Module 3: Cascading Style Sheets (CSS)

Introduction, CSS Overview , CSS Rules, Example with Type Selectors and the Universal Selector, CSS Syntax and Style, Class Selectors, ID Selectors, span and div Elements, Cascading, style Attribute, style Container, External CSS Files, CSS Properties, Color Properties, RGB Values for Color, Opacity Values for Color, HSL and HSLA Values for Color, Font Properties, line-height Property, Text Properties, Border Properties, Element Box, padding Property, margin Property , Case Study: Description of a Small City's Core Area.

Introduction

If you think appearance and format aren't all that important, think again. If your web page doesn't look good, people might go to it, but they'll leave quickly. An early exit might be OK if you're helping Grandma post her cat videos, but it's unacceptable for a business trying to generate revenue.

We put those things into practice by applying CSS rules to various elements, including span and div elements. We show you how to position those rules (1) at the top of the web page's main file or (2) in an external file. In the second half of the chapter, we describe CSS properties. Properties are the hooks used to specify the appearance of the elements within a web page. Specifically, we introduce CSS properties for color, font, and line height. Also, we introduce CSS properties for borders, padding, and margins.

CSS Overview

The W3C's philosophy in terms of how HTML and CSS should fit together is (1) use HTML elements to specify a web page's content, and (2) use CSS to specify a web page's appearance.CSS code is normally separated from web page content code. Specifically, web page content code goes in the body container, whereas CSS code goes either at the top of the web page in the head container or in an external file. Why is that separation strategy a good thing? Because if you want to change the appearance of something, it's easy to find the CSS code—at the top of the web page or in an external file.

CSS Rules

The way CSS works is that CSS rules are applied to elements within a web page. Browsers determine which elements to apply the CSS rules to with the help of selectors. There are Quite a few different types of selectors. For now, we'll introduce type selectors and the Universal selector. Type selectors are very popular. The universal selector is not as popular.

With a *type selector*, you use an element type (e.g., hr) to match all instances of that element type and then apply specified formatting features to those instances. For example, the following CSS rule uses a type selector with the hr element type and applies a width of 50% to all the hr elements in the current web page:

```
hr {width: 50%;}
```

A "width of 50%" means that for each hr element, its horizontal line will span 50% of the width of its enclosing container. Usually, but not always, the enclosing container will be the web page's body container.

Now for another type of selector—the *universal selector*. The universal selector uses the same syntax as the type selector, except that instead of specifying an element type, you specify *. The asterisk is a wildcard. In general, a wildcard is something that matches every item in a

collection of things. For CSS selector rules, the * matches every element in a web page's collection of elements. Here's an example universal selector CSS rule that centers
The text for every text-oriented element in the web page:

```
* {text-align: center;}
```

Even though the rule matches every element, because the property (text-align) deals with text, the rule affects only the elements that contain text.

Example with Type Selectors and the Universal Selector

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="author" content="John Dean">
<title>poem</title>
<style>
  * {text-align: center;}
  hr {width: 50%;}
  h2, p {font-style: italic; color: blue;}
</style>
</head>

<body>
<h2>Trees, by Joyce Kilmer</h2>
<hr>
<p>
  I think that I shall never see<br>
  A poem lovely as a tree.
</p>
</body>
</html>
```

universal selector

type selector with one element

type selector with comma-separated list of elements

CSS values

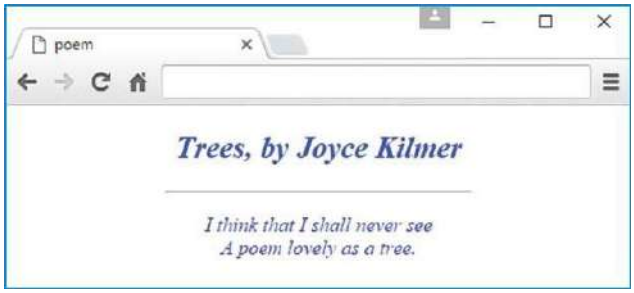
CSS properties

Figure 3.1 Source code for tree poem web page

Study the source code in Figure 3.1's Tree Poem web page. Notice the three CSS rules inside the style container. The first two rules should look familiar because they were presented in the previous section. The third rule uses a type selector with a slightly different syntax than before—there's a comma between two element types, h2 and p. If you want to apply the same formatting feature(s) to more than one type of element, you can implement that with one rule, where the element types appear at the left, as part of a comma- separated list

In Figure 3.1's three CSS rules, notice the four property-value pairs inside the {}'s, and copied here for your convenience:

- text-align: center
- width: 50%
- font-style: italic
- color: blue



In the Tree Poem web page, the * {text-align: center;} rule causes the elements that contain text to be centered. The hr element does not contain text, so it's not affected by the text align

property. Nonetheless, as you can see, it's also centered. That's because hr elements are centered by default.

The hr {width: 50%;} rule causes the horizontal line to render with a width that's 50% of the web page body's width.

Finally, the h2, p {font-style: italic; color: blue;} rule causes the heading and paragraph elements to be italicized and blue.

CSS Syntax and Style

CSS Syntax

In this section, we address CSS syntax details. First—the syntax for the style container. Refer back to Figure 3.1 and note how the three CSS rules are enclosed in a style container. Here's the relevant code:

