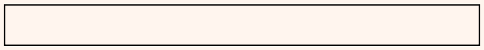
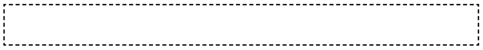
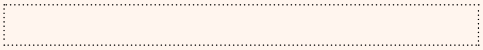
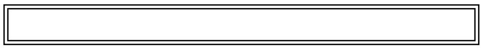
3.20 Border Properties

Now onto the next category of CSS properties—border properties. As expected, the border properties allow you to specify the appearance of borders that surround elements. We won't bother to cover all the border properties, just the more important ones. Specifically, we'll describe the `border-style`, `border-width`, and `border-color` properties. Then we'll finish with the `border` shorthand property.

`border-style` Property

The `border-style` property specifies the type of border that surrounds the matched element. Here are the valid values for the `border-style` property:

`border-style` Values	**Appearance**
`none`	The browser displays no border. This is the default.
`solid`	
`dashed`	
`dotted`	
`double`	

Here's an example class selector rule that uses the `border-style` property to draw a dashed border:

```
.coupon {border-style: dashed;}
```

We used "coupon" for the class selector because we want the matched element to look like a coupon, with a traditional dashed border surrounding it.

`border-width` Property

The `border-width` property specifies the width of the border that surrounds the matched element. There are quite a few values allowed for the `border-width` property. Here are the most appropriate ones:

`border-width` Values	**Description**
`thin, medium, thick`	The browser determines a border width that can be reasonably described as `thin`, `medium`, or `thick`. The default is `medium`.
`number of px units`	A CSS pixel unit is the size of a single projected dot on a computer monitor when the monitor's zooming factor is at its default position of 100%.

If you ever use the border-width property, remember to use it in conjunction with the border-style property. If you forget to provide a border-style property, then the default border-style value kicks in, and the default value is none. With a border-style value of none, no border will be displayed. Forgetting the border-style property is a very common bug.

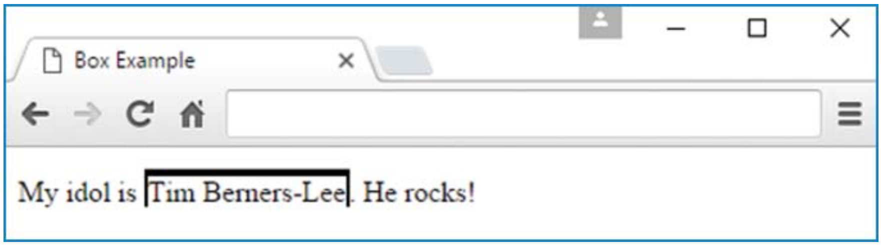
CSS pixel values use px units. As with all the other CSS size values, CSS pixel values are relative. If a user reduces the monitor's resolution or zooms in on his or her browser, then each CSS pixel expands, and elements that use CSS pixel units will likewise expand.

It's pretty rare to need different widths for the different border sides, but be aware that the feature does exist. If you specify four values for the border-width property, the four values get applied to the border's four sides in clockwise order, starting at the top. For example:

```
.boxed {
  border-style: solid;
  border-width: 4px 2px 0px 2px;
}
```

Note what happens when you apply the preceding CSS rule to the following HTML code fragment:

```
My idol is <span class="boxed">Tim Berners-Lee</span>. He rocks!
```



My idol is Tim Berners-Lee. He rocks!

The border's top side is thickest due to the 4px value. The right and left sides are both two pixels, and the bottom side is missing due to the 0px value.

Google's Style Guide says if you have a zero value for a CSS property, you should omit the unit in order to make the code more compact. So in the earlier code, that means using 0 instead of 0px. However, in the interest of parallelism with the other three values, I felt that keeping the px unit was appropriate.

With CSS properties that involve four sides (i.e., the preceding border-width property, and the margin and padding properties coming up), there are shortcut techniques for specifying the values. If you specify three values, then the first value applies to the top side, the second value applies to the left and right sides, and the third value applies to the bottom side. Thus, the preceding border-width property value pair could have been written as follows and the result would be the same:

```
border-width: 4px 2px 0px;
```

If you specify just two values, then the first value applies to the top and bottom sides and the second value applies to the left and right sides.

In addition to its shortcut techniques, the border-width property also has techniques that are more verbose. Specifically, you can use a separate font property (border-top-width, border-right-width, border-bottom-width, and border-left-width) for each side's border width. Google's Style Guide recommends not using those properties—too wordy—and we'll follow that recommendation for the most part.

Whew! There are a lot of alternative syntax techniques for the border-width property. Thankfully, borders should normally have the same width for all four sides, and to make that happen, the syntax is easy. Just specify one value for the border-width property, and that single value applies to all four sides.

border-color Property

The border-color property specifies the color of the border that surrounds the matched element. There's no new syntax to learn for the border-color property because it uses the same values as the color property and the background-color property. Remember the types of values that those properties use? Color values can be in the form of a color name, an RGB value, an RGBA value, an HSL value, or an HSLA value.

