CHAPTER

16

Using a CSS Grid System

So far you've learned a number of layout techniques, like how to create float-based layouts to create multiple side-by-side columns and how to create responsive designs that change their layout based on the width of the device displaying the web page. But when your site will appear on screens of all different shapes and sizes, you need all the control you can get.

This chapter expands on the concepts in the previous few chapters and shows you the most precise, yet responsive, page design technique of all—the CSS layout grid. A layout grid is a systematic arrangement of rows and columns used as an organizational tool for laying out content. In the world of CSS, a grid system is a predefined style sheet with styles that make it easy for web designers to add a layout grid to their pages. Grid systems use class names added to `<div>` tags in a page's HTML to create consistently sized columns.

How Grids Work

Grids have a long history in graphic design and traditional page layout. There are various theories about how to best organize content using grids to create beautiful designs, but most of them rely on the unchanging nature of print design and don't translate well to the flexible nature of browsers.

In web design, a grid acts as a way to organize content into rows and columns using consistent column widths. A page's width is divided into a set number of "units" that group easily to create columns of different widths. A common number of units is 12, since 12 divides easily into halves, thirds, and quarters. Because grids use a set

number of units, columns tend to align better and look better because they share a consistent rhythm.

For example, the web page pictured in the top image in Figure 16-1 is organized in rows and columns using a CSS grid system. The page looks well balanced and well aligned because its grid creates consistent column widths. In this example, the underlying grid is composed of 12 units, and the page is divided into five rows. The first row has two columns: The first column, containing the company name, is four units wide; the second, with the navigation bar, is eight units wide. In most grid systems, there's always a gutter, or empty space, separating columns. In this case, there's a single gutter between the two columns.

The second and third rows each hold a single column 12 units wide; the fourth row has three columns, four units wide each, with a gutter between each column; and the bottom row has two columns, one four units wide and the other eight units wide (the same as the first row).

As you may have guessed by now, a "unit" isn't any precise measurement. It's a relative measure, like a percentage value. In fact, in most CSS grid systems, the width of those units is defined using percentage values. So one unit is around 1/12 of the total width of the page—the exact value varies based on the number of columns and gutters in the row. As you read on page 461, relative widths are the key to creating responsive designs, so the onscreen width of a 3-unit-wide column will vary based on the width of the browser window. If you resize the browser window, the precise width of a column in pixels will change, but its relative width will always be the same—3 units.

And therein lies the beauty of a grid system. Remember all that math you needed to calculate the percentage value for a div's width using responsive design (page 470)? That math is already done for you and baked into the CSS file for the grid system. In addition, as you can see in Figure 16-1, using consistent unit values makes page elements line up nicely. For example, the second column in row 4 starts at the same left position as the second column in the bottom row, so they're perfectly aligned. What's more, column widths are consistent for different unit values: The three columns in row 4, for example, are all the same width, because of the consistent and accurate calculations in the grid system.

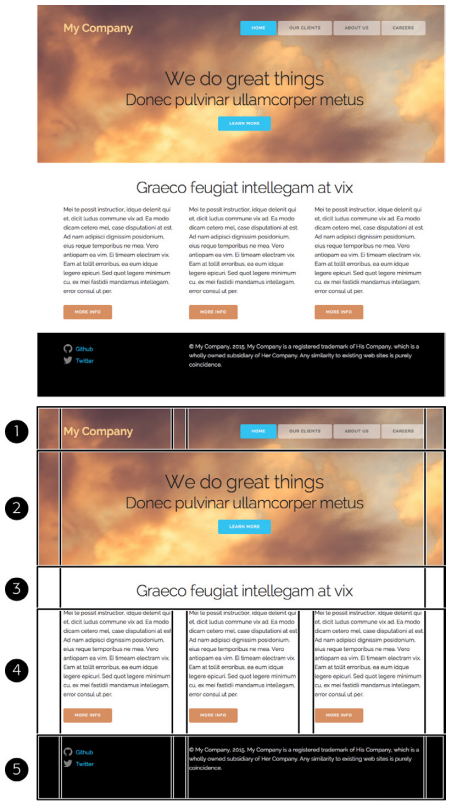

FIGURE 16-1

Many websites use a grid system to lay out their pages using rows and columns with predefined widths. While visitors just see well-designed and visually pleasing pages (top), web designers know that the underlying grid provides organization, balance, and alignment (bottom). Grids are divided into rows that run horizontally across the page (numbered), and columns that divide the rows into smaller units. In addition, grids usually include gutters or extra space on the sides of a row (left and right spaces in the bottom image) and between columns.

Structuring Your HTML for Grids

Individual web designers—and big companies like Twitter—have created dozens of CSS grid systems to choose from. You can read about a sampling of them in the box on page 497. These systems rely on one or more CSS files designed to create the kind of columns described in the previous section. Most of these systems use a similar approach for structuring the HTML to create a grid, relying on a series of nested `<div>` tags to form three different types of page elements:

- **Containers.** A container div holds one or more rows. The container helps set a width for the entire grid system, and often uses the `max-width` property (page 208) to keep the content from growing absurdly wide on large monitors. Containers are also usually centered in the middle of the browser window.

- **Rows.** A row is another tag, placed within a container, and the row holds more divs that make up the columns within it.
- **Columns.** A column is a tag within a row. Each row has one or more columns.

To define a container, a row, or a column, you simply add a class name to each div. The name you use will vary from one CSS grid system to another, but many use some variation of the words *container*, *row*, and *column*. For example, say you wanted to start by creating a page with two rows. The first row contains two columns: one column for a logo and another for the site navigation. The second row holds one column to display the text of an article. The HTML markup for this grid system might look like this:

```
<div class="container">
  <!-- row #1 -->
  <div class="row">
    <!-- 3 unit wide column -->
    <div class="three columns">
      <!-- logo here -->
    </div>
    <!-- 3 unit wide column -->
    <div class="nine columns">
      <!-- navigation here -->
    </div>
  </div>
  <!-- row #2 -->
  <div class="row">
    <!-- 12 unit wide column -->
    <div class="one columns">
       <!-- article text here -->
    </div>
  </div>
</div>
```

There's nothing terribly complicated going on here. In fact, this HTML is very similar to the HTML used in the layout techniques described in Chapter 13. The main difference is that someone else wrote the CSS styles used for laying out the page, saving you a lot of time. All you have to do is download the CSS files for the grid system, attach the style sheet to your web pages, and structure your pages' HTML according to the rules defined by that grid system. This chapter uses a particular grid system (called Skeleton), but you have many CSS grid systems to choose from (see the box below).

POWER USERS CLINIC

Common CSS Grid Systems

This chapter uses a simple but powerful CSS grid system called Skeleton (*http://getskeleton.com*). But it's certainly not your only choice.

- Simple Grid (*http://thisisdallas.github.io/Simple-Grid/*) is another lightweight grid system. It's *barebones*: It doesn't provide any CSS except for creating the grid. It's also responsive, meaning that the page layout adjusts automatically to changing browser widths. So on a mobile device, for example, any multicolumn design instantly changes into a single column design that's easy to read on a phone.
- Pure.css (*http://purecss.io*) is more of a CSS *framework*, meaning it supplies CSS styles that make it easy not only to create a grid, but also to format buttons, tables, menus, and forms. Pure.css is a project of Yahoo, so you know it has some smart people working on it.
- Foundation (*http://foundation.zurb.com*) is another responsive CSS framework. It's very popular, but it's also a lot more complex than most other grid systems. It's great for building web applications and interactive websites because it includes not only CSS, but also JavaScript code for creating drop-down menus, accordions, tooltips, and modal dialog boxes.
- Bootstrap (*http://getbootstrap.com*) is the most popular CSS framework (which explains why a lot of sites look very similar). Created by Twitter, Bootstrap is used in thousands of websites. It includes a CSS grid system, but like Foundation, it also includes everything but the kitchen sink. Bootstrap's CSS includes rules for styling buttons, tables, alerts, badges, labels, and even a large display container called the *jumbotron* (*http://getbootstrap.com/components/#jumbotron*). In addition, there are lot of JavaScript components to choose from, such as tooltips, a carousel, and modal dialog boxes.

