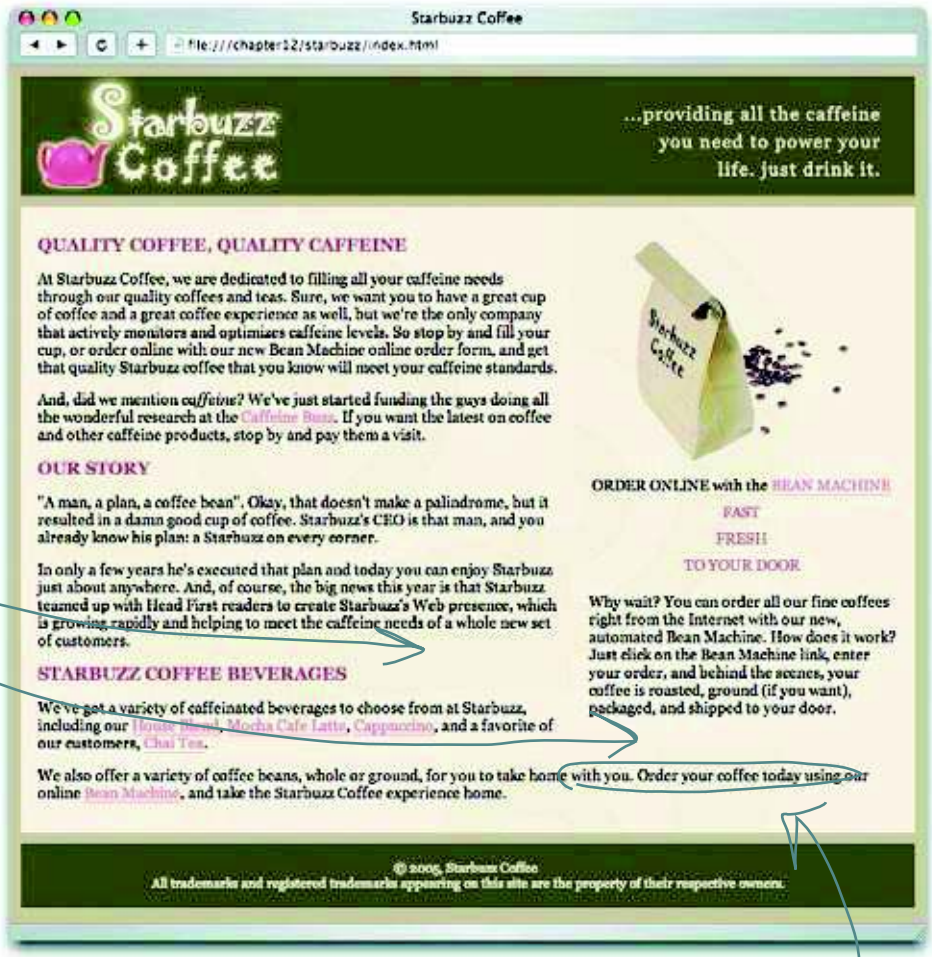
Test driving Starbuzz

Make sure you add the new sidebar properties to the "starbuzz.css" file in the "chapter12/starbuzz" folder, and then reload the Starbuzz page. Let's see what we've got...

Hmm, this looks pretty good, but if you flip back three pages you'll see we're not quite where we want to be.

The main content and the sidebar are on the left and the right, but they don't really look like two columns yet.

Look at how the background images of the two sections just run together. There's no separation between the columns.

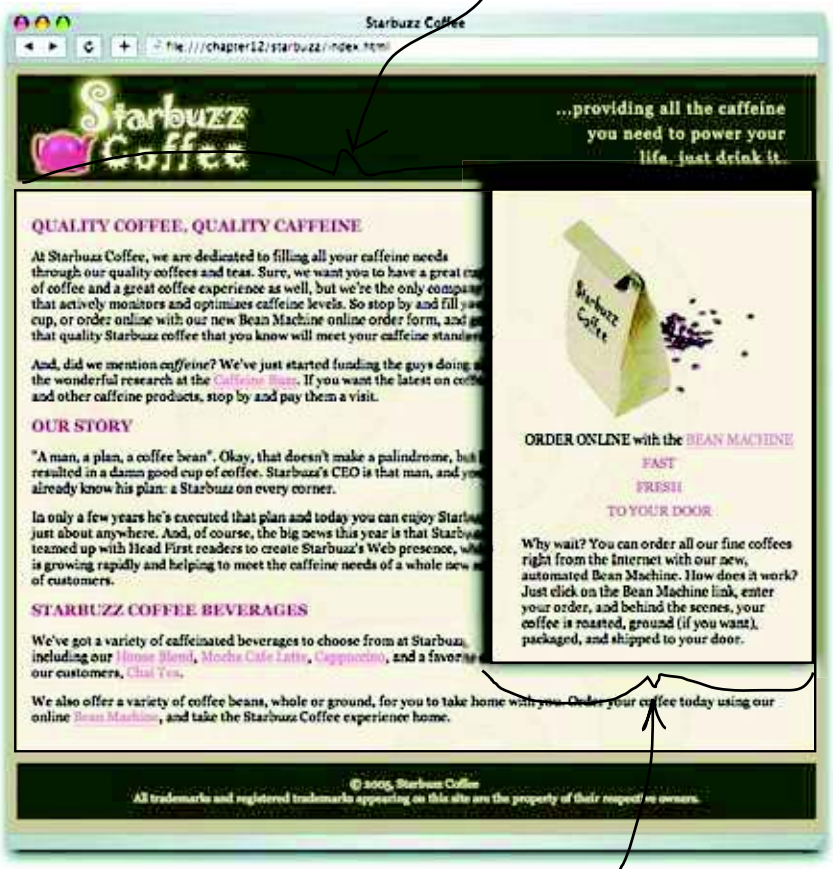


And the text wraps around and under the sidebar, which doesn't make this look like two columns either. Hmm, that is actually how the lounge worked too – maybe we should have expected that.

Fixing the two-column problem

Are you sitting there waiting for us to come riding in on a white horse with the magic property that solves all this? Well, that's not going to happen. This is the point in CSS where page layout becomes more an art – or at least a set of techniques – than a set of properties that can solve every problem. So, what we're going to do is solve this using a common technique that is widely used. It's not perfect, as you'll see, but in most cases it gives you good results. And after this, you're going to see a few other ways to approach the same two-column problem. What's important here is that you understand the techniques, and why they work, so you can apply them to your own problems, and even adapt them where necessary.

The first thing to remember is that the sidebar is floating on the page. The main content area extends all the way under it.



So, what if we give the main content area a right margin that is at least as big as the sidebar? Then its content will extend almost to the sidebar, but not all the way.

Then we'll have separation between the two, and since margins are transparent and don't show the background image, the background color of the page itself should show through. And that's what we're looking for (flip back a few pages and you'll see).

Let's make the margin as wide as the sidebar.

Sharpen your pencil

What we want to do is set a right margin on the main content section so that it's the same width as the sidebar. But how big is the sidebar? Well, we hope you aren't already rusty since the last chapter. Here's all the information you need to compute the width of the sidebar. Check your answer in the back of the chapter.

```
#sidebar {
        background: #efe5d0 url(images/background.gif) bottom right;
        font-size:  105%;
        padding:    15px;
        margin:     0px 10px 10px 10px;
        width:      280px;
        float:      right;
}
```

You'll find everything you need to compute the width of the sidebar in this rule.

Setting the margin on the main section

The width of the sidebar is 330 pixels, and that includes 10 pixels of left margin on the sidebar, which will provide the separation we need between the two columns (what the publishing world calls a "gutter"). Add the 330 pixel right margin to the **#main** rule in your "starbuzz.css" file, like we've done below:

```
#main {
        background: #efe5d0 url(images/background.gif) top left;
        font-size:  105%;
        padding:    15px;
        margin:     10px;
        margin:     0px 330px 10px 10px;
}
```

We're changing the right margin to 330 pixels to match the size of the sidebar.

Test drive

As usual, save your "starbuzz.css" file and then reload "index.html". You should now see a nice gutter between the two columns. Let's think through how this is working one more time. The sidebar is floating right, so it's been moved as far to the right as possible, and the whole ** ** has been removed from the normal flow and is floating on top of the page. Now the main content ** ** is still taking up the width of the browser (because that's what block elements do), but we've given it a margin as wide as the sidebar to reduce the width of the content area. The result is a nice two column look. You know the box of the main ** ** still goes under the sidebar, but we won't tell anyone if you don't.

By expanding the margin of the main , we're creating the illusion of a two column layout, complete with a gutter in between.

We've got a problem. When you resize your browser to a wide position, the footer and the sidebar start to overlap.