For the border-color property to work, you must use it in conjunction with a border-style property. That should sound familiar because we said the same thing about the border-width property. In order to change the border's color or change the border's width, you must have a visible border, and that's done by using a border-style property.

border Shorthand Property

If you want to apply more than one of the prior border-related properties to an element, you could specify each of those properties separately, but there's a more compact technique. The border property is a shorthand notation for specifying a border's width, style, and color in that order. Here are two examples:

```
.understated-box {border: thin dotted blue;}
.in-your-face-box {border: 10px solid;}
```

In the second example, there isn't a color value. You can omit the width and/or color values, but if you omit the style value, then there will be no visible border. You have to include the style value; if you don't, then border-style's default value of none will be used. In both examples, style values are included (dotted and solid), so the borders are visible.

3.21 Element Box, padding Property, margin Property

In the previous section, you learned how to display a border around an element. Usually, borders have no gaps inside or outside of them. Sometimes that's appropriate, but usually you'll want to introduce gaps to make the elements look comfortable, not cramped. To introduce gaps around

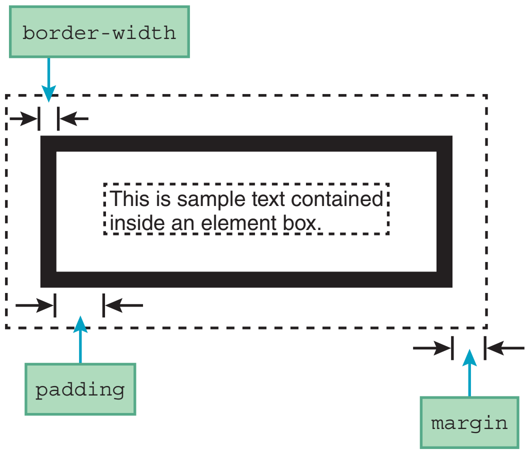


FIGURE 3.19 An element box's margin, border, and padding

an element's border, you need to take advantage of the element's element box. Every web page element has an element box associated with it. As you can see in **FIGURE 3.19**, an element box has a border, padding inside the border, and a margin outside the border. For most elements, but not all, the default border, padding, and margin widths are zero. You can adjust the widths with the `border-width`, `padding`, and `margin` properties.

In Figure 3.19, the dashed lines indicate the perimeters of the margin and padding areas. When a web page is displayed, only the border can be made visible; the dashed lines shown in the figure are only for illustration purposes.

If you have trouble keeping track of which property is inside the border, `padding` or `margin`, think of a package you get in the mail. You put padding on the inside of a fragile package's box, so the `padding` property is for inside the border. Therefore, the `margin` property must be for outside the border.

`padding` and `margin` Properties

The `padding` property specifies the width of the area on the interior of an element's border, whereas the `margin` property specifies the width of the area on the exterior of an element's border. Usually, the most appropriate type of value for the `padding` and `margin` properties is a CSS pixel value. Here's an example CSS rule that uses `padding` and `margin` properties:

```
.label {border: solid; padding: 20px; margin: 20px;}
```

Just as with the `border-width` property, you can specify different padding widths for the four different sides. You can use multiple values with one padding property. Or you can use separate padding side properties—`padding-top`, `padding-right`, `padding-bottom`, and `padding-left`. Likewise, you can specify different margin widths for the four different sides. You can use multiple values with one margin property. Or you can use separate margin side properties—`margin-top`, `margin-right`, `margin-bottom`, and `margin-left`.

The margin and padding properties allow negative values. While a positive value forces two elements to be separated by a specified amount, a negative value causes two elements to overlap by a specified amount.

Example That Uses padding and margin Properties

Let's put this padding and margin stuff into practice in the context of a complete web page. FIGURE 3.20's browser window shows span elements that could serve as labels for water faucet handles. The borders are curved to form ovals. We'll get to the implementation of the curved borders shortly, but let's first focus on the padding and margins.

FIGURE 3.21 shows the code for the Hot and Cold web page. Here's the relevant code for the padding and margins, where label is the class attribute value for both of the span elements:

```
.label {
  padding: 20px;
  margin: 20px;
  display: inline-block;
}
```

The padding and margin properties both use values of 20px, which provides significant space on the interior and exterior of the borders. For the web page's first-cut implementation, there was no display property (as shown), and there was no space provided above the two ovals—the ovals bumped up against the top edge of the browser window, which looked ugly. Figuring out the solution required some serious head-scratching. After much trial and error and googling, I learned that vertical margins do not work for standard phrasing elements, such as the span element. On the other hand, vertical margins do work for block elements. So, the trick was to turn the label's span element into a block element—well, sort of. To help with the explanation, we'll use some different nomenclature. In the world of CSS, we'll refer to phrasing elements as *inline elements* because CSS uses the inline value to describe elements that are to be formatted as phrasing elements (where the element's width matches the width of its contents). We'll also refer to elements that have