Using the Skeleton Grid System

In this chapter, you'll learn how to work with Skeleton, a simple, responsive CSS grid system. You'll also see a few additional CSS rules to help with basic styling of buttons, forms, and tables. Skeleton gets all this work done in 400 lines of code, so it makes for a nice small file. It's built to work really well with mobile devices, and best of all, it's easy to use.

Skeleton is considered a responsive grid system because, while it allows for multiple columns on a page, it collapses those columns to stacked divs when the browser window's width is below 550 pixels. On mobile phones, space is at a premium, and displaying text or any other content in columns makes no sense; it's just too hard to read that way. So most mobile web designers choose to display content in one long column—one div stacked on top of another.

Skeleton lets you build a complex grid layout so your page is neatly structured into multiple columns on tablets, laptops, and desktop displays. On a phone, however, Skeleton's CSS automatically removes the side-by-side columns and forces content into an easy-to-read, single-column design. You don't have to do anything; responsive design is built right into Skeleton's CSS.

USING THE SKELETON GRID SYSTEM

To start, visit the Skeleton website (*http://getskeleton.com*) and click the Download button (Figure 16-2). You'll end up with a folder containing a few other folders and files, but the only ones you really need to worry about are in the CSS folder: *normalize.css*, which provides a basic CSS reset (page 579) so all browsers will style HTML tags the same way; and *skeleton.css*, the file containing all the stuff you need to create a well organized layout.

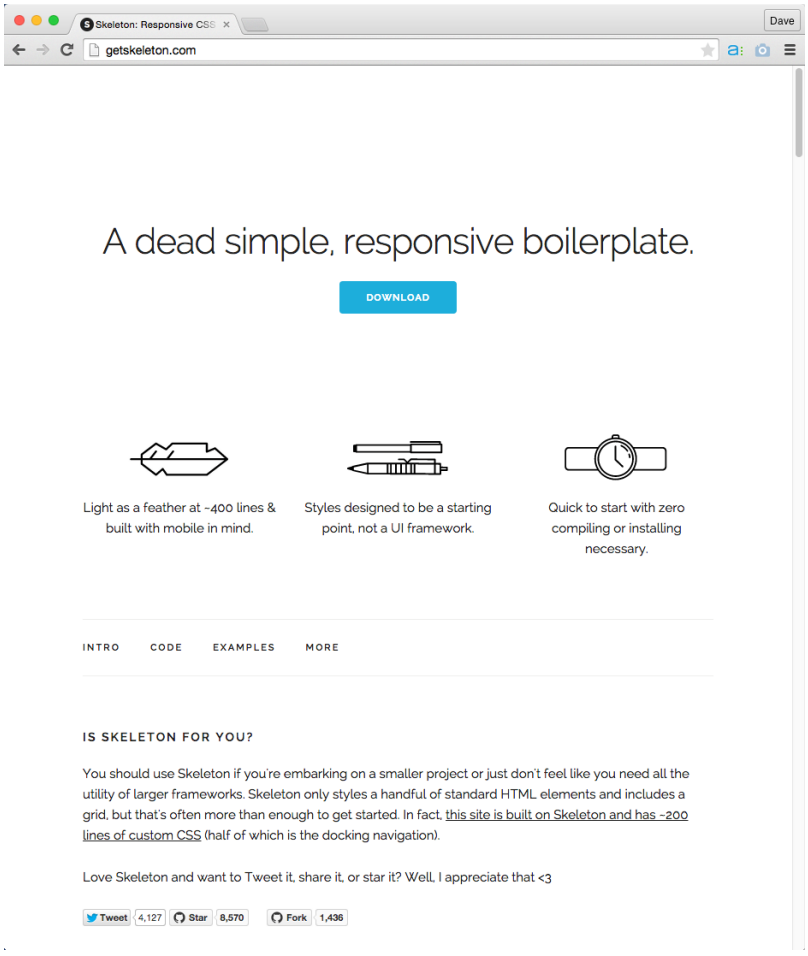

FIGURE 16-2
Skeleton is a set of CSS rules that provide a simple, responsive grid system for page layout.

Here's the basic process for using Skeleton:

1. **Attach the CSS files.**

   You can link to the *normalize.css* and the *skeleton.css* files like so:

```
<link rel="stylesheet" href="css/normalize.css">
<link rel="stylesheet" href="css/skeleton.css">
```

NOTE Skeleton's basic CSS styles use a Google Font (page 140) called Raleway. It's a beautiful font, so if you want to use it on your site, you should also link to those font files by adding another style sheet link to your web pages:

```
<link href="https://fonts.googleapis.com/css?family=Raleway:400,300,600"
rel="stylesheet" type="text/css">
```

2. **Add container divs.**

   As described earlier in this chapter, in a grid system a container div is used to hold rows and columns. A page might have just a single container, but there are times when multiple containers are necessary to create a particular design effect (more on that on page 503). A container in Skeleton is simply a div with the class `container`:

   ```
   <div class="container">

   </div>
   ```

   Don't put any HTML directly inside a container div, except for the row divs described next.

3. **Add divs for rows.**

   Inside each container, you'll have one or more rows. These are simply `<div>` tags nested inside the container div. They have the class `row`:

   ```
   <div class="container">
     <div class="row">

     </div>
   </div>
   ```

   You can put any number of rows inside a single container. In fact, unless you require the particular design effect described on page 503, you might only use a single container for the page containing multiple rows and columns:

   ```
   <div class="container">
     <div class="row">

     </div>
     <div class="row">

     </div>
     <div class="row">

     </div>
   </div>
   ```

4. **Add column divs.**

   Skeleton uses a 12-unit grid, so each div you add to a row must be at least one unit wide (that's quite small) or 12 units wide (the entire width of the container). In addition, if you have more than one row, their total width must come to exactly 12 units. For example, if you have three columns, they each might be four units wide, or one might be two units wide, while the other two are five units wide.

   You define a div as a column by adding the name `columns` to the class attribute. You then set that column's width using a number from one to twelve. Remember that CSS class names can't start with literal numbers like 1, 2, or 12, so you have to spell out these numbered class names. For example, to add three equal-width columns to a row you'd add three divs like this:

```
<div class="container">
  <div class="row">
    <div class="four columns">

    </div>
    <div class="four columns">

    </div>
    <div class="four columns">

    </div>
  </div>
</div>
```

TIP If you want to create a single column within a row, don't add another div inside the row. Just add your content like this:

```
<div class="row">
<p>This text will fill the entire width of the container. It's a single
column.</p>
</div>
```

5. **Add content inside the column divs.**

   The container and row divs are just structural elements used to contain your columns. The actual content—the good stuff like words, photos, and videos—goes inside each of the column divs.

6. **Create your own styles.**

   Skeleton doesn't provide much more than an underlying structure for your content, hence the term "skeleton." To make it look great, you'll need to add your own styles to decorate your site with colors, fonts, background images, and so

on. It's a good idea to create another CSS file named something like *custom.css* for these styles, and link it to your web page after the *skeleton.css* file:

```
<link rel="stylesheet" href="css/normalize.css">
<link rel="stylesheet" href="css/skeleton.css">
<link rel="stylesheet" href="css/custom.css">
```

Skeleton uses a mobile-first approach to its CSS (page 463), so you'll use media queries to refine your design for multiple browser widths, starting with the narrow screen of a phone. (You'll learn how to do that on page 507.)