Uh oh, we have another problem

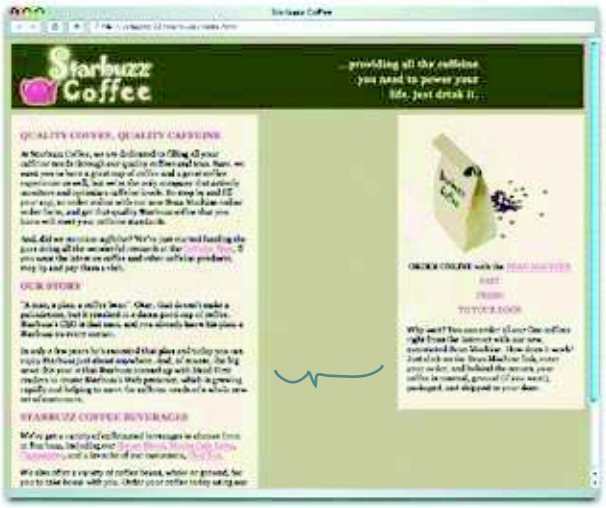
As you were test driving the page you might have noticed a little problem. If you resize the browser to a wide position, the footer comes up underneath the sidebar. Why? Well, remember, the sidebar is not in the flow, so the footer pretty much ignores it, and when the content area is too short, the footer moves right up. We could use the same margin trick on the footer, but then the footer would only be under the content area, not the whole page. So, what now?

Wait a sec. Before you get way into solving that problem, I have to ask, why did we have to go to all this trouble of using a margin? Why don't we just set the width of the main area? Wouldn't that do the same thing?

This is another solution that sounds good... until you try it.

The problem with setting a width on both the content area and the sidebar is that this doesn't allow the page to expand and contract correctly because both have fixed widths. Check the screen-shots below that show how it works (or rather, doesn't work).

But this is good. You're thinking in the right ways, and a little later in the chapter we're going to come back to this idea when we talk about "liquid versus fixed" layouts. There are ways to make your idea work if we lock a few other things down first.

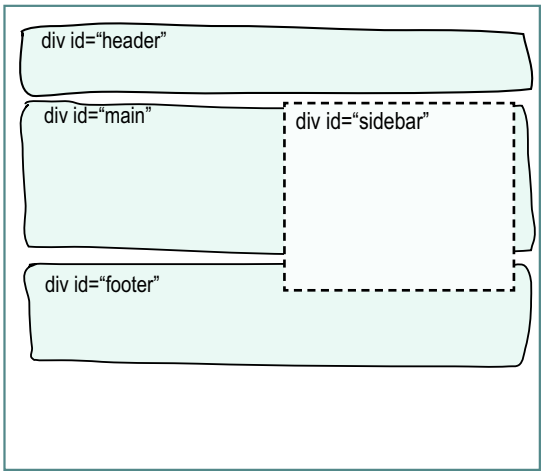


When the browser is wide, the two totally separate.

And when the browser window is made small, the two start to overlap.

Back to clearing up the overlap problem

Guess what, this time *we are* going to ride in on a white horse with a solution, but don't get used to it. The solution is called the **clear** property, and here's how it works...



Here's what we've got now. The "main" is short enough that the footer is coming right up and overlapping with the sidebar .

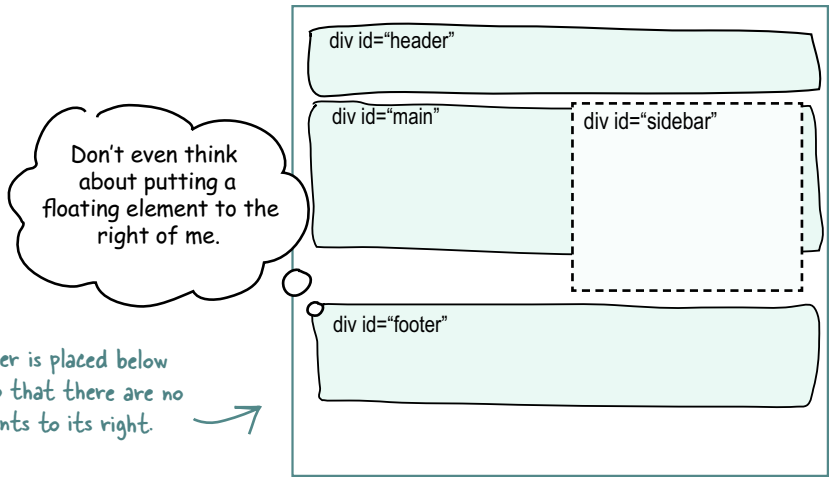
This happens because the sidebar has been pulled out of the flow. So, the browser just lays out the main and footer s like it normally would, ignoring the sidebar (although remember that when the browser flows inline elements, it will respect the borders of the sidebar and wrap inline elements around it).

We can solve this problem with the CSS **clear** property. You can set this property on an element to request that as the element is being flowed onto the page, no floating content is allowed to be on the left, right, or both sides of the element. Let's give it a try...

```
#footer {
        background-color: #675c47;
        color:            #efe5d0;
        text-align:       center;
        padding:          15px;
        margin:           10px;
        font-size:        90%;
        clear:            right;
}
```

Here we're adding a property to the footer rule, which says that no floating content is allowed on the right of the footer.

Now when the browser places the elements on the page, it looks to see if there is a floating element to the right side of the footer, and if there is, it moves the footer down until there is nothing on its right. Now, no matter how wide you open the browser, the footer will always be below the sidebar.



Now the footer is placed below the sidebar so that there are no floating elements to its right.

Test drive

Go ahead and add the clear property to your "starbuzz.css" file in the footer rule, and then reload "index.html". You'll see that when the screen is wide, the footer now stays below the sidebar.

There are other improvements we could think about making to this page, like having each column come down to meet the footer. As it is now, there is a gap either between the main content and the footer (if the browser window is set wide), or the sidebar and the footer (if the browser is set to a normal width). Unfortunately, it's not easy to fix this, and we're not going to try to do that in this chapter. Layout in CSS is an art, and no layout solution is perfect. When done right, layout with CSS gives you a better look for your Web page, while still allowing the page to look reasonably good in browsers that don't have as much (or any) support for CSS.

We will take a look at a few more ways to layout your pages using CSS beyond using float. There are many ways to do things in CSS, each with their own strengths and weaknesses.

Now our footer problems are solved. The footer will always be below the sidebar, no matter how narrow or wide the browser.

there are no Dumb Questions

Q: So why isn't there just a two-column property in CSS? Why is it so hard to get this stuff to work correctly?

A: Yes, we have a winner! You've asked the $64,000 question. But, more seriously, while it seems like CSS *should* have some way of specifying "give me two columns, dammit!", you have to keep in mind the whole purpose of XHTML and CSS. Remember, XHTML is meant to be a format for structure and content that can be styled by CSS but should be viewable on any device, even if the CSS isn't used. So, it's really no surprise that CSS isn't the end-all-be-all of document presentation, and, if that's what we wanted, we'd probably all just be using Microsoft Word. But CSS does give you some nice tools to create layouts that are attractive and usable, and does a good job of gracefully degrading in less than optimal viewing conditions.

Q: Can I float to the center?

A: No, CSS only allows you to float an element to the left or right. But if you think about it, if you were to float to the center, then the inline content under the floated element would have to be flowed around both sides of your element. While that might be doable, it probably wouldn't be very readable or attractive.

Q: Do margins collapse on floated elements?

A: No, they don't, and it's pretty easy to see why. Unlike block elements that are flowed on the page, floated elements are just, well, floating. In other words, the margins of floated elements aren't actually touching the margins of the elements in the normal flow, so they can't be collapsed.

But this raises a good point, and identifies a common error in layouts. If you have a main content area and a sidebar, it is common to set a top margin on each. Then, if you float the sidebar, it still has a margin, and that margin won't be collapsed with whatever is above it anymore. So you can easily end up having different margins on the sidebar and on the main content if you don't remember that floated elements don't collapse margins.

Q: Can I float an inline element?

A: Yes, you sure can. The best example – and a common one – is to float images. Give it a try – float an image left or right in a paragraph and you'll see your text flow around it. Don't forget to add padding to give the image a little room, and possibly a border. You can also float any other inline element you like, but it's not commonly done.

Q: Is it correct to think about floated elements as elements that are ignored by block elements, while inline elements know they are there?

A: Yes, that's a good way of thinking about it. Inline content nested inside a block element always flows around a floated element, observing the boundaries of the floated element, while block elements are flowed onto the page as normal. The exception is when you set the clear property on a block element, which causes a block element to move down until there are no floating elements next to it on the right, left or both sides, depending on the value of clear.

Right. That happens because of the way we ordered the s.

This is one of the disadvantages of the way we've designed this page – because we need the sidebar to be located just under the header and before the main content, anyone using a browser with limited capabilities (PDAs, mobile phones, screen readers, and so on) will see the page in the order it is written, with the sidebar first. However, most people would rather see your main content before any kind of sidebar or navigation.

So, let's look at another way of doing this, which goes back to your idea of using float "left" on the main content. While we're exploring that, we'll talk about *liquid* versus *frozen* designs as well.

Look Ma, no CSS!

Want to know how your pages are going to look to your users under bad conditions (like on a browser that doesn't support CSS)? Then open your "index.html" file and remove the <link> from the <head>, save, and reload the page in your browser. Now you can see the real order things will be seen in (or heard from a screen reader). Go ahead and give it a try. Just make sure you put it back when you're done (after all, this is a chapter on CSS).

