CSS LAYOUT WITH FLEXBOX

Laying out a design with CSS is fraught with peril of cross-browser bugs and mysteriously collapsing margins. But new tools like flexbox have transformed this once odious task into something you shouldn't dread. By the end of this chapter you'll be coding up new layouts for your design faster than you can think of them.

Remember those old cartoons with the road runner and the coyote? Rocket propelled rollerskates, bottled lightning, bundles of dynamite, magnetic bird seed, razor boomerangs, "super" bombs and giant trampolines -- no technology could catch the bird. Every tool backfired and sent Wile E. Coyote plummeting to hillarious death.

Using CSS to layout your design is like trying to catch the road runner with a jet-powered pogo stick. It won't end well.

CSS sucks at layout

Nothing is more frustrating than writing a pile of code, refreshing your browser and seeing a jumbled mess of overlapping text and images. You can't use tables because they aren't semantic. Floats and clears never do what you expect. Just to center something on the page requires strange incantations of auto margins or text-align -- the center tag is right out.

But you know what? This is old news because CSS doesn't suck at layout anymore.

Introducing Flexbox

Once you spend some time with flexbox you'll feel like an unstoppable super hero of CSS. Your footer will stay at the bottom of the page where it belongs. Your list items will flow effortlessly into neat little columns. Divs will shrink and grow and fly in delicate formations like the Blue Angels. Image tags will dance and jump and sing songs of your design prowess.

"Wait, let's not get carried away" you say, "Flexbox is too new. You can't even use it in IE9."

It doesn't matter. Think about it: how many browsers does a PSD work in? None? Exactly. As a designer you're not building the final version of the design. You don't care if flexbox doesn't work in Opera Mini because the comp you are creating for your client only needs to run in Chrome. Worry about SeaMonkey later.

The key is to keep your layout code separate from your style code. Create an entirely new stylesheet just for layout. Duplicate selectors if you have to. By keeping it separate, it's easier for a front-end developer to create the final, production-ready layout code once the design has solidified enough.

Diving into Flexbox

You only need to know a handful of CSS properties and a little terminology to use flexbox.

Everything starts with the flex container. Once an element is set as a flex container it's children follow the flexbox rules for layout instead of the standard block, inline and inline-block rules. Within a flex container items line up on the "main axis" and the main axis can either be horizontal or vertical so you can arrange items into columns or rows. The

axes are invisible, but in the examples added, visible lines will help you see how the main axis and cross axis are oriented.

Every example uses the same HTML.

```
<div class="flex">

  <header>
  </header>

  <article>
  </article>

  <aside>
  </aside>

  <footer>
  </footer>

</div>
```

There is a little CSS to style the boxes and give them some dimension, plus the div is set as a flex container.

```
.flex{
    display: flex;
}

.flex > *{
    min-width: 100px;
    min-height: 100px;
}

header{
    background: rgba(200, 200, 200, 1);
}
article{
    background: rgba(200, 200, 200, .8);
}
aside{
   background: rgba(200, 200, 200, .6);
}
footer{
   background: rgba(200, 200, 200, .4);
}
```

A quick overview of flexbox terminology

The first example centers the flex container children, (the <header>, <article>, <aside> and <footer>), vertically and horizontally. Vertical centering with CSS used to be really hard, with flexbox it's one line of CSS.

```
flex-flow: row;
align-items: center; /* This centers the items vertically. */
justify-content: center; /* This centers everything horizontally.…
```

The flex-flow property determines which way the main axis is oriented, (the cross-axis is always perpindicular to the main axis). When it's set to row, the main axis is horizontal, and it's vertical when set to column. You use justify-content to determine where the items line up on the main axis and align-items to shift them around on the cross-axis.

The main axis is horizontal, so the the flex items are lined up in a row.

What happens if the main axis is vertical, (so flex-flow is set to column)?

The main axis is vertical, so the the flex items are lined up in a column.

In the examples you've seen so far, justify-content and align-items have been set to center but you can also move them to the beginning or end of either axis. To move the flex container children to the end of either axis, set justify-content or align-items to flex-end. To move them to the beginning use flex-start.

The content is in the bottom right corner because it's aligned with the end of the main axis and the end of the cross axis.