```
<style>
  * {text-align: center;}
  hr {width: 50%;}
  h2, p {font-style: italic; color: blue;}
</style>
```

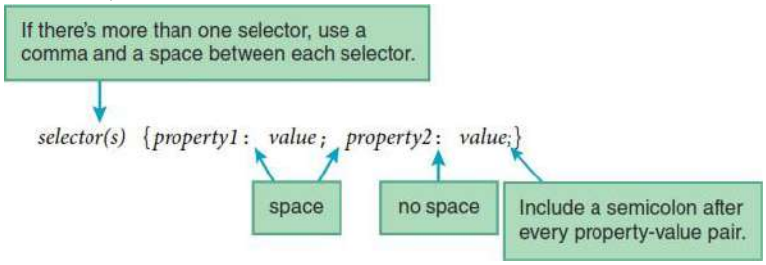
It's legal to position it in the body container, but don't do it. Coding conventions suggest positioning it at the bottom of the web page's head container. By following that convention, other web developers will be able to find your CSS rules quickly. In the style start tag, it's legal to include a type attribute with a value of "text/css", like this:

```
<style type="text/css">
```

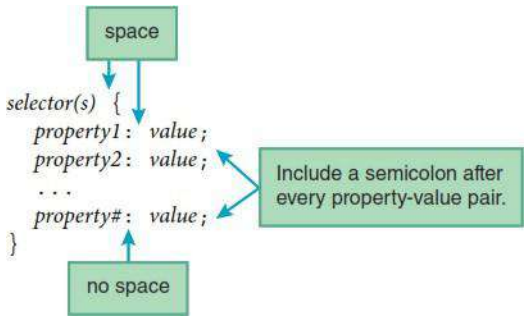
CSS Style

Now we'll look at some CSS guidelines that are not enforced by browsers or the HTML5 standard. They are style guidelines, and you should follow them so your code is easy to understand and maintain.

For short CSS rules, use this format:



Block formatting for CSS rules is similar in that the first and last lines are aligned at the left, and interior lines are indented. If you have a CSS rule that's kind of long (at least two or three property-value pairs), you should use block formatting like this:



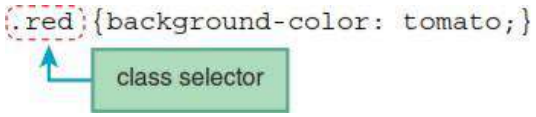
With both short and long CSS rules, the W3C CSS standard allows you to omit the semicolon

after the last property-value pair. However, coding conventions suggest that you should not omit the last semicolon—you should include it. That way, if another property-value pair is added later on, there will be less likelihood of accidentally forgetting to add a semicolon in front of the new property-value pair.

Class Selectors

Class Selector Overview

So far, we've talked about type selectors and the universal selector. We're now going to talk about a third type of CSS selector—a class selector. Let's jump right into an example. Here's a class selector rule with .red for its class selector and a background tomato color for matched elements:



The dot thing (.red in this example) is called a class selector because its purpose is to select elements that have a particular value for their class attribute. So the class selector rule would select/match the following element because it has a class attribute with a value of red:

```
<q class="red">It is better to keep your mouth closed and let people
    think you are a fool than to open it and remove all doubt.</q>
```

In applying the class selector rule to this element, the quote gets displayed with a tomato background color.

As with type selectors, you can have more than one class selector share one CSS rule. Just separate the selectors with commas and spaces, like this:

```
.red, .conspicuous, h1 {background-color: tomato;}
```

Note that in addition to a second class selector (.conspicuous), there's also a type selector (h1). In a single CSS rule, you can have as many comma-separated selectors as you like, all sharing the same set of property-value pairs.

With a type selector, your selector name (h1 in the this example) comes from the set of predefined HTML element names. But for a class selector, you make up the selector name. When you make up the selector name, make it descriptive, as is the case for red and conspicuous in the preceding example. As an alternative for red, you could get even more descriptive and use tomato. If you use tomato, that will be the same as the name used by the property value. There isn't anything wrong with that. Consistency is good.

Now let's look at class selectors in the context of a complete web page. In Figure 3.3, note the three CSS rules with their class selectors .red, .white, and .blue. Then take a look at the three q elements and their class attribute clauses class="red", class="white", and class="blue". Try to figure out what the web page will display before moving on to the next paragraph. In Figure 3.3, the first q element has a class attribute value of red, which means the .red CSS rule applies. That causes the browser to display the first q element with a tomato-colored background.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="author" content="John Dean">
<title>Mark Twain Quotes</title>
<style>
  .red {background-color: tomato;}
  .white {background-color: white;}
  .blue {background-color: skyblue;}
  q {font-family: Impact;}
</style>
</head>

<body>
<h1>Mark Twain Quotes</h1>
<q class="red">It is better to keep your mouth closed and
  let people think you are a fool than to open it and
  remove all doubt.</q><br>
<q class="white">Get your facts first, then you can distort
  them as you please.</q><br>
<q class="blue">Never put off till tomorrow what you can do
  the day after tomorrow.</q>
</body>
</html>
```