Here's the Starbuzz page without CSS. For the most part we're in good shape. It is still very readable, although the Bean Machine does come before the main content, which probably isn't what we want.

Righty tighty, lefty loosey

Let's get the Starbuzz page switched over so that the main content is floating left. We'll check out how that works, and then move on to make it *really* work. You're going to see the mnemonic righty tighty, lefty loosey holds true in the CSS world too... well, for our sidebar, anyway. Here's how we convert the page over... just a few simple steps.

Step One: start with the sidebar

We're basically swapping the roles of the sidebar and the main content area. The content area is going to have a fixed width and float, while the sidebar is going to wrap around the content. We're also going to use the same margin technique to keep the two visually separate. But before we start changing CSS, go to your "index.html" file and move the "sidebar" ** ** down below the "main" ** **. After you've done that, here are the changes you need to make to the sidebar CSS rule:

```
#sidebar {
        background:   #efe5d0 url(images/background.gif) bottom right;
        font-size:    105%;
        padding:      15px;
        margin:       0px 10px 10px 470px;
        width:        280px;
        float:        right;
}
```

Because the sidebar is now going to flow under the main content, we need to move the large margin to the sidebar. The total width of the main content area is 470 pixels. (Go ahead and compute that yourself in all that free time you have. Compute it in the same way as you did for the sidebar. You should know that we're going to set the width of the main content area to 420 pixels.)

We're setting a fixed width on the main content , so delete the sidebar width property along with the float.

Step Two: take care of the main content

Now we need to float the main ** **. Here's how to do it:

```
#main {
        background: #efe5d0 url(images/background.gif) top left;
        font-size:    105%;
        padding:      15px;
        margin:       0px 10px 10px 10px;
        width:        420px;
        float:        left;
}
```

We're changing the right margin from 330 pixels back to 10 pixels.

We need to set an explicit width because we're going to float this element. Let's use 420 pixels.

We're going to float the main to the left.

Step Three: take care of the footer

Now, we just need to adjust the footer to clear everything to the left, rather than the right.

```
#footer {
        background-color: #675c47;
        color:            #efe5d0;
        text-align:       center;
        padding:          15px;
        margin:           10px;
        font-size:        90%;
        clear:            left;
}
```

Change the clear property to have a value of left, rather than right. That way the footer will stay clear of the main content area.

A quick test drive

We've already said there might be a few problems with this method of floating the content to the left. Do a quick test drive before you move on just to see how this is all working. Go ahead and make the changes to your "starbuzz.css" file and then reload "index.html" in your browser. Get a good feel for how the page performs when it is resized to narrow, normal, and wide.

Actually, this looks pretty good, and we have the s in the right order now. But it's not great that the sidebar expands; it looks a lot better fixed. Sidebars are often used for navigation and they don't look very good when expanded.

When we float the sidebar right, then the design stays nice and tight, allowing the content to expand, but if we float the main content to the left, the design feels too loose, allowing the sidebar to expand.

BRAIN POWER

Design-wise, the first design worked better, while information-wise, the second works better (because of the placement of the s). Is there a way we can have the best of both worlds: have the sidebar at a fixed width, but the main still first in the XHTML? What design tradeoffs could we make to get there?

Liquid and Frozen Designs

All the designs we've been playing with so far are called *liquid layouts* because they expand to fill whatever width we resize the browser to. These layouts are useful because, by expanding, they fill the space available and allow users to make good use of their screen space. Sometimes, however, it is more important to have your layout locked down so that when a user resizes the screen, your design still looks as it should. There are a couple of layouts that work like this, but let's start with *frozen layouts*. Frozen layouts lock the elements down, frozen to the page, so they can't move at all, and so we avoid a lot of issues that are caused by the window expanding. Let's give a frozen layout a try.

Going from your current page to a frozen page only requires one addition to your XHTML, and one new rule in your CSS.

XHTML Changes

In your XHTML you're going to add a new ** ** element with the id "allcontent". Like its name suggests, this ** ** is going to go around all the content in your page. So place the opening ** ** tag before the header ** ** and the closing tag below the footer ** **.

```
<body>
   <div id="allcontent">
      <div id="header">
        ... rest of the XHTML goes here ...
      </div>
   </div>
</body>
```

Add a new with the id of "allcontent" around all the other elements in the <body>.

This closes the footer .

CSS Changes

Now we're going to use this ** ** to constrain the size of all the elements and content in the "allcontent" ** ** to a fixed width of 800 pixels. Here's the CSS rule to do that:

```
#allcontent {
        width:            800px;
        padding-top:      5px;
        padding-bottom:   5px;
        background-color: #675c47;
}
```

We're going to set the width of "allcontent" to 800 pixels. This will have the effect of constraining everything in it to fit within 800 pixels.

While we're at it, since this is the first time we're styling this , let's add some padding and give it its own background color. You'll see this helps to tie the whole page together.

The outer "allcontent" is always 800 pixels, even when the browser is resized, so we've effectively frozen the to the page, along with everything inside it.

A frozen test drive

Go ahead and add this rule to the bottom of "starbuzz.css", and then reload "index.html". Now you can see why we call it a frozen layout. It doesn't move when the browser is resized.

Now the "allcontent" is 800 pixels in width, no matter how you resize the browser. And, because all the other s are inside "allcontent", they will also fit into the 800 pixel space as well. So, the page is basically frozen to 800 pixels.

This certainly solves the problem of the sidebar expanding and it looks pretty nice. It is a little strange when the browser is very wide, though, because of all the empty space on the right side.

But, we're not done yet; we've got a little room for improvement.

What's the state between liquid and frozen? Jello, of course!

The frozen layout has some benefits, but it also just plain looks bad when you widen the browser. But we've got a fix, and it's a common design you'll see on the Web. This design is between frozen and liquid, and it has a name to match: *Jello*. Jello layouts lock down the width of the content area in the page, but center it in the browser. It's actually easier to change the layout to a jello layout and let you play with it, than to explain how it behaves, so let's just do it:

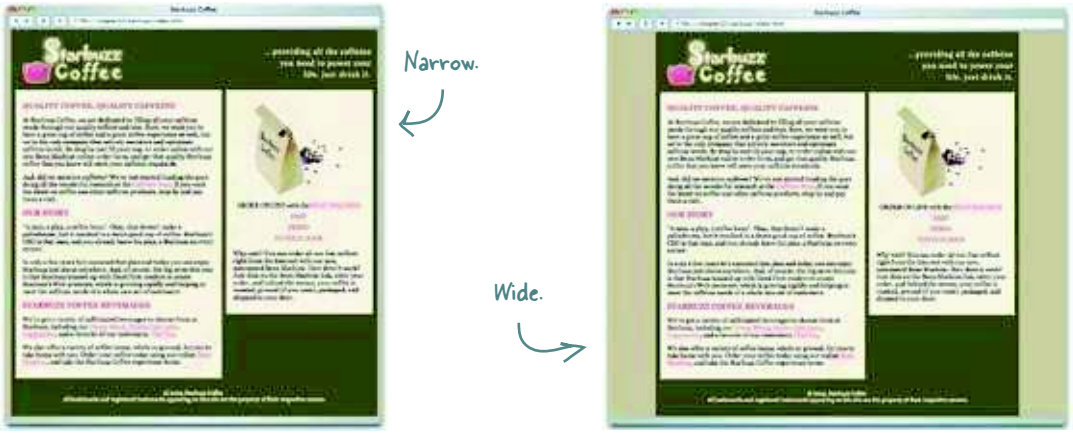
Rather than having fixed left and right margins on the "allcontent" , we're setting the margins to "auto".

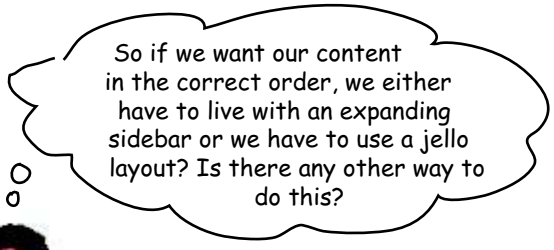
```
#allcontent {
        width: 800px;
        padding-top:    5px;
        padding-bottom: 5px;
        background-color: #675c47;
        margin-left:  auto;
        margin-right: auto;
}
```

If you remember, when we talked about giving a content area a width of "auto", the browser expanded the content area as much as it needed to. With "auto" margins, the browser figures out what the correct margins are, but also makes sure the left and right margins are the same, so that the content is centered.

Test driving with a tank of jello

Add the two margin properties to your "starbuzz.css" file, and then reload the page. Now play with the size of the browser. Pretty nice, huh?





With CSS, there are typically lots of ways to approach a layout, each with their own strengths and weaknesses. Actually, we're just about to look at another common technique for creating a two-column layout that keeps the content in the correct order, and avoids some of the problems of the liquid layouts. But, as you'll see, there are some tradeoffs.

With this new technique we're not going to float elements at all. Instead we're going to use a feature of CSS that allows you to precisely *position* elements on the page. It's called *absolute positioning*. You can also use absolute positioning for some nice effects beyond just multi-column layouts, and we'll look at an example of that, too.