Creating and Naming Columns

Skeleton uses a 12-unit grid (Figure 16-3). You can create columns of different widths ranging from a single unit to 12. To create a column, you assign two class names to a div. One class is the unit width—`one`, `two`, `three`, and so on—while the other is `columns`. Keep in mind that this isn't a single CSS class name: You must add two class namess and the names must be separated from each other by a space. Each name refers to a different CSS rule in the Skeleton style sheet.

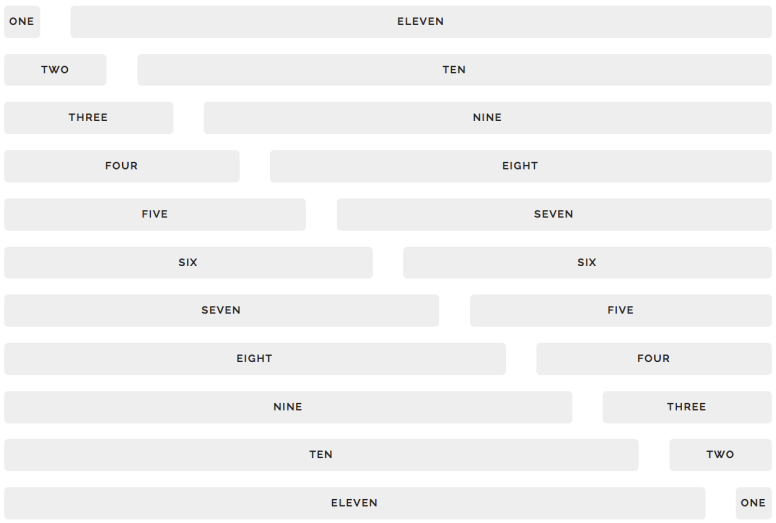

FIGURE 16-3
Skeleton's 12-unit grid system lets you create multiple combinations of column widths. As you can see, a single unit is not very wide at all, so it's unlikely that you'll ever create a column that thin. But since twelve divides into multiple combinations, you can create columns of a variety of different widths.

Take a look at the following two examples. Say you wanted to create a row with two columns of equal width. You'd simply add two divs, each six units wide, inside a row div and container div like this:

```
<div class="container">
  <div class="row">
    <div class="six columns">

    </div>
    <div class="six columns">

    </div>
  </div>
</div>
```

To create a design with four equal-size columns, add four divs, each with a unit size of three:

```
<div class="container">
  <div class="row">
    <div class="three columns">

    </div>
    <div class="three columns">

    </div>
    <div class="three columns">

    </div>
    <div class="three columns">

    </div>
  </div>
</div>
```

Columns don't need to be the same width. However, make sure that whatever width you use, the total number of units in a row always adds up to twelve. For example, it's perfectly fine to have a row with one column that's 4 units wide and another that's 8 units wide:

```
<div class="container">
  <div class="row">
    <div class="four columns">

    </div>
    <div class="eight columns">
```

```
    </div>
  </div>
</div>
```

One really helpful aspect of Skeleton is that its CSS automatically (and correctly) calculates the gutters needed for multiple columns. That is, you don't need to figure out how much space to place between two columns. If you have just a single column in a row, Skeleton won't add any gutter; if you have two columns in a row, it'll add one gutter to separate the two; and if you have six columns, it'll add five gutters—all calculated automatically in the Skeleton CSS file!

TIP Skeleton provides shorthand class names to create columns that are half, one-third, and two-thirds the container width. Just use the names `half`, `one-third`, or `two-thirds` along with the class name `column`. For example, to create two columns, one that's one-third the width of the container, and another that's two-thirds the width of the container, you'd add two divs inside a row, like this:

```
<div class="row">
  <div class="one-third column">

  </div>
  <div class="two-thirds column">

  </div>
</div>
```

Creating Full Width Sections

The container div you add to a page holds rows and columns. Skeleton's CSS doesn't extend the width of the `container` style to fill the browser window for tablet, laptop and desktop screens. The container's width is set to different widths using media queries, but they never extend all the way to the edges of the window. In some cases this is fine, but if you want to add a background color or image to a container your design might not look perfect (see Figure 16-4). In this case, you frequently want to extend the background to the edges of the browser window (as in the bottom image in Figure 16-4).

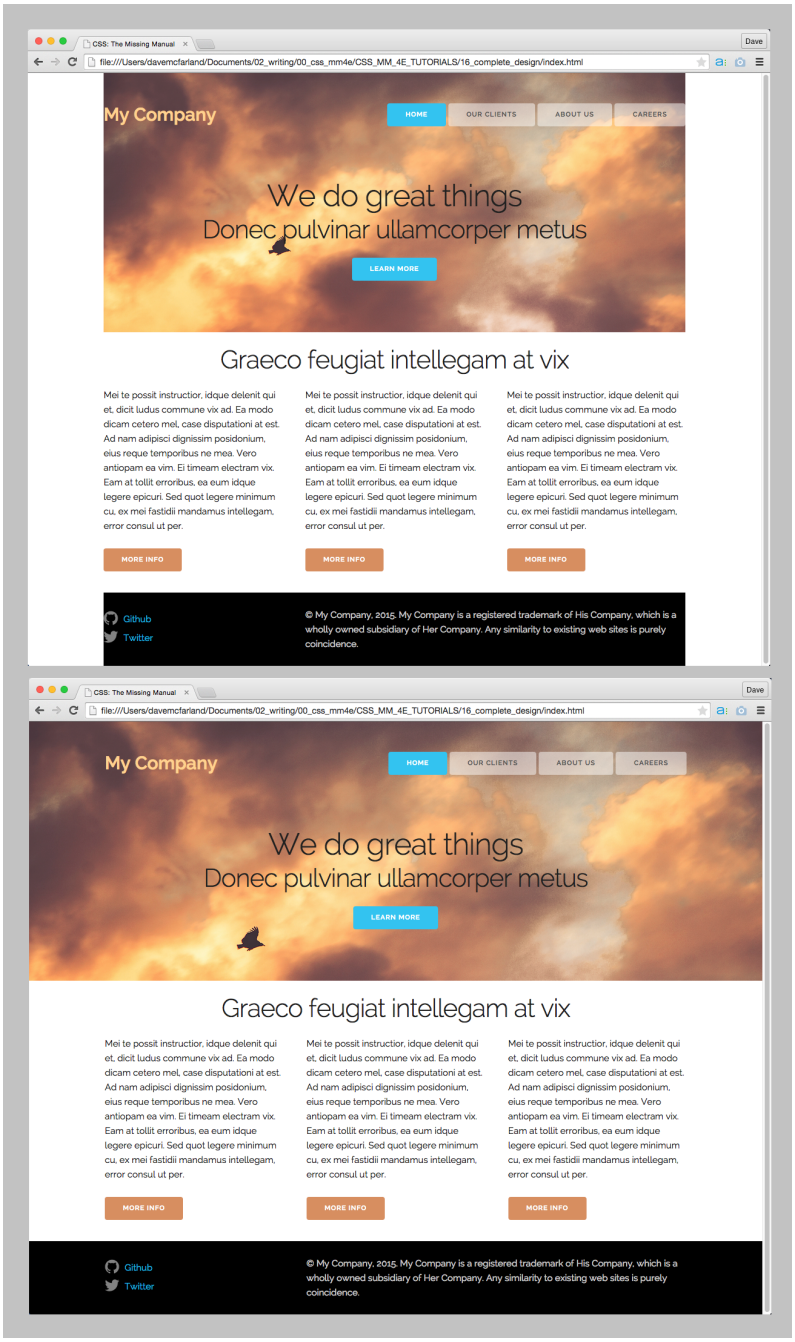

FIGURE 16-4

Skeleton's `container` *class doesn't extend to fill the browser window (top). If you set a background image or color on one of those containers, that background will cover only the container. In many cases, a design looks better if the background color or image extends to the edges of the browser window (bottom). The solution: add a* `<div>` *around each container, and add the background color or image to those wrapper divs.*