Space around flex items

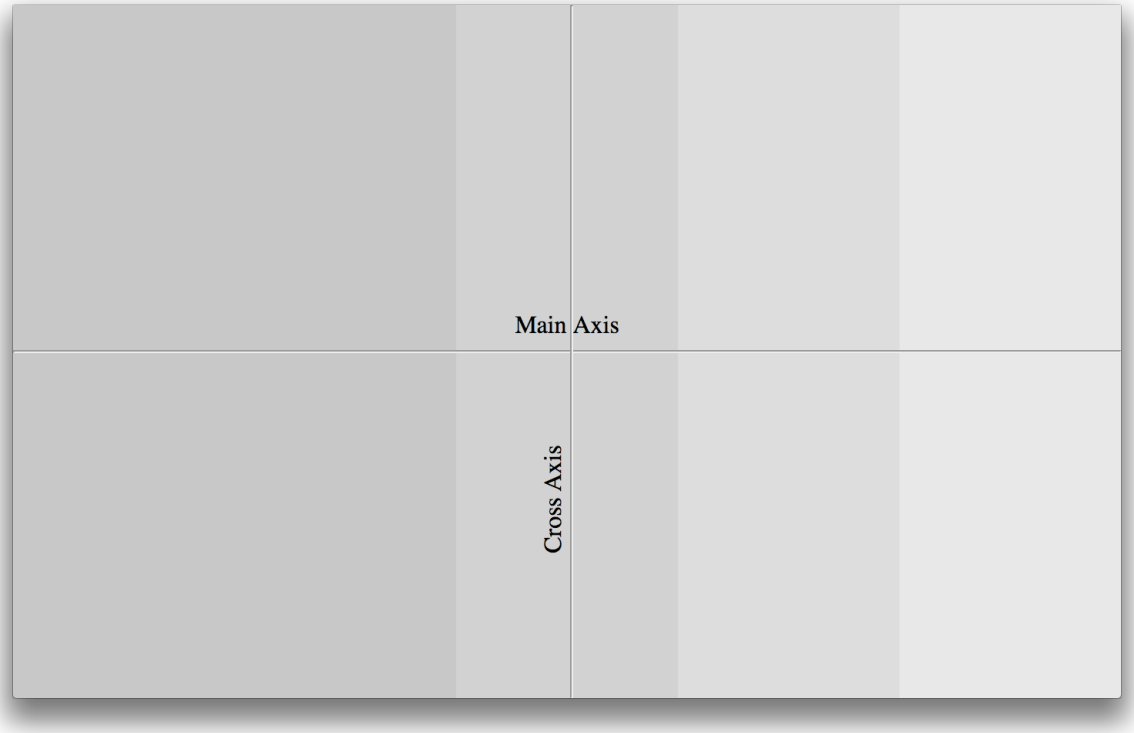
Instead of pushing items to the beginning, center or end of the main axis, you can distribute space evenly around the items. Set justify-content to space-around or space-between.

The flex items are centered on the cross axis and have space evenly distributed around them along the main axis.