To do all this, we're going to step back to the original XHTML and CSS we started with in the beginning of this chapter. You can find a fresh copy of these files in the "chapter12/absolute" folder. Be sure and take another look at these files so you remember what they originally looked like. Recall that we've got a bunch of ** **s: one for the header, one for main, one for the footer, and also a sidebar. Also remember that in the original XHTML, the sidebar ** ** is below the main content area, where we'd optimally like to have it.

How absolute positioning works

Let's start by getting an idea of what absolute positioning does, and how it works. Here's a little CSS to position the sidebar ** ** with absolute positioning. Don't type this in just yet; right now we just want you to get a feel for how this works:

The first thing we do is use the position property to specify that the element will be positioned absolutely.

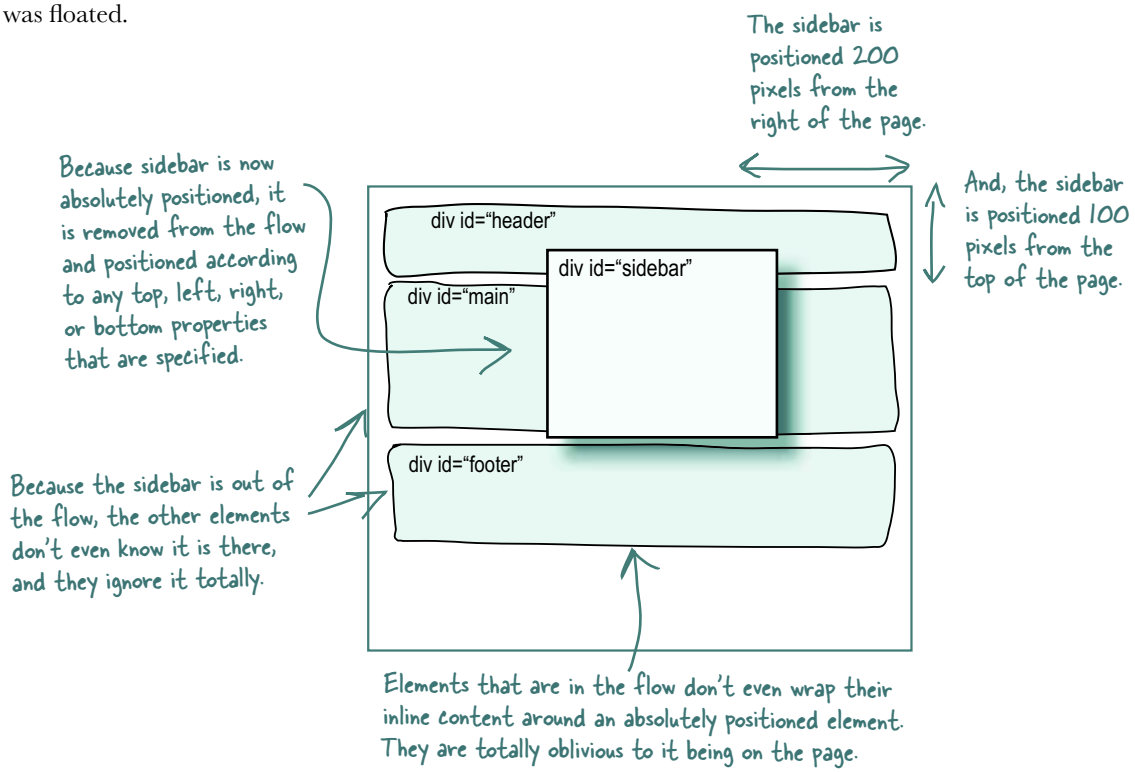
```
#sidebar {
    position: absolute;
    top:    100px;
    right:  200px;
    width:  280px;
}
```

Next we set top and right properties.

And we also give the a width.

What the CSS does

Now let's look at what this CSS does. When an element is absolutely positioned, the first thing the browser does is remove it completely from the flow. The browser then places the element in the position indicated by the **top** and **right** properties (you can use **bottom** and **left** as well). In this case, the sidebar is going to be 100 pixels from the top of the page, and 200 pixels from the right side of the page. We're also setting a width on the ** **, just like we did when it was floated.



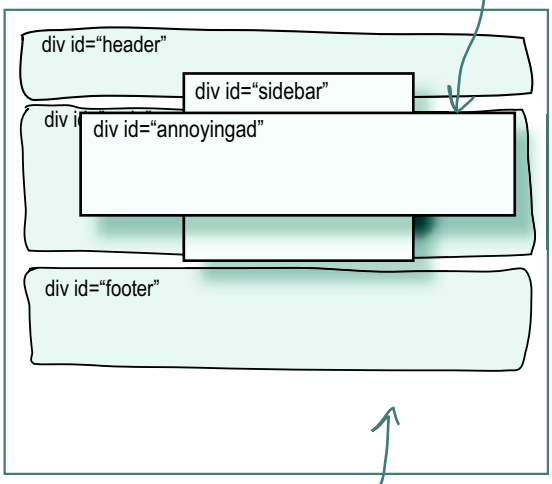
Another example of absolute positioning

Let's look at another example. Say we have another ** ** with the **id** "annoyingad". We could position it like this:

```
#annoyingad {
    position: absolute;
    top:    150px;
    left:   100px;
    width:  400px;
}
```

The annoying ad is being positioned 100 pixels from the left, and 150 pixels from the top. It's also a bit wider than the sidebar, at 400 pixels.

Now we have a second , positioned absolutely, about 100 pixels from the left and 150 pixels from the top.



Notice the annoyingad is on top of the sidebar .

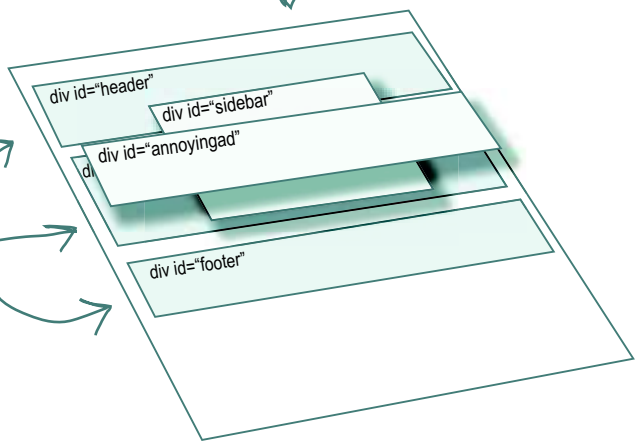
Just like with the sidebar, we've placed the annoying ad ** ** at a precise position on the page. Any elements underneath that are in the normal flow of the page don't have a clue about the absolutely positioned elements layered overhead. This is a little different from floating an element, because elements that were in the flow adjusted their inline content to respect the boundaries of the floated element. But absolutely positioned elements have no effect whatsoever on the other elements.

Who's on top?

Another interesting thing about absolutely positioned elements is that you can layer them on top of each other. But if you've got a few absolutely positioned elements at the same position in a page, how do you know the layering? In other words, who's on top?

Each positioned element has a property called a **z-index** that specifies its placement. You'll see how to specify a z-index in just a few pages.

The sidebar and annoyingad s are layered on the page, with the annoyingad having a greater z-index than the sidebar, so it's on top.



The header, main, and footer s are all in the flow, and flat on the page.

there are no Dumb Questions

Q: What is the position property set to by default?

A: The default value for positioning is "static". With static positioning the element is placed in the normal document flow and isn't positioned by you – the browser decides where it goes. You can use the float property to take an element out of the flow, and you can say it should float left or right, but the browser is still ultimately deciding where it goes. Compare this to the "absolute" value for the position property. With absolute positioning, you're telling the browser exactly where to position elements.

Q: Can I only position s?

A: You can absolutely position any element, block or inline. Just remember that when an element is absolutely positioned, it is removed from the normal flow of the page.

Q: So, I can position an inline element?

A: Yes, you sure can. For instance, it's common to position the <img> element. You can position s, s, and so on as well, but it isn't common to do so.

Q: Are there position property values other than static and absolute?

A: There are actually four: static, absolute, fixed, and relative. You've already heard about static and absolute. Fixed positioning places an element in a location that is relative to the browser window (rather than the page), so fixed elements never move. You'll see an example of fixed positioning in a few pages. Relative positioning takes an element and flows it on the page just like normal, but then offsets it before displaying it on the page. Relative positioning is commonly used for more advanced positioning and special effects. You're going to see an example of that too.

Q: Do I have to specify a width for an absolutely positioned element just like the floated elements?

A: No, you don't have to specify a width for absolutely positioned elements. But if you don't, by default, the block element will take up the entire width of the browser, minus any offset you specify from the left or right. This might be exactly what you want, or it might not. So set the value of the width property if you want to change this default behavior.

Q: Do I have to use pixels for positioning?

A: No – another common way to position elements is using percentages. If you use percentages, the positions of your elements may appear to change as you change the width of your browser. So, for example, if your browser is 800 pixels wide, and your element's left position is set to 10%, then your element will be 80 pixels from the left of the browser window. But if your browser is resized to 400 pixels wide, then the width will be reduced to 10% of 400 pixels, or 40 pixels from the left of the browser window.