class selectors

class attribute

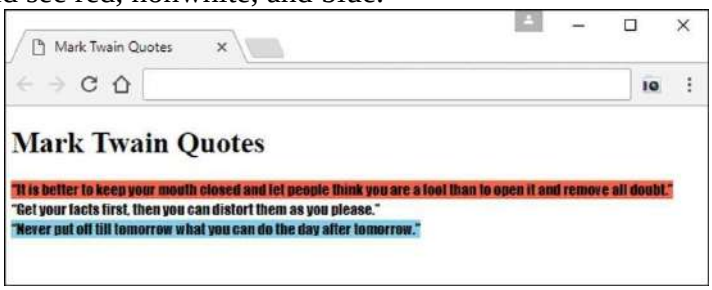
FIGURE 3.3 **Source code for Mark Twain Quotes web page**

I used a standard red background initially, but I found that the black text didn't show up very well. Thus, I chose tomato red, since it's lighter, and the color reminds me of my cherished home-grown tomatoes. Moral of the story: Get used to trying things out, viewing the result, and changing your code if appropriate.

The second and third q elements have class attribute values of white and blue. As you can see from the source code, that means they get matched with the .white and .blue class selector rules, and they get rendered with white and skyblue backgrounds, respectively. Take a look at Figure 3.4 and note the red, white, and blue background colors for the three quotes.

In addition to the three class selector rules, the Mark Twain Quotes web page also has a type selector rule, q {font-family: Impact;}. We'll discuss the font-family property later in his chapter, but for now, look at the Mark Twain quotes web page and observe the thick block lettering for the three q elements. That lettering is from the Impact font.

Usually, browsers use a default background color of white, so why did we specify white for the second q element's background color? One benefit is that it's a form of self-documentation. Another benefit is that it would handle a rogue browser with a nonwhite default background color. With such a browser, if there were no explicit CSS rule for the white background color, then the user would see red, nonwhite, and blue.



class Selectors with element type prefixes

element-type.class-value {property1: value; property2: value;}

And here's an example CSS rule that uses a class selector with an element type prefix:

q.blue {background-color: skyblue;}

Because q.blue has .blue in it, q.blue matches elements that have a class attribute value of "blue". But it's more granular than a standard class selector in that it looks for class="blue" only in q elements.

Figure 3.5 shows a modified version of the style container for the Mark Twain Quotes web page. It uses four class selectors with element type prefixes. How will that code change the appearance of the web page, compared to what's shown in Figure 3.4? The original style container used the simple class selector rule .blue {background-color: skyblue;}. That caused all elements with class="blue" to use the CSS color named skyblue. But suppose

```
<style>
  h1.blue {color: blue;}
  q.red {background-color: tomato;}
  q.white {background-color: white;}
  q.blue {background-color: skyblue;}
  q {font-family: Impact;}
</style>
```

FIGURE 3.5 Improved style container for Mark Twain Quotes web page

you want a different shade of blue for the "Mark Twain Quotes" header. You could use a distinct class attribute value for the header, like "header-blue," but having such a specific class attribute value would be considered poor style because it would lead to code that is harder to maintain. Specifically, It would be hard to remember a rather obscure name like "header-blue." So, what's the better approach? As Shown in Figure 3.5, it's better to use separate h1.blue and q.blue class selectors with element type prefixes. Note how the h1.blue rule specifies a background color of blue, and the q.blue rule specifies a background color of skyblue.

Figure 3.5's style container uses a class selector with an element prefix, q.red, whereas the original style container used a simple class selector, .red. Because there's only one element that uses class="red", the .red class selector was sufficient by itself; however, using q.red (and also q.white) makes the code parallel for the three q element colors. More importantly, using a class selector with an element prefix makes the code more maintainable. Maintainable code is code that is relatively easy to make changes to in the future. For example, suppose you decide later that you want a different shade of red for an h2 element. You can do that by using q.red and h2.red.

Class Selectors with * prefixes

Instead of prefacing a class selector with an element type, as an alternative, you can preface a class selector with an *. Because * is the universal selector, it matches all elements. Therefore, the following CSS rule is equivalent to a standard class selector rule (with no prefix):

*.class-value {property1: value; property2: value;}

So what would the following CSS rule do?

*.big-warning {font-size: x-large; color: red;}

It would match all elements in the web page that have a class attribute value of big warning, and it would display those elements with extra-large red font. In the preceding CSS rule, note the hyphen in the *.big-warning class selector rule. HTML5 standard does not allow spaces within class attribute values, so it would have been illegal to use *.big warning. If You want to use multiple words for a class attribute value, coding conventions suggest that you use hyphens to separate the words, as in big-warning.