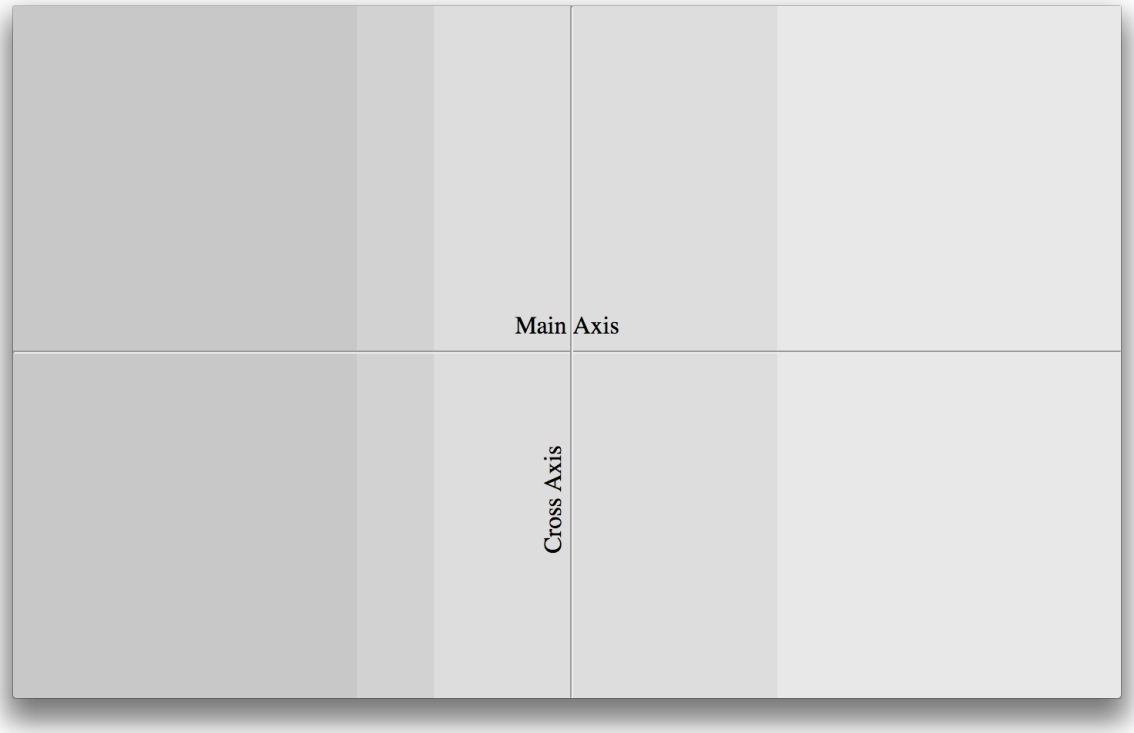
Shrinking and growing

You can define the size of flex items. This lets you create layouts with, for example, a column with a fixed width and a column that grows or shrinks with the space available.

You achieve this magic by setting three separate properties on the children of the flex container: flex-grow, flex-shrink, and flex-basis. Or you can combine all three values in one property: flex.



The left flex item stretches to fill about a third of the screen and the others shrink to fill the rest evenly. In this case I've also decided to have them grow and fill the vertical space.



The skinny column is fixed at 50 pixels. The other columns grow and shrink to fill in the remaining space evenly.

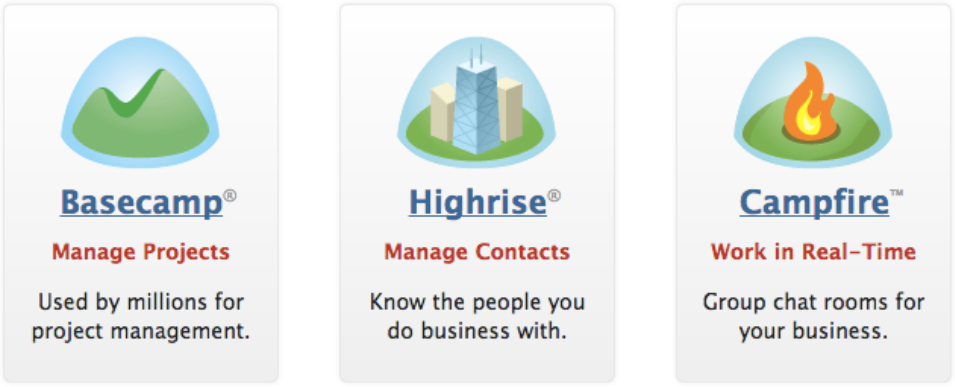
Beyond static images and text

You haven't seen much code up to this point because the CSS property names don't map well to the very simple concepts behind flexbox. Flexbox makes much more sense when you can watch and see what happens on the screen as the code is updated. Here are three screen-casts to help you add flexbox to your arsenal. You can also lean on this cheat sheet while you get used to the terminology.

- The basics
- Growing and shrinking flex items
- More advanced odds and ends

Examples from Real Websites

Flexbox doesn't solve every layout problem but it certainly fixes the worst of them. As you look at a few real world examples you will learn a few other techniques for arranging elements on the page but flexbox alone will get you 90% of the way most of the time.