Another common use for percentages is in specifying widths. If you don't need specific widths for your elements or margins, then you can use percentages to make both your main content area and your sidebars flexible in size. You'll see this done a lot in two- and three-column layouts.

Q: Do I have to know how to use z-indexes to use absolute positioning?

A: No, z-indexes tend to be used most often for various advanced uses of CSS, especially when Web page scripting is involved, so they're a little beyond the scope of this book. But they are a part of how absolute positioning works, so it's good to know about z-index (and in fact, you'll see a case where knowing about z-index is important in just a bit).

Using absolute positioning

We're now going to create a two column Starbuzz page using techniques similar to those we used with the float version of the page; however, this time we'll use absolute positioning. Here's what we're going to do:

1. First we're going to make the sidebar ** ** absolutely positioned. In fact, we're going to position it in exactly the same place that we floated it to before.
2. Next, we're going to give the main content another big margin so that the sidebar can sit on top of the margin space.
3. Finally, we're going to give this a good testing and see how it compares to the float version.

Changing the Starbuzz CSS

Our XHTML is all ready to go, and the sidebar ** ** is right where we want it (below the important main content). All we need to do is make a few CSS changes and we'll have a sidebar that is absolutely positioned. Open your "starbuzz.css" file and let's make a few changes to the sidebar:

Remember, we are going back to the original versions of the files, which you can find in the "chapter12/absolute" folder.

You can work out of the "absolute" folder, or copy the files "index.html" and "starbuzz.css" into the "starbuzz" folder and work from there, like we did.

```
#sidebar {
    background: #efe5d0 url(images/background.gif) bottom right;
    font-size:  105%;
    padding:    15px;
    margin:     0px 10px 10px 10px;
    position:   absolute;
    top:        128px;
    right:      0px;
    width:      280px;
}
```

Okay, now we're going to specify that the sidebar is absolutely positioned 128 pixels from the top, and 0 pixels from the right of the page. We also want the sidebar to have a width, so let's make it the same as the float version: 280 pixels.

You'll see where the "128" came from in a sec...

0 pixels from the right will make sure that the sidebar sticks to the right side of the browser.

Now we just need to rework the main

Actually, there's not much reworking to be done. We're just adding a margin like we did with the float version. So, change the right margin of the main ** ** to be 330 pixels, like you did last time.

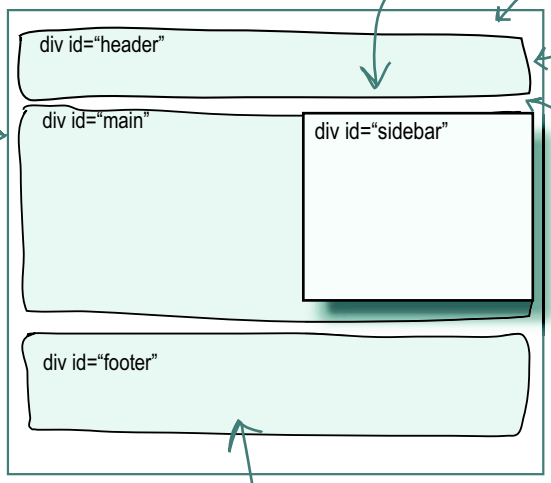
```
#main {
    background: #efe5d0 url(images/background.gif) top left;
    font-size:  105%;
    padding:    15px;
    margin:     0px 330px 10px 10px;
}
```

We're going to give the sidebar some space to be positioned over by giving the main a big margin. This is really the same technique we used with the float. The only difference is the way the sidebar is being placed over the margin.

Okay, all you need to do is make that change to your margin, and then save. But, before we take this for a test drive, let's think about how this is going to work with the absolutely positioned sidebar.

We're positioning the sidebar to be 128 pixels from the top, and up against the right side of the page. Keep in mind, the sidebar has 10 pixels of margin on the right, so the background color will show through that like before.

The main is flowed just below the header, so it will align with the top of the sidebar. Also, it has a right margin that is the same size as the sidebar, so all its inline content will be to the left of the sidebar. Remember that the flowed elements don't know about the absolutely positioned elements at all, so the inline content in the flowed elements doesn't wrap around the absolutely positioned elements.



10 pixel top margin.

108 pixels for the header. You can see this height set in the CSS.

10 pixel bottom margin.

The sidebar needs to be 128 pixels from the top because that's exactly how much room the header takes up, including margins.

You might want to think about what happens to the footer. Because flowed elements don't know about absolute elements, we can't use "clear" anymore.

Time for the absolute test drive

Make sure you've saved the new CSS and then reload "index.html" in your browser. Let's check out the results:

Wow, this looks amazingly like the float version; however, you know that the sidebar is being positioned absolutely.

As you resize the browser, the sidebar always sits 128 pixels from the top, and sticks to the right of the page.

And the sidebar has a 10 pixel right margin, so it has spacing between it and the edge of the page.

The main content area has a margin that is exactly the width of the sidebar, and the sidebar sits on top of that space.

And we've still got a nice gutter between the two columns.

But we are now back to having a problem with the footer. When the browser gets wide enough, the absolutely positioned sidebar comes down over the top of the footer. Unfortunately, we can't fall back on the **`clear`** property this time, because flowed elements ignore the presence of absolutely positioned elements.

When the browser is wide, the vertical space of the main content area is reduced, and the sidebar can come down over the footer.

What can we do? Or, can't you just tell me how to do a two-column layout that doesn't break?

Okay, you know that one of the big motivations behind CSS was to separate structure from style. Right? And CSS does a great job of allowing you to create XHTML documents that can be used in a lot of different browsers (even screen readers or text-only browsers) without having unnecessary style embedded into the XHTML. But this also means that CSS is not meant to be a full-blown page layout language. Rather, it gives you some interesting tools that you can use to arrange and position elements in XHTML documents. Depending on the environment the page is viewed in, your mileage may vary in terms of what the end result is. If you resize your browser to be ultra wide, well, then the layout may break.

So where does this leave you? In this chapter we've looked at several techniques for creating two-column layouts. None of them were perfect and, in fact, they all had various tradeoffs. Let's quickly review the various examples.

The Floating Layout

Ahh, how cute, remember your first two-column Starbuzz page? You used a **`float`** property along with a **`clear`** on the footer and life was good. The only problem is that this solution often results in putting your content in an order your users might not appreciate if they are using another kind of browser, like a screen reader that reads the content aloud to the user.

The Jello Layout

First we created a frozen layout by wrapping a fixed size **`<div>`** around all the content in the page, and then we made it jello by allowing the margins to expand with the "auto" property value. This makes for a great looking layout, and lots of pages on the Web use this design. This also solved the problem of our content ordering. The disadvantage here is that the content doesn't expand to fill the entire browser window (which many people don't find to be a disadvantage at all).

The Absolute Layout

Finally, we were on a mission to have a liquid layout and yet have the content ordered like we wanted. So, we used absolute positioning and actually achieved our goal. But, there was a downside: since you can't use the **`clear`** property with absolute elements, the footer creeps up under the sidebar when the browser is wide.

So are we done yet? Maybe. If one of these designs meets your needs, great, go with it. For instance, lots of people are very happy with jello layouts. But there is always more tweaking you can do to perfect your particular layout. For instance, take the absolute design. Can we fix the footer? Not really, but we can make a tradeoff. Your design might be fine if the footer only showed under the main content area. If that's the case, then we *can* fix the footer problem. Let's give that a quick try.

One tradeoff you can make to fix the footer

To try this solution, just give the footer the same right margin size as the main content area, like this:

```
#footer {
        background-color: #675c47;
        color:            #efe5d0;
        text-align:       center;
        padding:          15px;
        margin:           10px 330px 10px 10px;
        font-size:        90%;
}
```

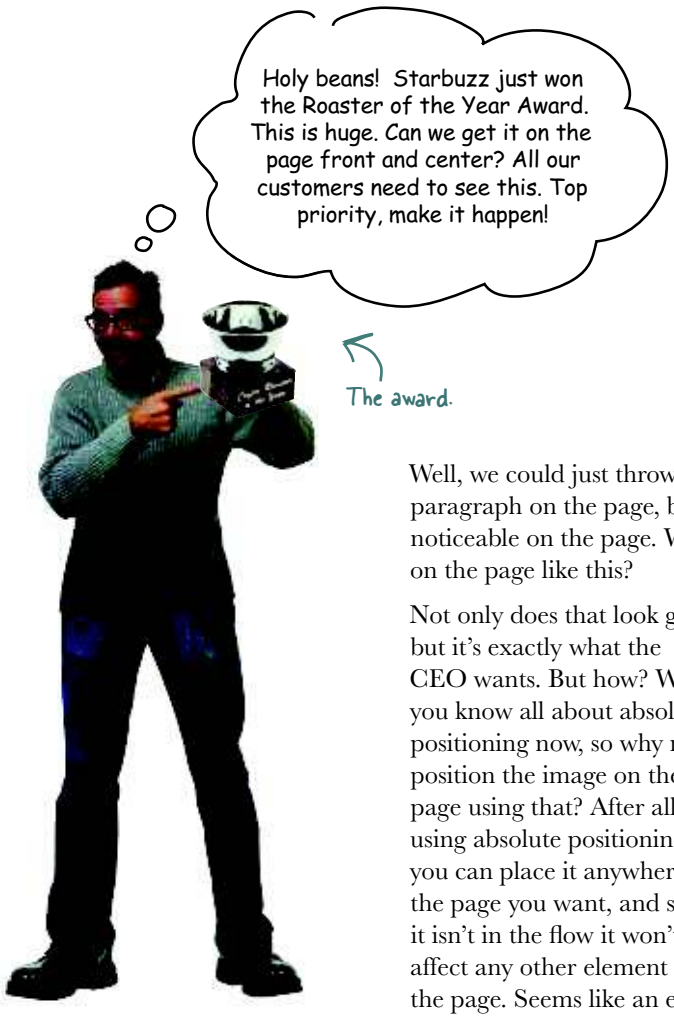
If you save this and reload your page, you'll see that the footer is now under the main content area only, and never creeps up under the sidebar. Is this optimal? No, but it's also not bad. And, as we've said, doing CSS layout is a bit of an art. To do layout, you need to *experiment*, *explore*, and keep an eye on the layouts others are creating with CSS (you'll find some references for good CSS hang-outs at the end of the chapter).