ID Selectors

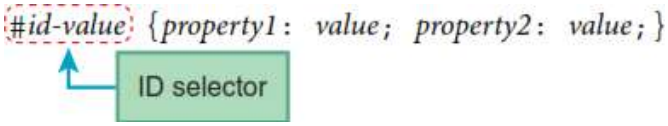
An ID selector is similar to a class selector in that it relies on an element attribute value in searching for element matches. As you might guess, an ID selector uses an element's id attribute (as opposed to a class selector, which uses an element's class attribute). A significant feature of an id attribute is that its value must be unique within a particular web page. That's different from a class attribute's value, which does not have to be unique within a particular web page. The ID selector's unique-value feature means that an ID selector's CSS rule matches only one element on a web page.

Suppose you want the user to be able to link/jump to the "Lizard's Lounge" section of your web page. To do that, you'd need a link element (which we'll discuss in a later chapter) and also an element that serves as the target of the link. Here's a heading element that could serve as the target of the link:

```
<h3 id="lizards-lounge">Lizards Lounge</h3>
```

In this code, note the id attribute. The link element (not shown) would use the id attribute's value to indicate which element the user jumps to when the user clicks the link. For the jump to work, there must be no confusion as to which element to jump to. That means the target element must be unique. Using an id attribute ensures that the target element is unique.

let's examine how to apply CSS formatting to an element with an id attribute. As always with CSS, you need a CSS rule. To match an element with an id attribute, you need an ID selector rule, and here's the syntax:



The syntax is the same as for a class selector rule, except that you use a pound sign (#) instead of a dot (.), and you use an id attribute value instead of a class attribute value.

How would the following ID selector rule affect the appearance of the Lizard's Lounge heading?

```
#lizards-lounge {color: green;}
```

This rule would cause browsers to display the Lizards Lounge heading with green font.

span and div Elements

No matter which selector you choose, you can apply it only if there's an element in the web page body that matches it. But suppose you want to apply CSS to text that doesn't coincide with any of the HTML5 elements. What should you do?

If you want to apply CSS to text that doesn't coincide with any of the HTML5 elements, put the text in a span element or a div element. If you want the affected text embedded within surrounding text, use span (since span is a phrasing element). On the other hand, if you want the text to span the width of its enclosing container, use div (since div is a block element).

See Figure 3.6 and note how the div and span elements surround text that doesn't fit very well with other elements. Specifically, the div element surrounds several advertising phrases that describe Parkville's world-famous Halloween on the River celebration, and the two span elements surround the two costs, $10 and $15.

```
<span class="white orange-background">$10</span>
```

In particular, note that there are two class selectors for the class attribute's value—white and orange background. As you'd expect, that means that both the white and orange background CSS rules get applied to the span element's content. Note that the two class selectors are separated with spaces.

In the Pumpkin Patch web page, there are competing CSS rules for the two costs, $10 and $15. The div container surrounds the entire web page body, so it surrounds both costs, and it attempts to apply its orange text rule to both costs. The first span container surrounds the first cost; consequently, the first span container attempts to apply its white text rule to the first cost. Likewise, the second span container surrounds the second cost; consequently, the second span container attempts to apply its black text rule to the second cost. So, what colors are used for the span text—white and black from the span containers or orange from the div container? As you can see in Figure 3.7's browser window, the "$10" cost text is white, and the "$15" cost text is black. That means that the more local CSS rules (the two span rules) take precedence over the more global CSS rule (the div rule). The span rules are considered to be more local because their start and end tags immediately surround the cost content. In other words, their tags surround only their cost content and no other content. The div rule is considered to be more global because its start and end tags do not immediately surround the cost content. In other words, their tags surround not only the cost content, but also additional content. This principle of locality, where local things override global things, parallels the nature of the "cascading" that takes place in applying CSS rules.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="author" content="John Dean">
<title>Halloween on the River</title>
<style>
  .orange {color: darkorange;}
  .white {color: white;}
  .black {color: black;}
  .orange-background {background-color: orange;}
</style>
</head>

<body>
<div class="orange">
  Parkville's Halloween on the River, every weekend in October.<br>
  Corn maze: <span class="white orange-background">$10</span><br>
  All you can eat pumpkins:
  <span class="black orange-background">$15</span>
</div>
</body>
</html>
```

div

Multiple class selectors for a class attribute's value.

span

FIGURE 3.6 **Source code for Pumpkin Patch web page**

FIGURE 3.7 **Pumpkin Patch web page**

Cascading

Traditionally, a "style sheet" is a collection of rules that assign appearance properties to structural elements in a document. For a web page, a style sheet "rule" refers to a value assigned to a particular display property of a particular HTML element.

Each stage/place has its own set of rules, and each set of rules is referred to as a style sheet. With multiple style sheets organized in a staged structure, together it's referred to as Cascading Style Sheets. To handle the possibility of conflicting rules at different places, different priorities are given to the different places. See Figure 3.8, which shows the places where CSS rules can be defined. The higher priority places are at the top, so an element's style attribute (shown at the top of the CSS rules list) has the highest priority. We'll explain the style attribute in the next section, but let's first do a cursory run-through of the other items in Figure 3.8's list.

The next place for CSS rules is in the settings defined by a user for a particular browser installation.

The last place for CSS rules, and the place with the lowest priority, is in the native default settings for the browser that's being used. As a programmer, there's nothing you can do to modify a browser's native default settings.

Places Where CSS Rules Can Be Defined, Highest to Lowest Priority
1. In an element's `style` attribute.
2. In a `style` element in the web page's head section.
3. In an external file.
4. In the settings defined by a user for a particular browser installation.
5. In the browser's native default settings.

FIGURE 3.8 **Places where CSS rules can be defined**

`style` Attribute, `style` Container

`style` attribute

when you use the style attribute for CSS rules, those rules are given the highest priority. Here's an example element that uses a style attribute:

```
<h2 style="text-decoration:underline;">Welcome!</h2>
```

As you can see, using the style attribute lets you insert CSS property-value pairs directly in the code for an individual element. So the preceding h2 element—but no other h2 elements would be rendered with an underline.

The style attribute is a global attribute, which means it can be used with any element. Even though it's legal to use it with every element, and you'll see it used in lots of legacy code, you should avoid using it in your pages. Why? Because it defeats the purpose of CSS—keeping presentation separate from content.

Let's imagine a scenario that demonstrates why the style attribute is bad. Suppose you embed a style attribute in each of your p elements so they display their first lines with an indentation (later on, you'll learn how to do that with the text-indent property). If you want to change the indentation width, you'd have to edit every p element. On the other hand, making such a change is much easier when the CSS code is at the top of the page in the head container because you only have to make the change in one place—in the p element's class selector rule. If you make the change there, it affects the entire web page.

`style` Container

The browser applies the CSS rules' property values by matching the CSS rules' selectors with elements in the web page. Normally, you should have just one style container per page, and you should put it in a web page's head container. It's legal to put a style container in the body, but don't do it because then it's harder to find the CSS rules.

More specific rules beat more general rules. For example, if a style attribute designates a paragraph as blue, but a rule in a style container designates paragraphs as red, then what color will the browser use to render the paragraph? The style attribute's blue color wins, and the browser renders that particular paragraph with blue text. This principle of more specific rules beating more general rules should sound familiar. It parallels the principle introduced earlier that says local things override global things.

External CSS Files

Overview

In general, splitting up a big thing into smaller parts makes the thing easier to understand. To improve understandability, you should consider moving your CSS rules to an external file. There are two steps necessary to tie a web page to an external file that contains CSS rules. First, for the external file to be recognized as a CSS file, the external file must be named with a .css extension. Second, for the web page to access a CSS file's CSS rules, the web page must use a link element in the web page's head container. The link element is a void element, so it's comprised of just one tag, with no end tag. Here's the syntax:

```
<link rel="stylesheet" href="name-of-external-file">
```

Note the rel="stylesheet" attribute-value pair. rel stands for "relationship," and its value tells the browser engine what to do with the href file. Having a rel value of stylesheet tells the browser engine to look for CSS rules in the href file and apply them to the current web page.

To justify the extra work of adding a link element to handle an external CSS file, typically an external CSS file will be nontrivial. That means the file will contain at least five CSS rules (usually a lot more), or it will be shared by more than one web page. Why is sharing an external CSS file helpful? With a shared external CSS file, it's easy to ensure
that all the web pages on your site follow the same common CSS rules. And if you want to change those rules, you change them in one place, in the external file, and the change affects all the web pages that share the external file.

CSS Properties

from prior examples, a CSS property specifies one aspect of an HTML element's appearance. Note the first keyword entry, :active. The keywords that start with a colon are known as pseudo-elements.

Color properties	color, background color
Font properties	font-style, font-variant, font-weight, font-size, font-family, font
Text properties	line-height, text-align, text-decoration, text-transform, text-indent
Border properties	border-bottom, border-bottom-color, ...
Margin and padding properties	margin-bottom, margin-left, ... padding-bottom, padding-left, ...

FIGURE 3.13 **CSS properties introduced in this chapter**

Color Properties

The color property specifies the color of an element's text. The background-color property specifies the background color of an element. You can specify a color value using one of five different formats.

- color name—for example, red
- RGB value—specifies amounts of red, green, and blue
- RGBA value—specifies red, green, and blue, plus amount of opacity
- HSL value—specifies amounts of hue, saturation, and lightness
- HSLA value—specifies hue, saturation, and lightness, plus amount of opacity

Color Names

The CSS3 specification defines 147 color names, and the major browsers support all those colors