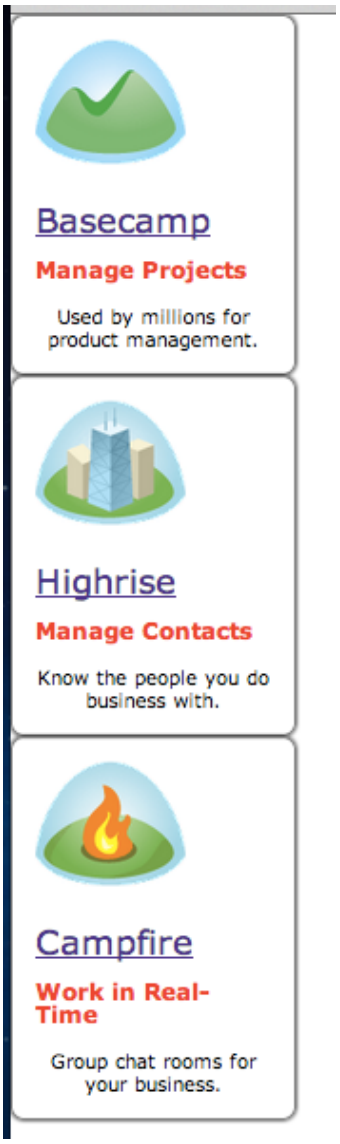


The product section of the 37signals homepage.

You're looking at three divs centered horizontally — certainly not the most complicated layout of all time. Even so, before flexbox creating this kind of layout required some thought; centering elements on the page was not intuitive.

The HTML

```
<div class="products">
  <div>
    <img src="basecamp.png"></img>
    <h2><a href="">Basecamp</a></h2>
    <h3>Manage Projects</h3>
    <p>Used by millions for product management.</p>
  </div>
  <div>
    <img src="highrise.png"></img>
    <h2><a href="">Highrise</a></h2>
    <h3><a href="">Manage Contacts</a></h3>
    <p>Know the people you do business with.</p>
  </div>
  <div>
    <img src="campfire.png"></img>
    <h2><a href="">Campfire</a></h2>
    <h3>Work in Real-Time</h3>
    <p>Group chat rooms for your business.</p>
  </div>
</div>
```



Before any layout logic it's all mushed to the left and in a column.

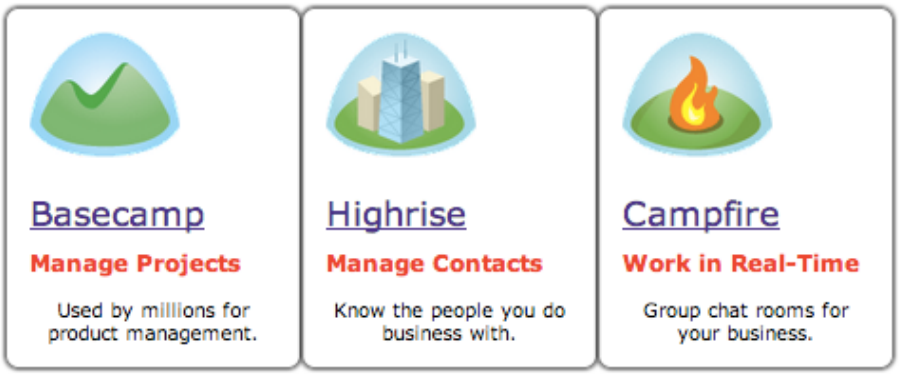
This is how the markup renders with just enough CSS to make the styling look about the same. You are looking at the default browser layout before we've applied any flexbox rules. The three boxes are div elements. Div elements are block elements and block elements stack, they don't flow next to each other.

First let's center it on the page and get everything flowing into a row.

The CSS

```
.products{
display: flex;
flex-flow: row;
justify-content: center;
}
```

Five lines of CSS later...



Five lines of flexbox code setting three properties and our Basecamp example is starting to look pretty good.

The div wrapping the product divs was set to "display: flex" to make it a flex container. Then to get its items into a row, set the flex-flow property to "row".