Fortunately, this is really easy to do. Since the default width of any block-level element like a tag is 100%, you just wrap the container div with yet another div and set the background color to that outermost div. For example, in the bottom image, the top area of the page has a large photo that fills that header portion from edge to edge. Here's the HTML for that section:

```
<div class="section header">
  <div class="container">
    <div class="row">
      <div class="three columns">
        <p class="logo">My Company</p>
      </div>
      <div class="nav nine columns">
        <a class="button button-primary" href="#">Home</a>
        <a class="button" href="#">Our Clients</a>
        <a class="button" href="#">About Us</a>
        <a class="button" href="#">Careers</a>
      </div>
    </div>
    <div class="row action">
      <h1>We do great things</h1>
      <h2>Donec pulvinar ullamcorper metus</h2>
      <a href="#" class="button button-primary">Learn More</a>
    </div>
  </div>
</div>
```

The critical part is the outermost . This div surrounds the container div (lines 2 and 19). By adding a class name to that div, you can easily set a background image using CSS:

```
.header {
  background-image: url(../imgs/header.jpg);
  background-size: cover;
}
```

Now, since the width of that outer div is always 100%, its background will always extend the full width of the browser window.

As mentioned on page 503, you may only need a single container div for an entire web page. However, you may want to include different backgrounds on different rows, and have *those* backgrounds fill the entire width of the window.

For example, in the bottom image in Figure 16-4, a photo fills the top part of the page, and a black strip fills the bottom, so there are two different backgrounds for two different sections of the page. In this case, you need to include multiple containers: one for the top section (the photo), one for the middle section (the white area), and one for the bottom footer (the black strip). In addition, each of those containers must be wrapped with its own tag. These are the divs that will fill the width of

the browser window, and you style each of those separately. You'll see an example of creating this full-width background effect in the tutorial on page 509.

Styling Buttons

While Skeleton is mainly a grid system, it does provide a couple of fun and attractive styles for formatting other types of page elements. In particular, Skeleton creates cool-looking buttons (see Figure 16-5).

The Skeleton CSS file automatically styles certain HTML elements to look like the buttons pictured in Figure 16-5. For example, it styles the HTML `<button>` element, as well as form input fields with the type `submit` or `button`, like the white buttons pictured in the top of Figure 16-5. Say you have this HTML on a page:

```
<button>Button Element</button>
```

Skeleton's CSS automatically styles it as a white button with rounded corners and a gray border. Now suppose you want to style other HTML elements like this. For example, if you have a series of links forming a navigation bar, you may want each link to look like a Skeleton button. To do so, add the button class name to the link:

```
<a href="index.html" class="button">Home</a>
```

Skeleton includes another class—`.button-primary`—that creates a blue highlighted button (the bottom blue buttons in Figure 16-5). To turn an HTML button element, for example, into one of these blue Skeleton buttons, add a class to the HTML like this:

```
<button class="button-primary">Button Element</button>
```

If you want to turn a plain link into one of these flashy blue buttons, you have to add two classes: one to turn the link to a button and another to make it bright blue, like this:

```
<a href="index.html" class="button button-primary">Home</a>
```

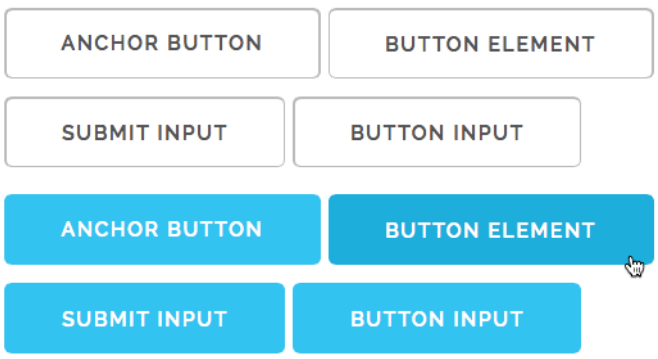

FIGURE 16-5
Skeleton makes it very easy to turn links and form submission fields into simple, attractive buttons.

Mobile First

Skeleton takes a mobile-first approach to CSS. Mobile first, which you read about on page 463, means starting with a design that looks good on small mobile devices like phones. Using media queries, you then add styles that improve the appearance of the page at different screen widths. Each media query defines a new, wider screen width, and you add styles to refine the design at each of these "breakpoints" (widths).

Mobile-first CSS involves using `min-width` settings in the `@media` directive (page 460). In other words, each media query adds new styles that apply when the screen is a minimum width. Take this media query, for example:

```
@media (min-width: 400px) {

}
```

The styles you add here will only work when the browser window is at least 400 pixels wide. If the screen is only 320 pixels wide, the styles inside this media query won't apply to the page. However, the styles will apply when the width of the browser window is 400 pixels or wider. The mobile-first approach adds successively wider media queries; each query applies to not only its `min-width` setting, but also to browsers that match queries with a larger `min-width` setting.

A good approach for creating a mobile-first design with Skeleton is to start with another style sheet. Name it something like *custom.css* or *site.css* and attach it to your web page after the Skeleton CSS files:

```
<link rel="stylesheet" href="css/normalize.css">
<link rel="stylesheet" href="css/skeleton.css">
<link rel="stylesheet" href="css/custom.css">
```

Within this CSS file, you'll add rules to refine the look of the site, adding colors, fonts, backgrounds, and so on. To create a mobile-first design, you need to start with some basic media queries in place. The creator of Skeleton recommends setting media queries to a number of different break points, like the following:

```
* Mobile first queries */
/* applies to ALL widths */

/* Larger than mobile */
/* applies to all widths 400px and greater */
@media (min-width: 400px) {

}
/* Larger than phablet (also point when grid becomes active) */
/* applies to all widths 550px and greater */
@media (min-width: 550px) {

}
```

```
/* Larger than tablet */
/* applies to all widths 750px and greater */
@media (min-width: 750px) {

}
/* Larger than desktop */
/* applies to all widths 1000px and greater */
@media (min-width: 1000px) {

}
/* Larger than Desktop HD */
/* applies to 1200px and greater */
@media (min-width: 1200px) {

}
```

In mobile-first design, you'll want to add styles that apply to all devices (mobile or not) *outside* the media queries. For example, if you want to apply the same font, font color, and background colors to page elements regardless of browser width, then add those CSS rules here.

In addition, you should add the styles that make your site look great on a mobile device here. You can use Chrome's Developer Tools (page 104) to help you see what the page will look like on a mobile device.

Once the mobile version looks good, expand your browser window (or use Chrome's Dev Tools again) to see what your page looks like at the first break point of 400 pixels (line 6 in the code above.) If the page looks perfectly fine at this width (they often do), you can skip this breakpoint and expand your browser window to the next one (550 pixels).

If the design doesn't look great—maybe there's not enough empty space around page elements—then add new styles at this breakpoint until it looks good. Remember, any addition you make within the media query won't apply to earlier breakpoints. So, for example, if you add a style that increases padding around headlines within the media query with the `min-width` of 550 pixels, that padding won't be added to the headlines when the browser window is less than 550 pixels wide. In other words, changes you make inside this media query won't affect how the design looks on a phone.

You can continue this process by checking how your site looks at each breakpoint, adding any styles within the appropriate breakpoint to improve the look at that browser width. Although the code starting on page 507 has five media queries with five different breakpoints, don't feel you have to add styles at each breakpoint. You'll probably find that you only need to adjust the look of page elements for a few different widths. In addition, you may find that the `min-width` values supplied in this code don't work with your particular design. For example, your page design might look great all the way up to 650 pixels, but beyond that, page elements don't seem

to fit or look as good. In this case, just change the `min-width` value for a particular breakpoint and add your styles inside that media query (for instance, change `550px` to `650px`). In fact, you probably won't need all of the media queries listed in the code starting on page 507, so feel free to remove them from the style sheet.