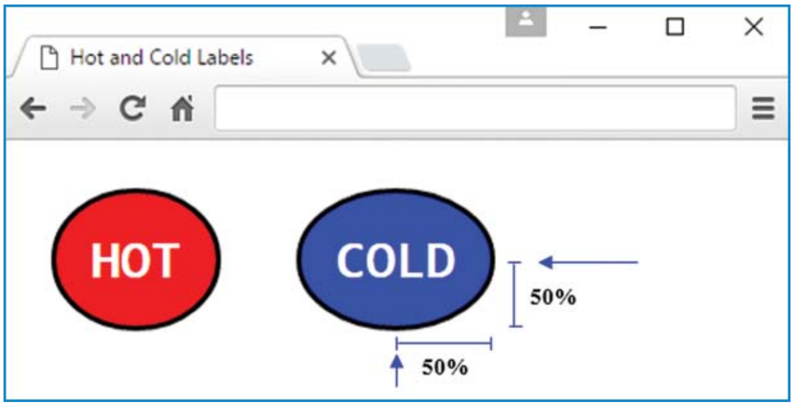


FIGURE 3.20 Hot and Cold web page

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="author" content="John Dean">
<title>Hot and Cold Labels</title>
<style>
  .hot {background-color: red;}
  .cold {background-color: blue;}
  .label {
    color: white;
    font: bold xx-large Lucida, monospace;
    border: solid black;
    border-radius: 50%;
    padding: 20px;
    margin: 20px;
    display: inline-block;
  }
</style>
</head>

<body>
<span class="hot label">HOT</span>
<span class="cold label">COLD</span>
</body>
</html>
```

This implements curved corners.

This is necessary for the vertical margins to work.

FIGURE 3.21 Source code for Hot and Cold web page

characteristics of both phrasing and block elements as *inline block elements*. So here's the deal—vertical margins do work for inline block elements. The `span` element is considered to be a standard inline element by default. By adding the preceding `display: inlineblock` property-value pair to the `label` CSS rule, we make the `span` element more "blockish," and the browser accommodates by adding space above the ovals. This is another example where research and tweaking are necessary.

Now onto the curved borders. The `border-radius` property allows you to specify how much curvature you want at each of the four corners of an element. The default, of course, is to have no curvature. To achieve curved corners, you need to specify the focal point for each of the corners' curves. Often, you'll have different focal points for each of the four corners, but for both oval elements in the Hot and Cold web page, all four focal points are in the same place—at the center of the oval. To understand how the blue oval's lower-right-corner curve is drawn, imagine a compass with its pointy arm stuck at the center of the blue oval and its free arm sweeping from the bottom of the blue oval toward the right edge of the blue oval. As it moves to the right and up, the compass arm will have to swivel open slightly in order to reach the far right edge of the oval. That sweeping motion forms the lower-right corner's curvature.

So, where does the CSS rule come into play? As you can see in Figure 3.21, the `border-radius` property has a value of `50%`. That `50%` value is used to form the spanning lines shown in Figure 3.20. The specified percentage (50% in this case) is relative to the width and height of the element. So for the lower-right corner, one spanning line goes to the left by 50% of the element's width and the other spanning line goes up by 50% of the element's height. As you can see in Figure 3.20, the ends of the two spanning lines determine the starts of the two arrows that intersect within the element. That intersection point serves as the focal point for the lower-right corner's curved border.

Yes, the `border-radius` property is somewhat complex. If you want to learn all its behaviors and its different syntax options, check out the W3C's explanation at https://www.w3.org/TR/css3-background/#the-border-radius.

Using a Percentage for a Web Page's Margin

As indicated, you'll normally want to use CSS pixel values for margin widths. However, it's sometimes appropriate to use percentage values instead of pixel values. Case in point—specifying the margin around the entire web page.

Typically, browsers leave a small blank area on the four sides of the window, which is a result of the `body` element having a default margin value of around 8 pixels. To change the body's margin from the default, you can provide a CSS rule for the `body` element's `margin` property. It's OK to use a CSS pixel value, but if you want to have the web page's margin shrink and grow when the user resizes the browser window by dragging a corner, use a percentage value, like this:

```
body {margin: 10%;}
```

The `10%` value indicates that the web page's left and right sides will have margin widths that each span 10% of the web page's width. Likewise, the web page's top and bottom sides will have margin heights that each span 10% of the web page's height.

When to Use the Different Length Units

In this chapter, we introduced various CSS units that are used for specifying the size, distance, or length of something. Specifically, we described `em`, `px`, `%`, and several absolute units such as `pc`, `in`, `cm`, and so forth. All those units are called *length units*. What follows is a summary of when to use each of the different length units:

- Use `em` for font-related properties (like `font-size`).
- Use `px` for properties that should be fixed for a given resolution, even if the element's font size changes or even if the containing element's size changes. Typically, that means using `px` for things like border properties and layout.
- Use `%` for properties that should grow and shrink with the size of the containing element (like margins for the `body` element).
- Use absolute units sparingly—only when a fixed size is essential and the target device has a high resolution.