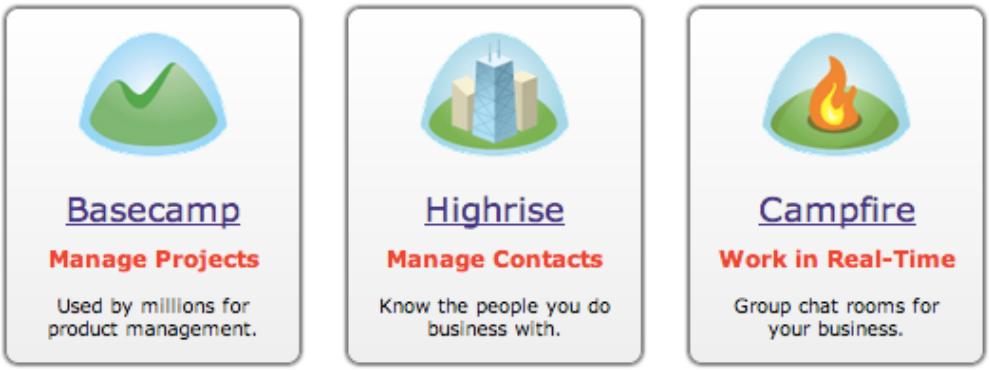
Getting everything centered was easy. Justify-content let's you move flex items on the main axis. In this case the main axis is horizontal because the flex-flow property is set to "row". When the main axis is horizontal, justify-content will move the items to the left and right. If the main axis is vertical, (set flex-flow to column), justify-content moves the items up and down instead of left and right. To center the product divs in the example horizontally,set justify-content to "center".

The first real world example is almost done. The middle container needs some space around it and all of the items in each container should be centered horizontally.

To center the items in the div it's easiest to think about them as flowing in a column and then centering them on the cross axis. So in this case the main axis is vertical and the cross axis is horizontal. Here's the code:

```
/*
1. Make all of the product divs flex containers.
2. Set their items to flow into a column.
3. Center them left-to-right by centering them
on the cross axis using align-items.
*/
.products > div{
display: flex;
flex-flow: column;
align-items: center;
}

/* Using selectors like this
is covered in another chapter.*/
.products > div:nth-child(2){
margin: 0 2em;
}
```



The finished example with everything lined up inside the product divs and margin around the middle.

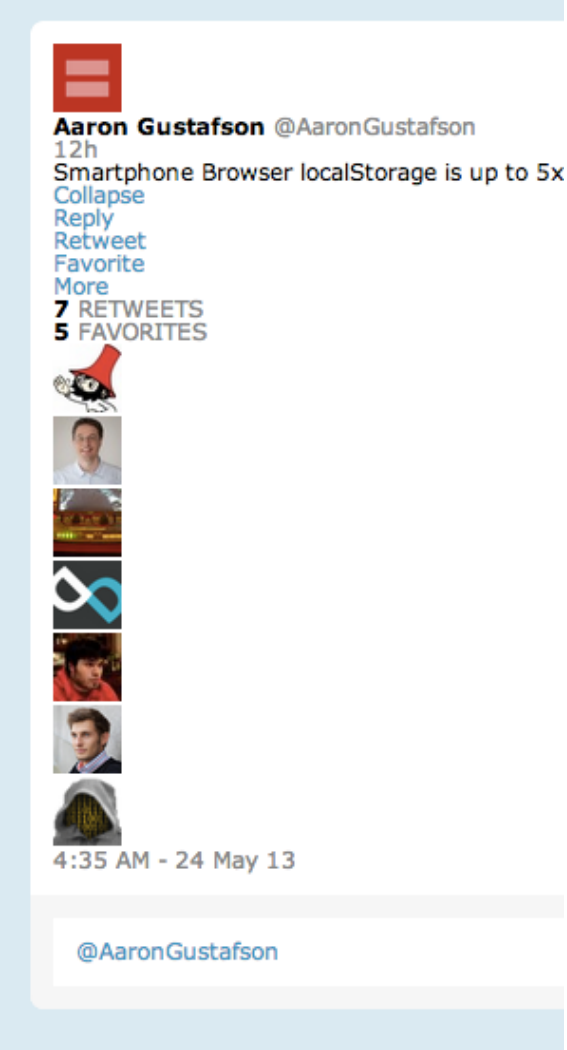
Anatomy of a Tweet

An example tweet.