The best way to understand mobile-first design is to see it in action, which you'll get a chance to do in the tutorial coming up next.

Tutorial: Using a Grid System

This tutorial shows you how to apply a grid system to a web page using Skeleton. You'll start with some basic HTML, add the required CSS files, add and structure your HTML to create grids, and finally, add custom CSS rules to make the page look great—including on mobile devices.

To get started, download the tutorial files located on this book's companion website at *https://github.com/sawmac/css_mm_4e*.

Adding a Grid

To get started, you'll attach the required CSS files and add some basic HTML to create a grid container with rows and columns.

1. **In your text editor, open the file *16→index.html*.**

   This is a basic HTML file. It has a head and body, but no HTML in the body yet. Before you add HTML, however, add links to a few CSS files. There's already a link to a Google Font (page 140), but you also need links to the Skeleton files.

2. **On the empty line just before the closing `</head>` tag, add the following three lines.**

   ```
   <link rel="stylesheet" href="css/normalize.css">
   <link rel="stylesheet" href="css/skeleton.css">
   <link rel="stylesheet" href="css/custom.css">
   ```

   The first line of code loads the *normalize.css* file. *normalize.css* is a very popular CSS reset file (page 579) and is used in many projects. The second file is the basic *skeleton.css* file, which includes the CSS for the grid system. The final CSS file, *custom.css*, is mostly empty; it has a few media queries to add styles for different screen widths. You'll use this file to add styles to customize the look of the page.

   Next, you'll add a Skeleton container.

3. **Just below the HTML comment `<!-- top section -->` add the following HTML:**

   ```
   <!-- top section -->
   <div class="container">

   </div>
   ```

As explained on page 495, Skeleton uses the class name `container` to define the div that holds the rows you'll add to a page. You can have any number of containers on a page, or use just a single container div for all of your rows. As you'll see in a minute, there's a benefit of having more than one container.

Time to add your first row.

4. **Inside the `<div>` you just added, insert two divs to create two rows (additions are in bold):**

```
<!-- top section -->
<div class="container">
  <div class="row">

  </div>
  <div class="row">

  </div>
</div>
```

These divs create two rows. To the first row, you'll add two columns—one for the site name and the other for navigation.

5. **Inside the first `<div>` with the class `row`, type the following HTML:**

```
<div class="four columns">
  <p class="logo">My Company</p>
</div>
<div class="eight columns nav">

</div>
```

The Skeleton grid system divides a row into 12 units. You can divide those units to make individual columns, so here you have two columns, one 4 units and the other 8 units. Visually, you have one column that's 1/3, or around 33%, of the container width; the second is 2/3, or around 66%, of the container width.

The second column is wider, because it's going to hold a navigation bar. In fact, notice that there's a class name—`nav`—added to this tag. That's not a requirement of the Skeleton framework; you're adding it now so you can use it later to style the navigation bar.

6. **Open the file *01-nav.html*. Copy the HTML inside it and paste it into the 9-unit-wide div so the HTML looks like this (additions in bold):**

```
<div class="eight columns nav">
  <a href="#">Home</a>
  <a href="#">Our Clients</a>
  <a href="#">About Us</a>
  <a href="#">Careers</a>
</div>
```

These are just simple links and don't look like much. However, Skeleton includes some very fine-looking styles to format any element as a button. You simply add a class name to the <a> tags.

7. **Add a class attribute to each link, and assign a** `button` **class, like this:**

```
<div class="eight columns nav">
  <a class="button" href="#">Home</a>
  <a class="button"href="#">Our Clients</a>
  <a class="button" href="#">About Us</a>
  <a class="button" href="#">Careers</a>
</div>
```

Skeleton also includes a special class named `button-primary` that assigns a special look to a button. It's a great way to make the link for the current page stand out from the others, so visitors can tell what page they're on. The page you're working on is the home page, so you'll add that special class to the Home link.

8. **Add the class name** `button-primary` **to the first** <a> **tag:**

```
<div class="eight columns nav">
  <a class="button button-primary" href="#">Home</a>
  <a class="button" href="#">Our Clients</a>
  <a class="button" href="#">About Us</a>
  <a class="button" href="#">Careers</a>
</div>
```

There's one last row that needs content. This one is simple, because you want just a single row that spans the entire width of the container.

9. **Open the file *02-action.html*. Copy the HTML, return to the *index.html* file, and paste it inside the second row div inside the container div.** The final HTML of the container div will look like this (additions from this step are in bold):

```
<!-- top section -->
<div class="container">
  <div class="row">
    <div class="four columns">
      <p class="logo">My Company</p>
    </div>
    <div class="eight columns nav">
      <a class="button button-primary" href="#">Home</a>
      <a class="button" href="#">Our Clients</a>
      <a class="button" href="#">About Us</a>
      <a class="button" href="#">Careers</a>
    </div>
  </div>
  <div class="row">
    <h1>We do great things</h1>
    <h2>Donec pulvinar ullamcorper metus</h2>
```

```
      <a href="#" class="button button-primary">Learn More</a>
    </div>
  </div>
```

The content in this row will span the entire width of the container—in other words, there's only a single column. In this case you don't add any additional divs; you just put the HTML inside the row div.

10. **Add a class, `action`, to the last row:**

```
<div class="row action">
  <h1>We do great things</h1>
  <h2>Donec pulvinar ullamcorper metus</h2>
  <a href="#" class="button button-primary">Learn More</a>
</div>
```

You'll use that class name later when you style this area of the page. The name, `action`, doesn't have anything to do with Skeleton—it's just a class name you can use can style this section of the page.

11. **Save the *index.html* file and preview it in a web browser.**

The page should look like Figure 16-6. At this point it doesn't look like a marvelously organized grid layout. But it will. Let's add another row and three columns this time. To keep your fingers from going numb, we'll supply you with the basic HTML for the main content on this page.

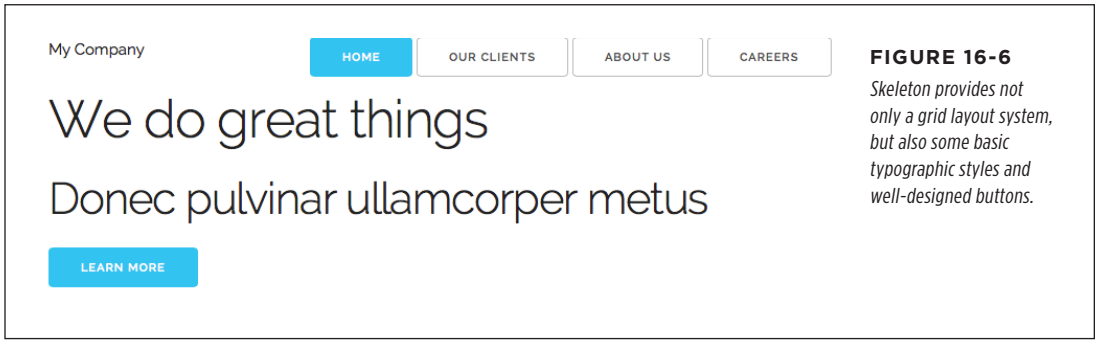

FIGURE 16-6
Skeleton provides not only a grid layout system, but also some basic typographic styles and well-designed buttons.

12. **Open the file *03-info.html*. Copy the HTML and, in the *index.html* file, paste it just below the comment `<!-- info section -->`.**

This is just a basic set of nested divs. There's one `<div>` that represents a container for rows, and two `<div>` tags inside that represent rows. In the first row there's only an `<h2>` tag, which will be a single column that spans the entire container. In the second row are three additional divs, which will be three columns.

The HTML doesn't yet have any of Skeleton's class names—the secret sauce that applies the grid layout—so you'll add those next.