This solution trades having the footer under just the main content for having a footer under the entire page, to keep it from creeping up under the sidebar.

In terms of whether you want your layout to be liquid or frozen or jello, that really is a matter of deciding what works best for your pages. For some pages, a fixed content area size with expandable margins works great, and in fact can be more readable on wide browsers. For other uses, you might want to use as much of the browser as you can. So, decide what works best for you.

Once you've decided, you still need to figure out which method you're going to use to create your pages (float? absolute? some combination?). You've already learned the basics, so now it is time to start exploring, as there are many other approaches out there, and new ones being created every day. The techniques you've learned in this chapter are often used as the basis for more sophisticated designs.

You should know that, in general, using float is considered the most flexible solution for multi-column layouts. Just keep in mind, you may have to be careful in the order of your content, depending on the design.



Well, we could just throw this as an image into any old paragraph on the page, but the CEO really wants this to be noticeable on the page. What if we could place the award on the page like this?

Not only does that look great, but it's exactly what the CEO wants. But how? Well, you know all about absolute positioning now, so why not position the image on the page using that? After all, by using absolute positioning you can place it anywhere on the page you want, and since it isn't in the flow it won't affect any other element on the page. Seems like an easy addition to make to the page without disrupting what's already there.

Let's give it a try. Start by adding a new ** **, just below the header (the CEO thinks this is pretty important, so it should be up high in the order of content). Here's the ** **

```
<div id="award">
    <img src="images/award.gif"
         alt="Roaster of the Year award" />
</div>
```

The contains the image of the award.

Positioning the award

We want the award to sit just about in the middle of the page when the browser's open to 800 pixels (the width of the image in the header, and a typical size for browser widths) and just overlapping the main content ** **.

So we're going to use the top and left properties to position the award 30 pixels from the top, and 365 pixels from the left.

```
#award {
    position: absolute;
    top:      30px;
    left:     365px;
}
```

We're using an absolute position for the award that is 30 pixels from the top and 365 pixels from the left.

Add this CSS to your "starbuzz.css" file, save, and reload the Web page. You'll see the award image appear like magic, right where we want it. Make sure you resize the browser to see how the award displays.

A small glitch

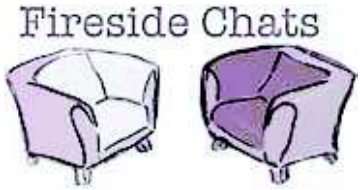
Did you notice a small glitch when you were resizing the browser? Depending on your browser, you may have seen the sidebar ** ** overlap the awards image. What on earth is going on? Remember how each absolutely positioned element has a z-index that describes the stacking behavior of the elements? Some browsers will give the award element a lower z-index than the sidebar ** ** by default. All you need to do to fix that is to specify the z-index for the award, and give it a number higher than the sidebar.

On some browsers the sidebar will overlap and occlude the awards image. Oh dear.


```
#award {
    position: absolute;
    top:      30px;
    left:     365px;
    z-index:  99;
}
```

Let's give the award a really high number to make sure it is always on top.

Go ahead and add the z-index property to your "starbuzz.css" file. When you save and reload, you'll see that now the award ** ** is on top of the sidebar, and the overlap glitch is gone.



Tonight's talk: **Float and Absolute discuss who's more appropriate for layout.**

Float

Absolute, have you noticed lots of people are going with me to do their layouts?

Well, everyone knows I'm better for CSS layout. I'm so much easier to use. Didn't you see? All you have to do is add one little float property to your CSS in the right place, and bam! You've got two columns.

Details, details. My point is that, with me, you don't have to go around counting pixels to figure out where your content's going to go – you can just float it and leave the rest to me.

Well, what about that footer issue? You're always going to overlap things in weird ways, aren't you? If readers aren't careful, they'll have big chunks of their Web pages sitting right on top of other content. At least I respect that handy clear property.

Absolute

Are you sure? I'm used on a lot of pages too, you know.

Hmm, I seem to remember a width property and a margin property to get things just right...

Now, wait a sec. We had to move the *entire* sidebar to a different location in the XHTML to get the float to work the first time. I don't call that "leaving the rest to you." That's a lot of work. At least with me, it doesn't matter what order the content comes in; I'll always do the right thing.

Hey, you sit on top of elements too.

Float

But the important part, the inline content, flows around me, just like it should.

You're missing the point. I'm more flexible and I give people a great way to lay out their pages. I'm sure I can do any layout you can.

Hmmm...

Well, maybe I can't do *everything* you can do, but I think I'm a lot easier for people doing basic layouts, which is mostly what people want.

I used to have that reputation, but most modern browsers are just fine with me now. And, now that Web developers are figuring that out, they're going with me, like I said at the beginning.

I never said you didn't. But check out all the designs out there that use float.

Well, that's not really the point, is it? Anyway, I've got a float to clear, gotta run.

Absolute

Sometimes people *want* to position elements right on top of other elements, you know. And with me, you can position elements anywhere you want. None of this right and left crap like with you. You don't give people that many options, really, if you think about it.

Really? There's no way you could have done that cool thing with the award.

Admit it! You're actually not as flexible as I am.

I dunno, I heard you're kind of buggy in older browsers. That would be frustrating for new Web developers.

I don't think you've seen the end of me; I've got a lot of uses on Web pages.

Hey, don't start thinking you're perfect. You might be good for Web layouts, but you're not exactly the state of art in graphic design.

Clear this, Float.

One more thing you should know about absolute positioning

So far, when you've used the **left**, **right**, **top**, and **bottom** properties to specify the position, these numbers have always been with respect to the edge of the page, right? Well, we need to refine that just a bit.

When you position an element, you're specifying the position relative to the closest ancestor element that is also positioned. So, you're probably saying, "What? I haven't positioned anything except for the sidebar. How could there be an ancestor in my XHTML that is already positioned?" Believe it or not there is – the **<html>** element, which the browser positions for you when it creates the page.

But, let's take this one step further. Say you wanted to absolutely position another ** ** inside the sidebar.

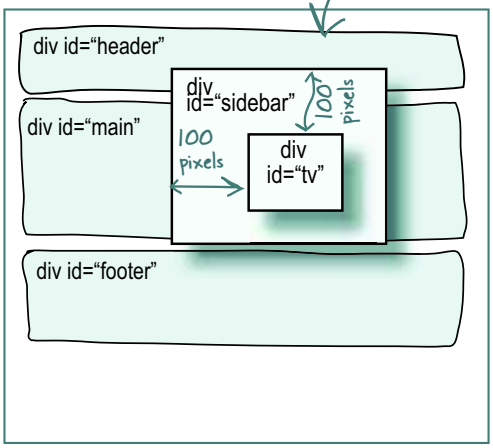
Here's a new nested inside the sidebar.

```
<div id="sidebar">

    <div id="tv">
        <img src="AsSeenOnTv.gif" alt="As Seen on TV" />
    </div>

    <p class="beanheading">
        ... more XHTML here ...
    </p>
    ...
</div>
```

The "tv" is positioned relative to the sidebar , not to the page.



If we absolutely position the "tv" ** **, its closest positioned ancestor is the sidebar ** **. And so, the positioning will be relative to the edges of the sidebar, not the page.