The HTML

```
<div class="tweet">
  <div class="content">
    <img src="aaron.png"></img>
    <h2>Aaron Gustafson <span>@AaronGustafson</span></h2>
    <time>12h</time>
    <p>
      Smartphone Browser localStorage is up to 5x Faster than N…
      <a href="/">mobify.com/blog/smartphon ...</a>
    </p>
    <ul class="actions">
      <li><a href="">Collapse</a></li>
      <li><a href="">Reply</a></li>
      <li><a href="">Retweet</a></li>
      <li><a href="">Favorite</a></li>
      <li><a href="">More</a></li>
    </ul>
    <ul class="meta">
      <li class="retweets">
        <span>7</span>
        <span>RETWEETS</span>
      </li>
      <li class="favorites">
        <span>5</span>
        <span>FAVORITES</span>
      </li>
      <li class="people">
        <ul class="people">
          <li>
            <img src="people1.jpg"/>
          </li>
          <li>
            <img src="people2.jpg"/>
```

```
            </li>
            <li>
              <img src="people3.jpg"/>
            </li>
            <li>
              <img src="people4.jpg"/>
            </li>
            <li>
              <img src="people5.jpg"/>
            </li>
            <li>
              <img src="people6.jpg"/>
            </li>
            <li>
              <img src="people7.jpg"/>
            </li>
          </ul>
        </li>
      </ul>
      <time>4:35 AM - 24 May 13</time>
    </div>
    <div class="reply">
      <div contenteditable="true">
        Reply to <span>@AaronGustafson</span>
      </div>
    </div>
  </div>
```



The default browser layout: stacked in a column and pushed over to the left.

Perfect is the enemy of done

When confronted with a confounding layout problem a common pitfall is to waste time finding the "best" solution. As soon as you find yourself pondering the merits of changing the markup and worrying about messing up the semantics, stop what you're doing and see if absolute positioning would work.

Whenever you're thinking about technical decisions you're not thinking about design. Remember, your job is to create a comp, not the final

site. The code you are writing will be reworked and rewritten so don't spend any time making it perfect -- perfect is the enemy of done.

The first challenge with the tweet is the user profile image and the date tag. Although not obvious in this example the tweet text flows under the date and the profile image seems to have it's own column. The way the markup was written does not make a flexbox layout immediately obvious. You might be tempted to spend a few minutes to figure out a way you could change the markup and get the layout you want with flexbox, but here is an example where you could easily get mired down in technical details. Instead, just absolutely position both elements where you want them. There is danger in absolute positioning for production code, but it's fine in small doses for prototyping.

The CSS

```
/* create the space on the left for the profile image*/
.tweet .content,
.tweet .reply{
padding-left: 70px;
}

/*move the image to the left into the column*/
.content > img{
position: absolute;
margin-left: -60px;
}

/*
Move the timestamp up and to the right
Used "em" instead of pixels here, but pixels
are just fine.
*/
.content > time:first-of-type{
position: absolute;
margin-top: -1.5em;
margin-left: 30em;
}

/*Added some vertical padding to all
of the elements in the content div.
*/
.content > *{
padding: .25em 0;
}
```

What the tweet looks like after absolutely positioning two elements and adding some spacing.

The next step is to take the actions, (e.g. collapse, reply, retweet, etc.), the retweets and favorites and the list of small profile pictures and put them into rows instead of columns.

A dash of flexbox

```
.actions, li.people, .meta, .people > ul {
display: flex;
}
```

Almost done

One line of CSS and it's almost done.

Elements in containers are arranged into rows by default so you just need to set the container to display: flex.

You'll notice in the real tweet the numbers are on top of "favorites" and "retweets". There are two ways to do this. The first is to treat them like flex containers and put them into a column. The second way is to make each of the spans wrapping the numbers into a block element instead of inline.

The second way is only obvious if you know that block elements stack. The beauty of flexbox is you don't need to know the ins and outs of block, inline or inline-block elements. Learn the basics but don't waste time — use flexbox when it's not obvious.