13. **Add** `class="container"` **to the first** `<div>` **in the HTML you just pasted into the page.**

```
<!-- info section -->
<div class="container">
```

This applies Skeleton's container style to the tag, creating a place to insert rows. Next you'll add some rows.

14. **Add** `class="row"` **to the next two divs inside the container, like this (additions in bold):**

```
<div class="container">
  <div class="row">
    <h2>Graeco feugiat intellegam at vix</h2>
  </div>
  <div class="row">
```

Now that you've created the rows, you need to create three columns in that second row. You can do so by adding a class attribute with two class names: The first name is the number of units the column should take up, and the second is the name `columns`.

15. **Add** `class="four columns"` **to each of the** `<div>` **tags inside the second row within the "info" section of this page. While you're at it, add the button class to the three links in each column.** The finished HTML for the "info" section should look like this:

```
<!-- info section -->
<div class="container">
  <div class="row">
    <h2>Graeco feugiat intellegam at vix</h2>
  </div>
  <div class="row">
    <div class="four columns">
      <p>Mei te possit instructior…</p>
      <p><a href="#" class="button">More info</a></p>
    </div>
    <div class="four columns">
      <p>Mei te possit instructior…</p>
      <p><a href="#" class="button">More info</a></p>
    </div>
    <div class="four columns">
      <p>Mei te possit instructior..</p>
      <p><a href="#" class="button">More info</a></p>
    </div>
  </div>
</div>
```

(The HTML shown here has been condensed by not including all of the Latin gibberish in each column.) Finally, you'll add a footer to the page.

16. **Open the file *04-footer.html*. Copy the HTML and, in the *index.html* file, paste it just below the comment <!-- footer -->.**

    This HTML contains the proper Skeleton classes already in place (by now you probably know how Skeleton works, and don't need any more practice typing it all out). The HTML you've just added inserts another container with one additional row and two columns—one 8 units wide and the second 4 units wide.

17. **Save the *index.html* file and preview it in a browser.**

    The page should look like Figure 16-7. The design is coming together: There are two columns in the header (the logo and navigation), three columns of main text, and two columns in the footer.

    If you resize your browser window as thin as it will go, you'll notice that the columns eventually collapse and stack on top of each other. You're watching the responsive part of the Skeleton framework in action. It uses media queries, just like the ones you read about in the previous chapter, to create a single-column design for mobile devices.

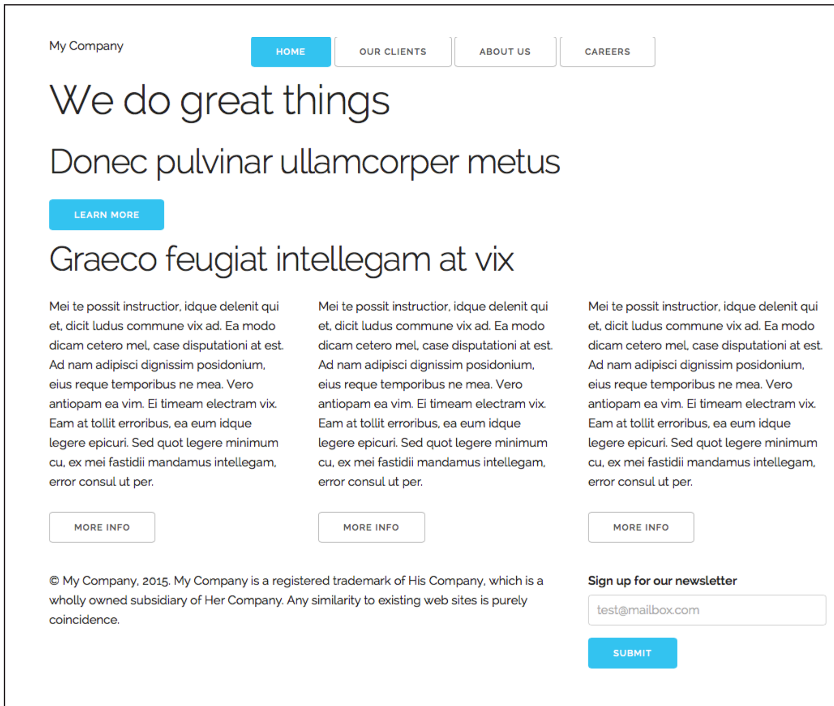

FIGURE 16-7

With just some well-structured HTML, a handful of class names, and the Skeleton CSS file, creating multicolumn layouts is a breeze.

Adding Style

Skeleton takes your designs a long way toward an organized visual presentation, but it's rather plain. To add skin and clothes to this structure, you need to add your own CSS. In this section of the tutorial, you'll add a few design touches.

1. **In your text editor, open the file *16→css→custom.css*.**

   This is a mostly empty CSS file. It only includes media queries set to different breakpoints—the screen widths at which you might want to change the size and look of page elements. For example, on a phone you'll probably want to decrease the size of headlines and eliminate excessive margins and padding.

   You'll start by seeing what happens if you add a background color to a container.

2. **On the empty line following the** `/* Mobile first queries */` **comment, add the following style:**

   ```
   /* Mobile first queries */
   /* applies to ALL widths */
   .container {
     background-color: pink;
   }
   ```

   Remember that your page has several `<div>` tags with the class `container`. These are the tags that contain rows.

3. **Save the *custom.css* file. Preview the *index.html* file in a browser.**

   The page should look like Figure 16-8. Notice that the pink color is constrained to the width of the content, and is centered in the browser window. If you look back at the final design in Figure 16-4, you'll see that you want a photo in the background of the header area, and that photo should extend the entire width of the browser window. Skeleton containers don't do that, so you need to add another div that will span the entire width of the browser window.

4. **Open the *index.html* file in a text editor. Locate the** `<!-- top section -->` **comment and add another div, just below it:**

   ```
   <!-- top section -->
   <div class="section header">
     <div class="container">
   ```

   Here, you're beginning to wrap that container div with another div. You've given it a couple of class names—`section` and `header`—so you can style this area separate from other parts of the page. Now you need to close the div.

5. **Add a closing** `</div>` **just *before* the** `<!-- info section -->` **comment:**

   ```
   </div>
   <!-- info section -->
   ```

Now there's a new div that wraps around the container. By default, like all block-level elements, divs have a width of 100%. That is, they stretch to fill their parent element, which in this case is the page's body. So this new div will fill the browser window. Now you can add a photo that will fill the width of the browser window, but only sit inside the header section.

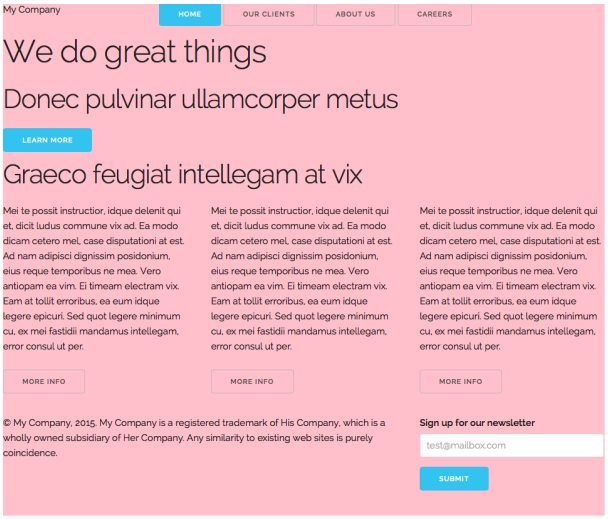

FIGURE 16-8
Skeleton's `container` *class doesn't extend to fill the width of a browser window. This is a problem if you add a background color or image to the container.*

6. **Save the *index.html* file, and return to the *custom.css* file in your text editor. Delete the `.container` rule you added in step 2 above, and replace it with this rule:**

```
/* Mobile first queries */
/* applies to ALL widths */
.header {
  padding: 50px 0 70px 0;
  background-image: url(../imgs/header.jpg);
  background-size: cover;
}
```

The `padding` property just adds a little breathing room at the top and bottom of this region of the page. The `background-size` property (discussed on page 245) scales the image added by the `background-image` property so it always fills the background. If you save the files now and preview the *index.html* file, you'll notice that the picture does span the browser window but only fills the header area.