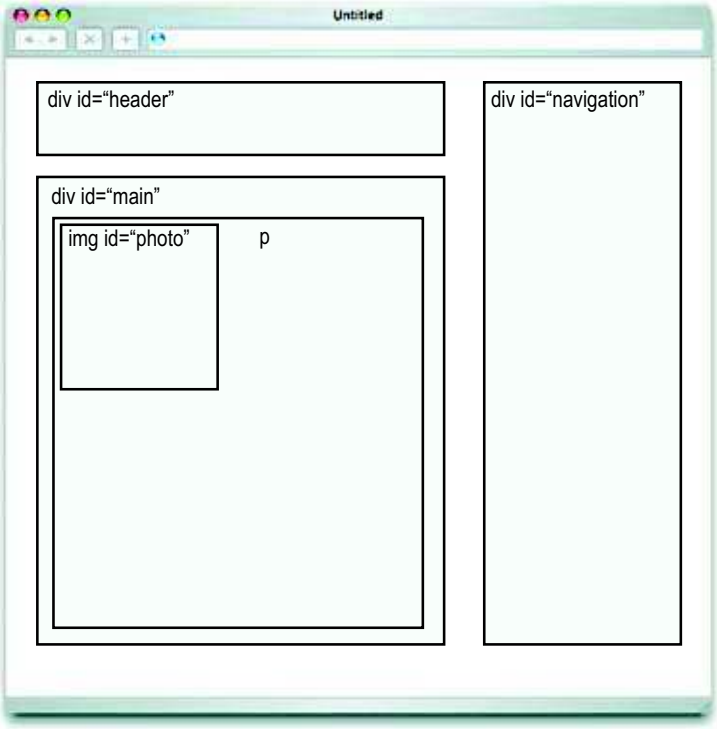
```
#tv {
    position: absolute;
    top:   100px;
    left:  100px;
    width: 100px;
}
```

Another thing to know... if you get caught in a conversation about "closest ancestors that are positioned" at your next cocktail party, just say the "nearest containing block that is positioned." That's the terminology the experts use.

If you're positioning with respect to the <html> element, then the bottom property may not do what you'd expect. You'd think the "bottom" would be the very bottom of the Web page itself, but the <html> element actually defines this as the bottom of the browser window. So, if you want to absolutely position an element from the bottom of the page, rather than the browser window, you need to place your element inside an element that extends to the bottom of your page, and is positioned. One way to do this is to put your element into a relatively positioned element at the bottom of the page. We'll look at relative positioning later in the chapter.

Sharpen your pencil

Time to put all this knowledge about floating and positioning to a test! Take a look at the Web page below. There are four elements with an id. Your job is to correctly match each of these elements with the CSS rules on the right, and fill in the correct id selector for each one. Check your answers at the end of the chapter.



Fill in the selectors to complete the CSS.

```
_____________ {
   margin-top:  140px;
   margin-left: 20px;
   width:       500px;
}

_____________ {
   position: absolute;
   top:      20px;
   left:     550px;
   width:    200px;
}

_____________ {
   float: left;
}

_____________ {
   position: absolute;
   top:      20px;
   left:     20px;
   width:    500px;
   height:   100px;
}
```

Just the words we've been waiting for: "right in the customer's face"

Why? Because it's going to give us the opportunity to try a little *fixed* positioning. What we're going to do is put a coupon on the page that always stays on the screen, even if you scroll. Is this a great technique to make your users happy? Probably not, but work with us here... it is going to be a fun way to play with fixed positioning.

How does fixed positioning work?

Compared to absolute positioning, fixed positioning is pretty straightforward. With fixed positioning, you specify the position of an element just like you do with absolute positioning, but the position is an offset from the edge of the browser's *window* rather than the *page*. The interesting effect this has is that once you've placed content with fixed positioning, it stays right where you put it, and doesn't move, even if you scroll the page.

Impress friends and coworkers by referring to the browser window as the viewport. Try it, it works, and the W3C will nod approvingly.

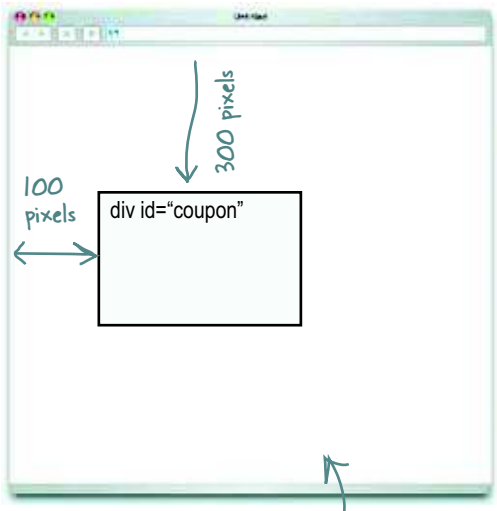
So, say you have a ** ** with an **id** of "coupon". You can position the ** ** fixed to a spot 300 pixels from the top of the viewport, and 100 pixels from the left side, like this:

Here's the id selector for the coupon .

We're using fixed positioning.

```
#coupon {
    position: fixed;
    top:      300px;
    left:     100px;
}
```

Position the coupon 300 pixels from the top, and 100 pixels from the left. You can also use right and bottom, just like with absolute positioning.



Here's where the element gets positioned within the viewport.

Once you've got an element positioned, then comes the fun: scroll around... it doesn't move. Resize the window... it doesn't move. Pick up your monitor and shake it... it doesn't move. Okay, just kidding on the last one. But, the point is, fixed position elements don't move; they are there for good as long as the page is displayed.

Now, we're sure you're already thinking of fun things to do with fixed positioning, but you've got a job to do. So let's get that coupon on the Starbuzz page.

Unfortunately, Internet Explorer version 6 (and earlier) doesn't support fixed positioning. So if you're using Internet Explorer you won't be able to see the coupon properly placed on the Starbuzz Coffee Web page.

Putting the coupon on the page

Now we're going to get the Free Coffee Coupon on the page.
Let's start by creating a ** ** for the coupon to go into:

Here's the with an id of "coupon".

Inside we've got an image of the coupon, which you'll find in the "chapter12/starbuzz/images" folder.

```
<div id="coupon">
    <a href="freecoffee.html" title="Click here to get your free coffee">
       <img src="images/ticket.gif" alt="Starbuzz coupon ticket" />
    </a>
</div>
```

And we've wrapped the image in an <a> element so that users can click on the image to be taken to a page with a coupon they can print.

Go ahead and add this ** ** at the bottom of your "index.html" file, just below the footer. Because we're going to position it, the placement in the XHTML will only matter to browsers that don't support positioning, and the coupon isn't important enough to have at the top.

Now let's write the CSS to position the coupon:

```
#coupon {
    position: fixed;
    top: 300px;
    left: 0px;
}

#coupon img {
    border: none;
}

#coupon a:link {
    border: none;
}
#coupon a:visited {
    border: none;
}
```

We're going to set the coupon to fixed positioning, 300 pixels from the top of the viewport, and let's put the left side right up against the edge of the viewport. So we need to specify 0 pixels from the left.

We need to style the image and the links, too; otherwise, we may have borders popping up on the image because it is clickable. So, let's set the border on the image to none, and do the same on both links and visited links.

Remember that we have a rule in the CSS that says to turn off text-decoration, and use a border to underline links, instead. Here, we're overriding that rule for the link in the coupon and saying we don't want any border on the link. Go back and look at the original CSS if you need to remind yourself of the other rules for the links.

Putting the coupon on the page

Add the new coupon rules to your "starbuzz.css" file, save, and then reload the page. You may need to make the browser smaller to be able to see that the coupon stays put even when you scroll. Clicking on the coupon should take you to the "freecoffee.html" page.

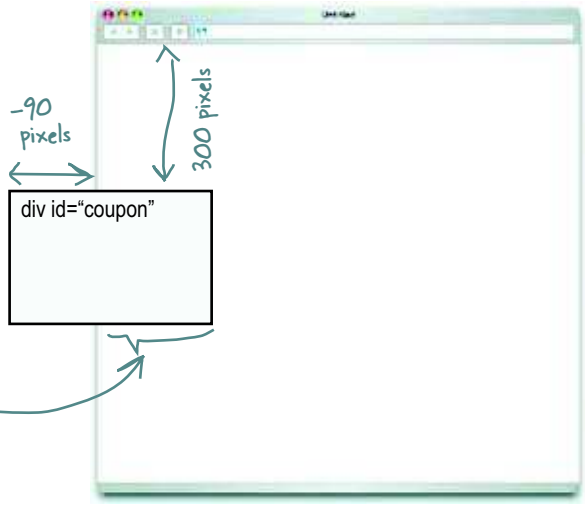
You know, this looks great, but it might just be even more snazzy if the coupon was offset to the left, so it looks like it's coming out of the side of the viewport. Now, we *could* get into our photo editing software and cut off the left hand side of the image to create that effect. Or, we could just use a negative offset so that the left side of the image is positioned to the left of the edge of the viewport. That's right, *you can do that.*

Using a negative left property value

Specify a negative property value just like you do a positive one: just put a minus sign in front. Like this:

```
#coupon {
    position: fixed;
    top: 300px;
    left: -90px;
}
```

By specifying –90 pixels, we're telling the browser to position the image 90 pixels to the left of the edge of the viewport.



The browser will gladly position the image to the left of the viewport for you, and only the part of the image that is still on the screen will be viewable.

A rather positive, negative test drive

Make sure you've put in the negative left property value, save, and reload the page. Doesn't that look slick? Congrats, you've just achieved your first CSS special effect. Watch out George Lucas!

Just remember, using fixed positioning to cover up your content is not the most user-friendly thing to do, but it is FUN.

Can you believe how good this site looks? I mean, look at where it started compared to now. Okay, but we've still got our work cut out for us. We still need to build the Bean Machine, so see you in a couple of chapters.

WOW! What a difference!