Stack the elements

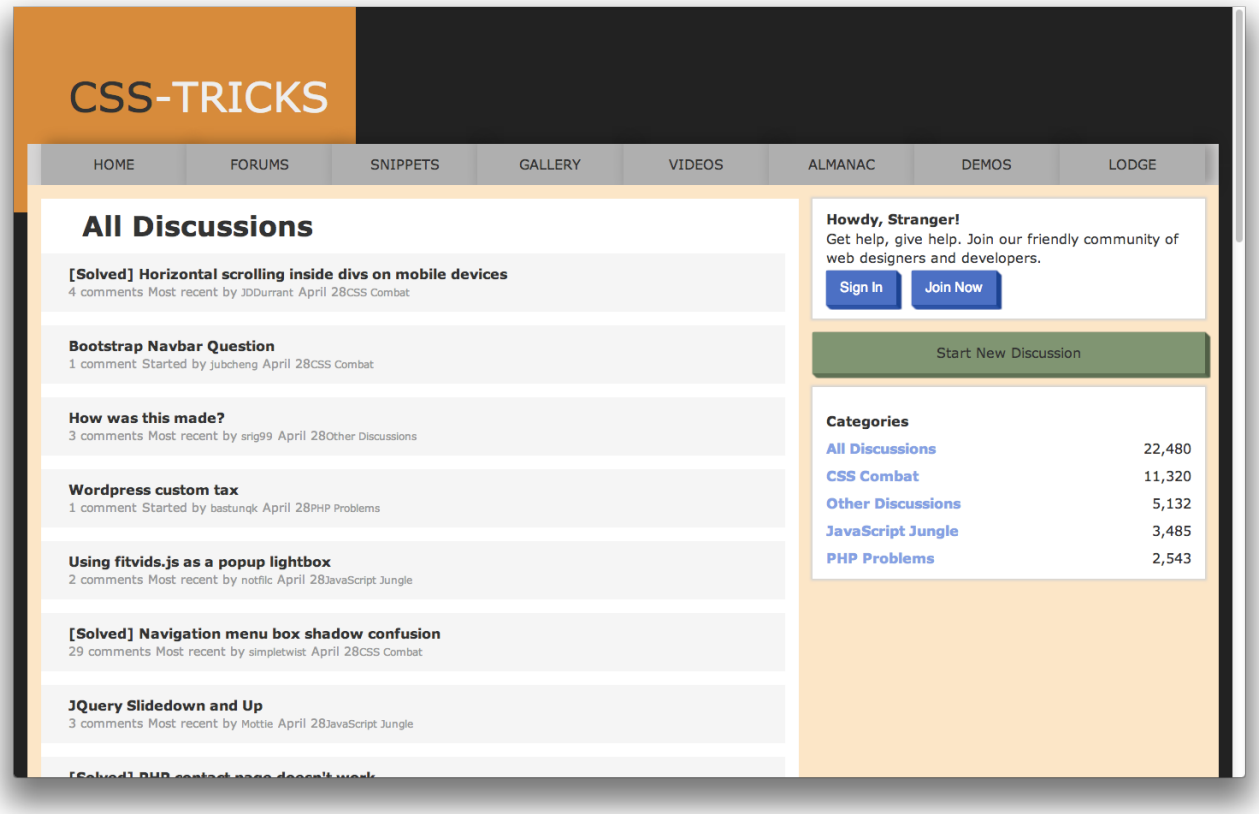
```
.meta .retweets,
.meta .favorites{
display: flex;
flex-flow: column;
}
```

Finished

Two more lines of CSS for layout plus some spacing and borders around key elements. Yes, it's missing icons for the action links. It's just an example.

CSS-Tricks

Included in the book example code is a third, much more complicated example for you to look at. Most of the CSS-Tricks forum has been recreated using flexbox for layout.



The CSS-Tricks forum reverse engineered using flexbox.

Load the example in Chrome and use the skills you learned in the chapter on using Chrome's Developer tools to see how some of the elements in the example are laid out using flexbox.

Tips and tricks

- When absolutely positioning an element use margins instead of top, left, right, bottom. Elements absolutely positioned with margins are more consistent across different screen resolutions.

- Use ems instead of pixels. Ems adjust when you change font-size. If you increased the base font size in this example the date would maintain its position.

- Use padding instead of margin for spacing. Spaces created with margin sometimes collapse into each other instead of stacking. Sometimes you want this but most of the time it's not helpful. Padding doesn't collapse so it's more predictable than using margins.

- Use the universal selector, *, without fear. Experienced web developers avoid the universal selector for performance reasons. You shouldn't spend any time thinking about this. There are performance issues but nothing to worry about when creating the initial design.

- The key to creating more complicated layouts with flexbox is to break it into pieces. Don't be afraid to make everything a flex container if that's what works for you.

Summary

- Mastering the art of laying out elements on a page using CSS is the hardest part of designing in the browser.

- Flexbox has made this task much easier.

- Here is your flexbox cheat sheet, use it wisely.

- Every example in this chapter has code, included with the book, you can look at and use in your projects. Open the examples, (included with this book), with Chrome and use the developer tools to get a closer look.

- The real-world examples are also included with your copy of the book and hosted on the web. The 37signals example, the twitter example and the CSS-Tricks forum.