Now you can format that header area.

7. **Just below the** `.header` **style, add a style for the logo:**

```
.logo {
  font-weight: 600;
  color: rgb(255, 209, 143);
}
```

This page uses a Google font (page 140) named Raleway, which includes three font weights. As you read on page 157, you can use numbers to indicate different font styles from very thin to very heavy and bold. 600 is a bold version of the font.

Next, you'll align the navigation to the right side of the page and improve the way the buttons look against the photo background.

8. **Add three more styles after the** `.logo` **style:**

```
.nav {
  text-align: right;
}
.button {
  background-color: rgba(255,255,255,.5);
}
.button:hover {
  background-color: rgba(255,255,255,.3);
}
```

The `.nav` style formats the column containing the navigation buttons. In step 5 on page 510, you added `nav` to the class attribute for that div. The other two styles change the appearance of the buttons so that they stand out better from the photo background.

Lastly, you'll center the text and button that appears below the navigation area.

9. **Add three more styles below the button styles you just added:**

```
.action {
  text-align: center;
  padding-top: 75px;
}
.action h1 {
  margin: 0;
}
.action h2 {
  margin: 0 0 20px 0;
}
```

You added the `action` class in step 10 on page 512. Here, you're centering the content in that area, and removing some of the spacing around the headlines.

10. **Save the files and preview the *index.html* file in a browser.**

    The page should look like Figure 16-9.

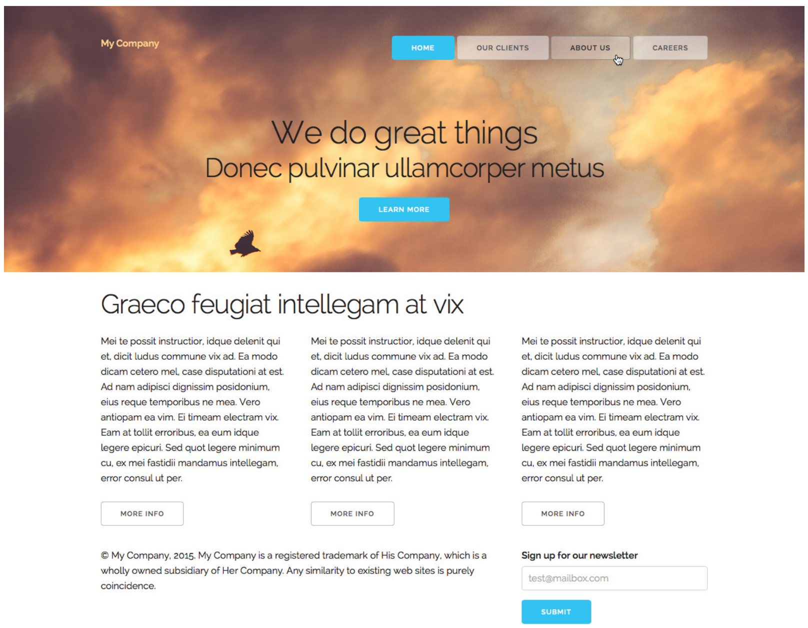

FIGURE 16-9

An image and a few styles of your own add some skin to Skeleton's bones.

The Mobile Design

As you read on page 507, Skeleton uses a mobile-first approach to its CSS. That is, you start with a design for small mobile devices like phones. Then you add styles inside a series of media queries that define progressively wider screen widths. Each set of styles from the previous query applies to the current query and all other queries with greater `min-width` values.

Think about a few enhancements you could make for visitors using mobile phones. Figure 16-10 shows what the current page would look like in an iPhone 5: There's a lot of space at top of the screen. You can fix the logo style to put the company name at the top of the screen, and fix the navigation buttons to fit better.

1. **In your text editor, open the file *16→css→custom.css*.**

   The styles you added to this file earlier in this tutorial are at the top of the file, outside any media queries. In keeping with the mobile first approach, the styles here will apply regardless of the browser width, on both small and super-wide desktop displays.

   First, you'll refine the `.logo` style so that it looks best on a small phone.

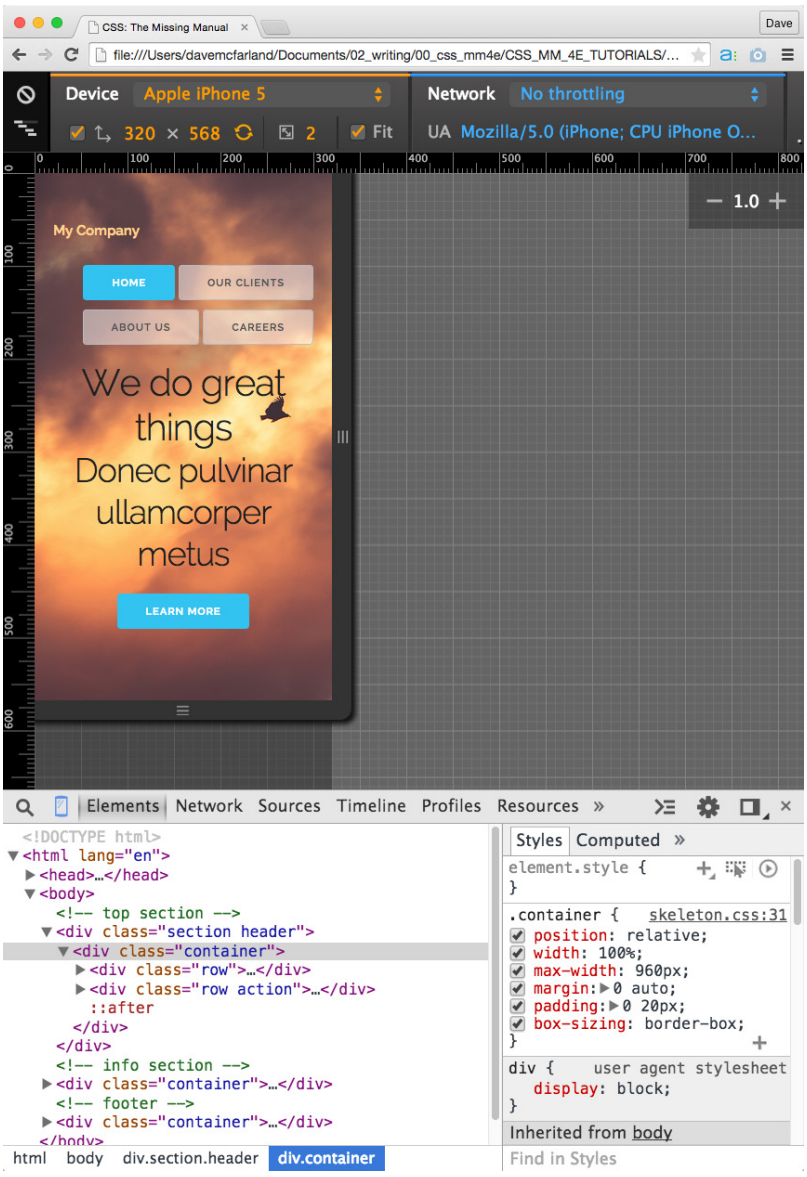

FIGURE 16-10

Using Google Chrome's Developer Tools, you can simulate the screen size of many different devices, including an iPhone 5 (pictured). See page 475 for more information on this cool tool built into Chrome.