```
<head>
<style>
  .roofColor {color: darkslategray;}
</style>
</head>

<body>
<p>
  Mackay Hall's roof is
  <span class="roofColor">dark slate gray</span>.
</p>
</body>
```

color name

RGB Values for Color

RGB stands for red, green, and blue. An RGB value specifies the amounts of red, green, and blue that mix together to form the displayed color. To specify an amount of a color, you can use a percentage, an integer, or a hexadecimal number

percentage—0% to 100% for each color
integer—0 to 255 for each color
hexadecimal—00 to ff for each color

RGB Values with percentages

To specify an RGB value with percentages, use this format:

rgb(red-percent, green-percent, blue-percent)

Each percent value must be between 0% and 100%. Here's an example class selector rule that uses an RGB value with percentages:

Eggplants are dark purple, and that's why we use eggplant in the preceding class selector rule.

If you want to specify black, then you need to use the least intensity (a value of 0%) for each of the three colors

.black {background-color: rgb(0%,0%,0%);}

To specify white, you need to use the greatest intensity (a value of 100%) for each of the three colors.

.white {color: rgb(100%,100%,100%);}

RGB Values with Integers

To specify an RGB value with integers, use this format:

rgb(red-integer,green-integer,blue-integer)

Each integer value must be between 0 and 255, with 0 providing the least intensity and 255 providing the most. Here are two class selector rules that use RGB values with integers:

```
<style>
   .favorite1 {color: rgb(144,238,144);}
   .favorite2 {color: rgb(127,127,127);}
</style>
```

light green- favorite 1, favorite2 produces a medium gray color

RGB Values with hexadecimal

With many programming languages, including HTML and CSS, numbers can be represented not only with base-10 decimal numbers, but also with base-16 hexadecimal (hex) numbers. A number system's "base" indicates the number of unique symbols in the number system. base 16 (for the hexadecimal number system) means there are 16 unique symbols—0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.

When specifying an RGB color value, you have choices. You can use percentages or standard integers as described earlier, or you can use hexadecimal values. For hexadecimal RGB values, you'll need to use the format #rrggbb where:

rr = two hexadecimal digits that specify the amount of red
gg = two hexadecimal digits that specify the amount of green
bb = two hexadecimal digits that specify the amount of blue

With the percentage and integer RGB values, a smaller number for a particular color means less of that color. Likewise, with hexadecimal RGB values, a smaller number means less of a particular color. The smallest hexadecimal digit is 0, so 00 represents the absence of a particular color. If all colors are absent, that's black. Therefore, #000000 (00 for each of the three colors) indicates black. The largest hexadecimal digit is f, so ff represents the greatest intensity of a particular color. If all colors are maximally intense, that's white. Therefore, #ffffff (ff for each of the three colors) indicates white.

That means #ffbbbb is light red (otherwise known as pink), and it looks like this:

Opacity Values for Color

The fourth value, opacity, determines how opaque the color is, where opaque refers to the inability to see through something. It's the opposite of transparency. If the opacity value is 100%, that

means the color is completely opaque, and if there is content behind the color, that content gets covered up. At the other extreme, if the opacity value is 0%, that means the color is completely transparent. To specify an RGBA value, use one of these two formats:

rgba(red-integer,green-integer,blue-integer,opacity-number-between-0-and-1)
rgba(red-percent,green-percent,blue-percent,opacity-number-between-0-and-1)

For both formats, the fourth value specifies the opacity. The opacity value must be in the form of a decimal number between 0 and 1, with 0 being completely transparent, 1 being completely opaque, and .5 in between.

For the first format, each integer value must be between 0 and 255, with 0 providing the least intensity and 255 providing the most. That should sound familiar because that was also the case for integers with the rgb construct. For the second format, each percent value must be between 0% and 100%.

Figure 3.14's Opacity Example web page illustrates what happens when a transparent yellow color is placed on top of a red background—orange is formed. Before looking at the CSS rules that generate the yellow colors, let's first examine the CSS rule that generates the window's red background color:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="author" content="John Dean">
<title>Opacity Example</title>
<style>
  .red {background-color: red;}
  .yellow-bg {background-color: rgb(255,255,0);}
  .yellow-bg2 {background-color: rgba(255,255,0,.4);}
  .yellow-bg3 {background-color: rgb(255,255,0); opacity: .4;}
</style>
</head>
<body class="red">
<p class="yellow-bg">This line uses background-color: rgb(255,255,0);</p>
<p class="yellow-bg2">
  This line uses background-color: rgba(255,255,0,.4);
</p>
<p class="yellow-bg3">
  This line uses background-color: rgb(255,255,0); opacity: .4;
</p>
</body>
</html>
```

rgba construct

opacity property

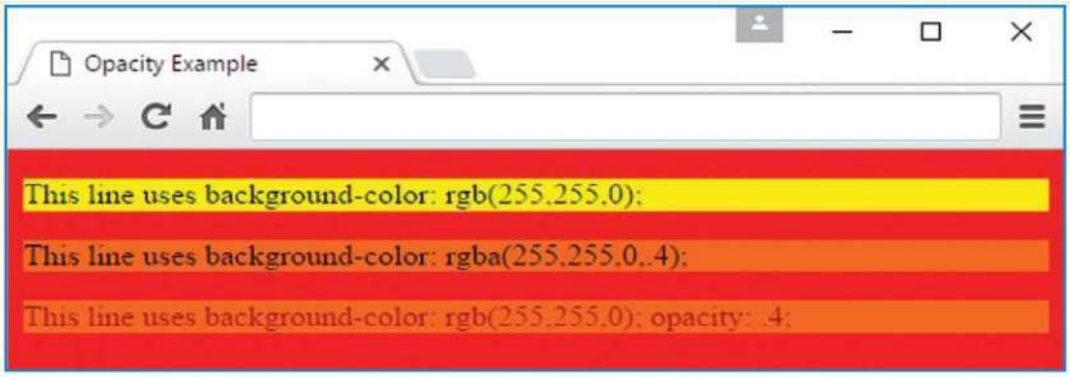


FIGURE 3.14 An opacity example that illustrates the `rgba` construct and the `opacity` property