Getting relative

This is it, the last type of positioning: *relative positioning*. Truth be told, it's also the loneliest of the positions because you just won't find a lot of people using it in their designs. But, new designs come along every day, so when you see relative positioning, you'll want to know how it works and what it does.

Unlike absolute and fixed positioning, an element that is relatively positioned is still part of the flow of the page, but at the last moment, just before the element is displayed, the browser offsets its position. Let's see how this works on the coffee bag in the Starbuzz Page. We're going to take the coffee bag and offset it to the side, so those coffee beans that spilled out of the bag look like they're spilling out of the page, too.

Now we could absolutely position the coffee bag, but if we did, we'd have to find a way for the space it's taking up on the page to get reserved, since absolute positioning takes the document completely out of the flow.

That's where relative positioning comes in. We can *keep* the element in the flow, have its space reserved, and then offset where it actually gets displayed. Let's try it.

We want to take the Starbuzz Coffee bag and move it about 100 pixels to the right.

With absolute positioning, the coffee bag moves, but since it is no longer in the flow, the rest of the page moves up underneath.

Here's a new rule that selects the image. We're using a descendant selector here to select only images inside the beanheading.

Notice that images are inline elements, but that's okay. You can use any of the positioning techniques, or even a float, on inline elements too.

```
.beanheading img {
    position: relative;
    left: 120px;
}
```

Then we specify a position of relative, and whatever offsets we want on the image. The offset is from the position where it is placed in the flow.

So here we're specifying that the image should be displayed 120 pixels from the left of where it sits in the flow of the document. You can use right, top, and bottom as well, when specifying offsets.

Add this rule to your CSS and then save and reload.

The test drive

After reloading the Starbuzz page, you should see the coffee bag over to the right part of the sidebar. What is interesting is that part of the image is actually extending *beyond* the sidebar into the margin and off the edge of the page. How does that work? Well, as you've seen, the browser first flows a relative element onto the page, and only then does it offset where it is displayed. So the element still takes up the same spot on the page, it's just *drawn* in a different location. Relative is a little like static positioning, but with a dash of absolute thrown in. But, unlike absolute, relative positioning is specified just as an offset from the element's real location, not in absolute coordinates from the nearest containing block.

So, does this improve the page? We're not sure, but it *was fun*. (You might want to remove the relative positioning before you show it to the CEO.)

Not really.

No matter how you tweaked the padding and margins you still can't get an image to be positioned outside of the box it's in. And why try to do it the hard way? We achieved a better effect with two lines of CSS. You can use relative positioning to display an element well beyond the element's box in the flow, which you just can't do with padding or margins.

To three-columns and beyond...

While we've spent this chapter looking at two-column layouts, the real goal was to learn about the **float** and **clear** properties, along with the various forms of positioning that CSS offers. Now that you've got the basics down, you're in a good position to think about three-column layouts, or any other layout you might desire. So, that's it, the chapter's over.

But, wait! Before we finish it off, let's just think through how a three-column layout might work (and if you want to give it a try, just look in the "chapter12/threecolumn" folder).

This design is built using techniques that you already understand. To explore beyond what you've done here, it really does help to see how others have used CSS to create interesting designs, and we encourage you to get out there and look around. Check out some of our favorite online resources for CSS design at:

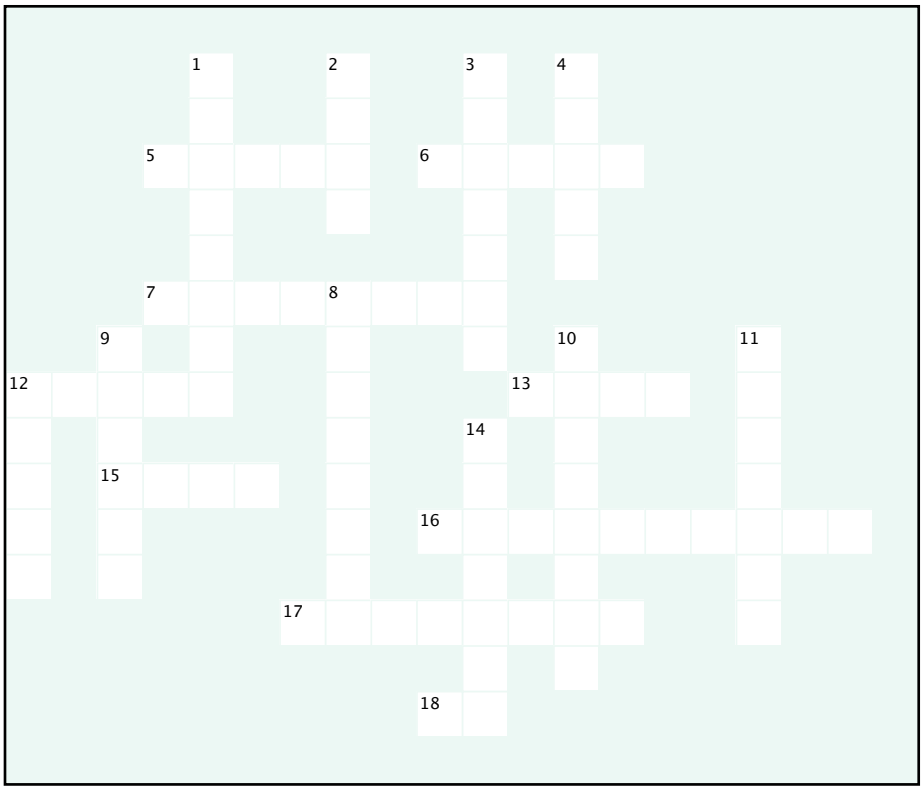
http://headfirstlabs.com/books/hfhtml/chapter12/cssdesign.html

BULLET POINTS

- Browsers place elements in a page using flow.
- Block elements flow from the top down, with a linebreak between elements. By default, each block element takes up the entire width of the browser window.
- Inline elements flow inside a block element from the top left to the bottom right. If more than one line is needed, the browser creates a new line, and expands the containing block element vertically to contain the inline elements.
- The top and bottom adjacent margins of two block elements in the normal page flow collapse to the size of the larger margin, or to the size of one margin if they are the same size.
- Floated elements are taken out of the normal flow and placed to the left or right.
- Floated elements sit on top of block elements and don't affect their flow. However, the inline content respects the boundaries of a floated element and flows around it.
- The clear property is used to specify that no floated elements can be on the left or right (or both) of a block element. A block element with clear set will move down until it is free of the block element on its side.
- A floated element must have a specific width set to a value other than auto.
- A liquid layout is one in which the content of the page expands to fit the page when you expand the browser window.
- A frozen layout is one in which the width of the content is fixed and it doesn't expand or shrink with the browser window. This has the advantage of providing more control over your design, but at the cost of not using the browser width as efficiently.
- A jello layout is one in which the content width is fixed, but the margins expand and shrink with the browser window. A jello layout usually places the content in the center of the page. This has the same advantages as the frozen layout, but is often more attractive.
- There are four values the position property can be set to: static, absolute, fixed, and relative.
- Static positioning is the default, and places an element in the normal flow of the page.
- Absolute positioning lets you place elements anywhere in the page. By default, absolutely positioned elements are placed relative to the sides of the page.
- If an absolutely positioned element is nested within another positioned element, then its position is relative to the containing element that is positioned.
- The properties top, right, bottom, and left are used to position elements for absolute, fixed, and relative positioning.
- Absolutely positioned elements can be layered on top of one another using the z-index property. A larger z-index value indicates it is higher in the stack (closer to you on the screen).
- Fixed position elements are always positioned relative to the browser window and do not move when the page is scrolled. Other content in the page scrolls underneath these elements.
- Relatively positioned elements are first flowed into the page as normal, and then offset by the specified amount, leaving empty the space where they would normally sit.
- When using relative positioning, left, right, top, and bottom refer to the amount of offset from the element's position in the normal flow.
- The same design can often be achieved using floating elements or absolutely positioned elements.
- Float provides a flexible solution for multi-column layouts and allows elements to clear floated elements from their sides, something that can't be done with absolute positioning.

XHTMLcross

This has been a turbo-charged chapter, with lots to learn. Help it all sink in by doing this crossword. All the answers come from the chapter.



Across

5. State between liquid and frozen.
6. Type of positioning that is relative to the viewport.
7. When you place two inline elements next to each other, their margins don't _____.
12. In general _____ is a better technique for column layouts because you can use clear.
13. Inline elements are flowed from the top _____.
15. Special inline elements that get grouped together into boxes as the page is laid out.
16. Absolute positioning is relative to the positioned _____ block.
17. This kind of margin was used on the coupon for a special effect.
18. Usually used to identify an element that is going to be positioned.

Down

1. Another name for the browser window.
2. Method browser uses to position static elements on the page.
3. Property that describes the layering behavior of positioned elements.
4. Property used to fix footer overlap problems.
8. With this positioning, you specify the position relative to the edges of the containing block.
9. Block elements are flowed top to _____.
10. A positioning type that keeps elements in the flow.
11. When boxes are placed on top of each other, these collapse.
12. Removes element from the flow, and sets it to one side.
14. Inline content flows around _____ elements.