2. **Locate the** `.logo` **style and add seven new declarations (additions in bold):**

```
.logo {
  font-weight: 600;
  color: rgb(255, 209, 143);
  background-color: black;
  position: fixed;
  top: 0;
  left: 0;
  right: 0;
  padding-left: 10px;
  z-index: 100;
}
```

The `background-color` setting makes the company name stand out. The `position`, `top`, `left`, and `right` settings fix this element to the top of the browser window. The `padding` setting adds a little breathing room on the left of the screen. Finally, the `z-index` setting (page 439) makes sure that as a visitor scrolls down the page, the logo sits above and on top of other page content (Figure 16-11, left).

The navigation buttons would look better if they were the same width and didn't have so much space around them.

3. **Below the** `.nav` **rule in the *custom.css* file, add the following style:**

```
.nav .button {
  width: 48%;
  margin: 2px 0;
}
```

This style makes the buttons in the navigation bar all the same width. It also adjusts the margins so the buttons don't touch (Figure 16-11, top right).

There's still too much space at the top of the page.

4. **Locate the first style in the *custom.css*—the** `.header` **rule—and change the** `padding` **values to** `30px 0 20px 0`**; to remove the space at top.**

Now the navigation and logo fit better on a small screen, and there's not so much space below the "Learn More" button (Figure 16-11, bottom left). Lastly, it would be good to make that big headline a tad smaller and add more space below the nav buttons.

5. **Locate the** `.action` **rule and change its** `padding-top` **value to** `37px`**.**

This adds some space between the first headline and the nav buttons. Now to shrink the font size of those headlines.

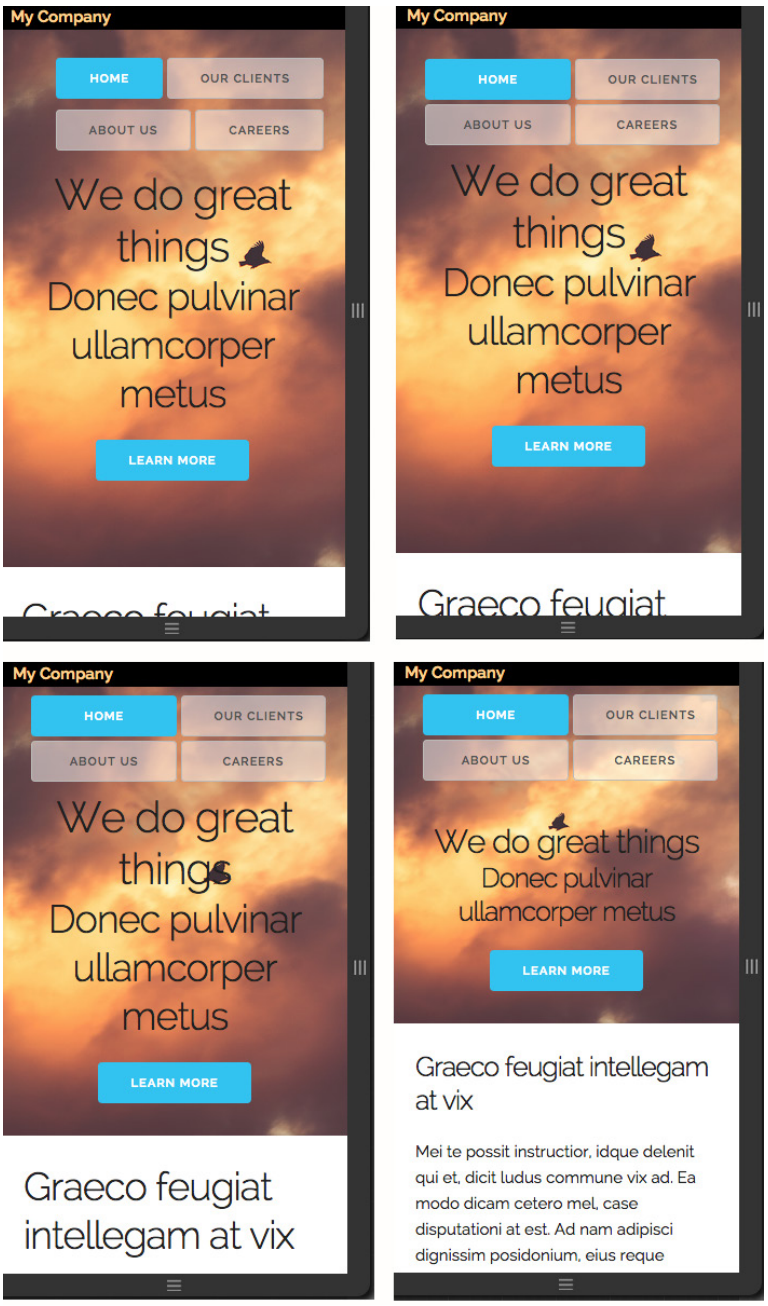

FIGURE 16-11

Creating a mobile-first design requires constant iteration—adjusting styles to adjust spacing, font size, and element placement for different types of devices with different screen sizes.

6. **In the *custom.css* file, after the `.action h2` rule, add the following two styles:**

```
h1 {
  font-size: 3rem;
}
h2 {
  font-size: 2.5rem;
}
```

If you save the file and preview it using an iPhone 5 setting in Chrome's dev tools you'll see something like the bottom right image in Figure 16-11.

Styling a Breakpoint

So far you've made a great start on a mobile-friendly design. It even looks good on screens up to around 550 pixels. In other words, you don't need to worry about adding any styles to the first media query—the one with the `min-width` value of 400 pixels. However, at 550 pixels, the design starts to look a little weird. This is the point at which Skeleton's built-in grid kicks in, and the design changes from stacked divs to columns.

As you can see in Figure 16-12, the navigation buttons take up a lot of space and aren't aligned. It's time to make your design look better on progressively wider screens.

1. **Open the *custom.css* file in your text editor. Locate the media query with a `min-width` of `550px` and add the following styles inside it (additions in bold):**

```
@media (min-width: 550px) {
  .header {
    padding: 40px 0 50px 0;
  }
  .logo {
    position: static;
    background-color: transparent;
    font-size: 2rem;
    line-height: 1;
  }
  .nav .button {
    width: auto;
  }
  h1 {
    font-size: 5rem;
  }
  h2 {
    font-size: 4.2rem;
  }
}
```

FIGURE 16-12
Skeleton's CSS grid springs into action at 550 pixels. This is when a media query kicks in and the styles inside it follow the column settings you specified in the HTML.

These styles reset the spacing in the header section of the page; change the logo so it's no longer fixed on the screen; adjust the width of the buttons; and increase the font size of the headlines (Figure 16-13, top right).

Finally, you'll just pump up the size of the logo text.

2. **In the media query with a `min-width` of `750px`, add one last style:**

```
/* applies to all widths 750px and greater */
@media (min-width: 750px) {
  .logo {
    font-size: 3rem;
    position: relative;
    top: 5px;
  }
}
```

This changes the size of the company name and moves it over slightly.

FIGURE 16-13

Skeleton's responsive grid system makes it easy to create web pages that adjust to a variety of different browser widths, from phones to television screens.

3. **Save the *custom.css* file. Open the *index.html* file in a web browser, and shrink the window to less than 550 pixels wide. Then widen it slowly.**

   The page design when the browser window is less than 550 pixels looks like the top-left image in Figure 16-13. As you widen the browser, the page will change two times: once at 550 pixels (top right) and once again at 750 pixels (bottom).

   There's a lot more you can do to perfect this design at different breakpoints. Play around by adding different styles at the different media query breakpoints and see how you can improve this design.

   You'll find the finished design in the *16_finished* folder inside the tutorials folder.

As you can see, a grid system is a handy tool that makes creating consistent column widths easy. However, underneath the hood, you'll find the same principles you learned earlier in this section of the book: floats, percentage widths, and so on. If you spend some time looking at the *skeleton.css* file, you can learn all about how it works.

TIP For a great discussion on how to build your own grid system with CSS read the article at *https://css-tricks.com/dont-overthink-it-grids/*.