Note the browser window's first sentence and the CSS rule that generates its yellow background: Next, note the browser window's second sentence and the CSS rule that generates its orange background:

HSL and HSLA Values for Color

Here's the syntax:

hsl(hue-integer,saturation-percent,lightness-percent)

HSL stands for hue, saturation, and lightness. Hue is a degree on the color wheel shown in Figure 3.15. The wheel is a circle, so the wheel's degrees go from 0 to 360. As you can see in the figure, 0 degrees is for red, 120 degrees is for green, and 240 degrees is for blue. For a circle, 0 degrees is equivalent to 360 degrees. The second value in the hsl construct is the color's percentage of saturation. The W3C says 0% means a shade of gray, and 100% is the full color. The third value in the hsl construct is the color's percentage of lightness. A lightness value of 0% generates black, regardless of the values for hue and saturation. A lightness value of 100% generates white, regardless of the values for hue and saturation. A lightness value of 50% generates a "normal" color.

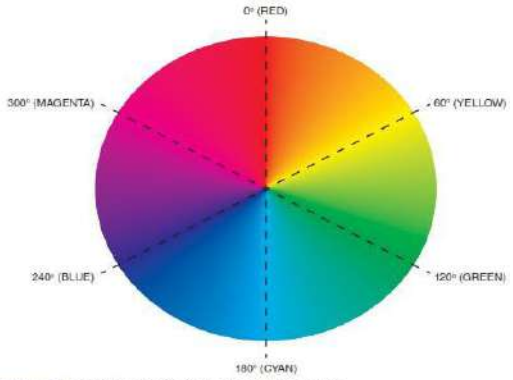


FIGURE 3.15 Color wheel for the hsl construct's hue value

Here's an example CSS rule with an HSL color value:

```
<style>
   p {background-color: hsl(120,50%,75%);}
</style>
```

That color forms a light shade of grayish green, and here's what it looks like:

To add transparency to an RGB value by using the rgba construct. Likewise, to add transparency to an HSL value, you can use the hsla construct. Here's the syntax:

hsla(hue-integer,saturation-percent,lightness-percent,opacity-number-between-0-and-1)

The fourth argument specifies the opacity. The opacity value must be in the form of a decimal number between 0 and 1, with 0 being completely transparent and 1 being completely opaque. Here's an example CSS rule with an HSLA color value:

```
<style>
  .background {background-color: hsla(120,50%,75%,.5);}
</style>
```

It's the same as the earlier grayish-green color, except that this time, the grayish green blends with the web page's background color as a result of the 50% opacity value. With a default white web page background, the result would be a lighter shade of grayish green, like this:

Font Properties

font refers to the characteristics of text characters—height, width, thickness, slantedness, body curvatures, and endpoint decorations. That should make more sense later on when we present CSS font property details and show examples. Specifically, you'll learn about the font-style, font-variant, font-weight, font-size, font- family, and **font** shorthand properties.

`font-style` property

The font-style property specifies whether the text is to be displayed normally or slanted.

font-style Values	Description
normal	Upright characters (not slanted).
oblique	Use the same font as the current font, but slant the characters.
italic	Use a cursive font (which tends to be slanted and is supposed to look like handwriting).

These descriptions indicate a slight difference between the oblique and italic properties, with italic tending to be more decorative. Most web developers use the value italic. Because italics are so common, you should memorize the following technique for generating italics:

```
.italics {font-style: italic;}
```

As always, choose a name for the class selector that's descriptive. Here, we chose the name italics because it's descriptive and easy. Upright (normal) characters are the default, so why would you ever want to specify normal for the font-style property? Suppose you have a whole paragraph that's italicized and you want one word in the paragraph not italicized. To make that word normal (not italicized), you can use font-style: normal.

`font-variant` property

The font-variant property specifies how lowercase letters are displayed.

font-variant Values	Description
normal	Display lowercase letters normally.
small-caps	Display lowercase letters with smaller-font uppercase letters.

