If you make the browser window narrow, you'll see that this gives us our entire mobile layout. Pretty easy, huh? No media queries required. That's why it's called "mobile-first"—the mobile version doesn't require any special handling. Also notice that `flex-wrap` property in the containing `.page` div. This will make it very easy to implement our tablet and desktop layouts.

By keeping these base styles outside of the media queries, we're able to override and add on to them as we implement our specific layouts. This is really convenient when, for instance, your designer wants to tweak the color scheme for the entire website. Instead of tracking down redundant `background-color` declarations in several `@media` rules, you only have to update it here. That change automatically applies to the mobile, tablet, and desktop layouts.

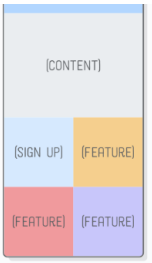
TABLET LAYOUT

On to the tablet layout. The only difference between the mobile and tablet mockups is that the **Sign Up** and **Feature** sections form a 2×2 grid instead of a single column.

Flexbox makes this real easy. Simply adjust the widths of the flex items to be half the screen and `flex-wrap` will take care of the rest. Of course, we only want this behavior to apply to tablet-sized screens, so it needs to go into an `@media` rule. Replace the existing `/* Tablet Styles */` media query with the following:

```
/* Tablet Styles */
@media only screen and (min-width: 401px) and (max-width: 960px) {
  .sign-up,
  .feature-1,
  .feature-2,
  .feature-3 {
    width: 50%;
  }
}
```

To see these changes, make sure your browser window is between 400 pixels and 960 pixels wide, then scroll down to the bottom of the page. You should see a colorful grid:



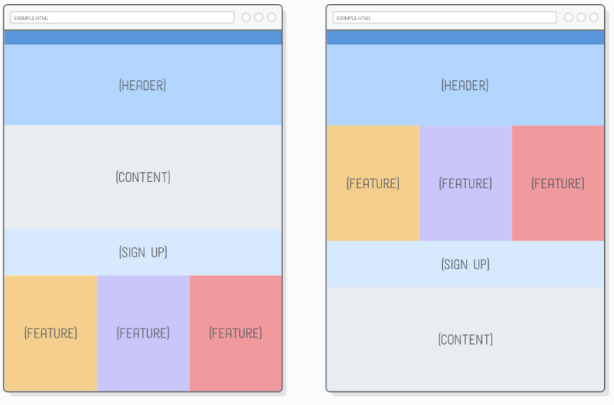
Again, it doesn't matter what the exact width of the screen is: this layout will fluidly respond to any width in the media query's range. Our mobile layout is also fluid, so we now have a website that looks beautiful (if a bit empty) in every device smaller than `960px` wide.

DESKTOP LAYOUT

And that's where our desktop layout comes in. We don't want our web page to expand endlessly, so we're going to give it a fixed width and center it with auto-margins. As with tablet styles, this needs to go into a media query. Replace the existing `/* Desktop Styles */` media query with the following:

```
/* Desktop Styles */
@media only screen and (min-width: 961px) {
  .page {
    width: 960px;
    margin: 0 auto;
  }
  .feature-1,
  .feature-2,
  .feature-3 {
    width: 33.3%;
  }
  .header {
    height: 400px;
  }
}
```

This gives us the correct widths for everything, and we have more real estate to play with, so we made the header a little taller, too. Almost there, but our desktop layout calls for some reordering: the **Sign Up** and **Content** boxes should appear *underneath* all the **Feature** sections.



BEFORE REORDERING **AFTER REORDERING**

This is where flexbox really shines. Trying to create this combination of mobile and desktop layouts would be very difficult with floats. With flexbox's `order` property, it's just a few lines of CSS. Append these rules to the

desktop media query:

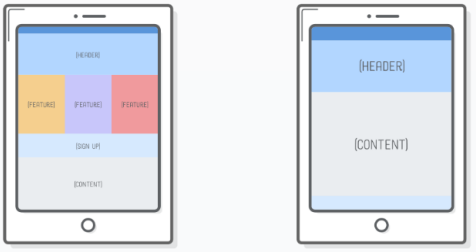
```
.sign-up {
  height: 200px;
  order: 1;
}
.content {
  order: 2;
}
```

Ta da! A responsive website! Not bad for less than a hundred lines of CSS. More importantly, we didn't have to alter a single line of HTML to accommodate our mobile, tablet, and desktop layouts.

This was just one example of laying out a responsive site. You can use these exact same techniques to implement all sorts of other designs. Start with the base styles that apply to your entire site, then tweak them for various device widths by selectively applying CSS rules with `@media`. You could even add another media query to, say, create a dedicated layout for ultra-widescreen monitors.

DISABLING VIEWPORT ZOOMING

We've got one final task for making a responsive web page. Before responsive design was a thing, mobile devices only had a desktop layout to work with. To cope with this, they zoomed out to fit the entire desktop layout into the width of the screen, letting the user interact with it by zooming in when necessary.

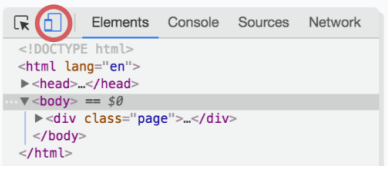


ZOOM ENABLED ZOOM DISABLED

This default behavior will prevent mobile devices from using our mobile layout, which is obviously very terrible. To disable it, add the following element to the `<head>` of our document. Just like `<meta charset='UTF-8'/>`, this is a critical element that should be on every single web page you create:

```
<meta name='viewport'
      content='width=device-width, initial-scale=1.0, maximum-scale=1.0' />
```

To see this in action, we'll need to *simulate* a mobile device in our desktop browser. This is a little advanced for where we're at right now, but we can give it a shot. Open up `responsive.html` in Google Chrome, then hit **View > Developer > Developer Tools** in the menu bar. Then, to simulate a mobile device, click the **Toggle Device Toolbar** icon, highlighted below.



You should see the zoom-disabled version of the above diagram in your browser, since it's now pretending to be a mobile device. (We'll save the in-depth discussion of Chrome dev tools for a future tutorial.)

Alternatively, if you're reading this chapter on a smartphone, you can navigate to the live before and after versions of our example project to experience the effect of our `viewport` changes.

SUMMARY

Believe it or not, that's actually all you need to know to create responsive websites. If we boil it down, we're really only concerned with three things:

- The responsive *design* (the mockups for each layout)
- CSS rules for implementing each of those layouts
- Media queries for conditionally applying those CSS rules

We started this chapter by learning about the difference between fluid layouts and fixed-width layouts. Then, we went on to create a mobile-first stylesheet that used media queries to build tablet and desktop layouts on top of a shared set of base styles. Finally, we disabled the default viewport zoom behavior of mobile browsers.

So, that was the easy part of responsive design. In the next chapter, we'll discover the hard part: images. Presenting different CSS to specific devices isn't too bad, but optimizing images for those devices requires a bit more planning.

NEXT CHAPTER >