Here's an example that uses a small-caps CSS rule:

```
.title {font-variant: small-caps;}
...
<div class="title">The Great Gatsby</div>
```

And here's the resulting displayed text:

THE GREAT GATSBY

`font-weight` property

The font-weight property specifies the boldness of the text characters. With bolder and lighter, the targeted text inherits a default fontweight value from its surrounding text, and then the targeted text's weight gets adjusted up or down relative to that inherited weight. For example, if you specify a bold font weight for a paragraph, you can make a particular word within the paragraph even bolder by Specifying bolder for that word's font weight. Because boldfacing is such a common need, you should memorize the following technique for making something bold:

```
.bold {font-weight: bold;}
```

font-weight Values	Description
normal, bold	It's up to the browser to determine a font weight that can be described as normal or bold.
bolder, lighter	Using a value of bolder causes its targeted text to have thicker characters than the text that surrounds it. Using a value of lighter causes its targeted text to have thinner characters than the text that surrounds it.
100, 200, 300, 400, 500, 600, 700, 800, 900	100 generates the thinnest characters and 900 the thickest characters. 400 is the same as normal, and 700 is the same as bold.

font-size property

The font-size property specifies the size of the text characters.

font-size Values	Description
xx-small, x-small, small, medium, large, x-large, xx-large	It's up to the browser to determine a font size that can be reasonably described as xx-small, x-small, small, etc.
smaller, larger	Using a value of smaller causes its targeted text to have smaller characters than the text that surrounds it. Using a value of larger causes its targeted text to have larger characters than the text that surrounds it.
number of em units	One em unit is the height of the element's normal font size.

Here's an example class selector rule that uses the font-size property with an xx-large value:

```
.huge-font {font-size: xx-large;}
```

The em unit's name comes from the letter M. Originally, one em unit equaled the height of the letter M.

Here are class selector rules that use the font-size property with em values:

```
.disclaimer {font-size: .5em;}
.advertisement {font-size: 3em;}
```

.5em value displays text that is half the size of normal text. The second rule is for advertisement text, which is supposed to be annoyingly large to draw attention, and its 3em value displays text that is three times the size of normal text.

font-family property

The font-family property allows the web developer to choose the set of characters that the browser uses when displaying the element's text. Here's an example class selector rule that uses the font-family property:

```
.ascii-art {font-family: Courier, Prestige, monospace;}
```

Note that with the font-family property, you should normally have a comma-separated list of fonts, not just one font. In applying the preceding rule to elements that use "asci-art" for their class attribute, the browser works its way through the font-family list from left to right, and uses the first font value it finds installed on the browser's computer and skips the other fonts. So if Courier and Prestige are both installed on a computer, the browser uses the Courier font because it appears further left in the list.

A generic font is a name that represents a group of fonts that are similar in appearance. For example, monospace is a generic font, and it represents all the fonts where each character's width

is uniform. Whenever you use a font-family CSS rule, you should include a generic font at the end of the rule's list of font names.

```
.ascii-art {font-family: Courier, Prestige, monospace;}
```

Generic Font Names	Description	Example Font
`monospace`	All characters have the same width.	`Courier New looks like this.`
`serif`	Characters have decorative embellishments on their endpoints.	Times New Roman looks like this.
`sans-serif`	Characters do not have decorative embellishments on their endpoints.	Arial looks like this.
`cursive`	Supposed to mimic cursive handwriting, such that the characters are partially or completely connected.	*Monotype Corsiva looks like this.*
`fantasy`	Supposed to be decorative and playful.	**Impact looks like this.**

`font` Shorthand property

Fairly often as a web programmer, you'll want to apply more than one of the prior font-related properties to an element. You could specify each of those font properties separately, but there's an easier way. The font property can be used to specify all these more granular font properties— font-style, font-variant, font-weight, font-size, line-height, and font- family. Here's the syntax for a font property-value pair:

```
font: [font-style-value] [font-variant-value] [font-weight-value]
      font-size-value [/line-height-value] font-family-value
```

As usual, the italics are the book's way of telling you that the italicized thing is a description of what goes there, so for a font property-value pair in a CSS rule, you would replace font-style value with one of the font-style values, such as italic. The square brackets are the book's way of telling you that the bracketed thing is optional, so the font-style, font-variant, font-weight, and line-height values are all optional. On the other hand, the font-size and font-family values have no square brackets, so you must include them whenever you use the font property.

Here's an example type selector rule that uses a font property:

```
blockquote {
  font: italic large "Arial Black", Helvetica, sans-serif;
}
```

Use spaces to separate property values.

Use commas to separate `font-family` values.

`line-height` Property

It's used to specify the vertical separation between each For example, in Figure 3.17, note the sentence1 CSS rule with line-height: 2em. That rule causes its matching element to display its lines with a vertical separation equal to twice the height of a normal character (remember, an em unit represents the height of a normal character). Note the resulting double-spaced line heights in Figure 3.18's displayed text. line of text in an element.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="author" content="John Dean">
<title>Declaration of Independence</title>
<style>
  .sentence1 {line-height: 2em; font-family: Times, serif;}
  .sentence2 {font: 1em/2em Times, serif;}
</style>
</head>

<body>
<h2>Declaration of Independence, working draft v. 2.01</h2>
<p>
  <span class="sentence1">We hold these truths to be self-evident,
    that <del>every man is</del><ins>all men are</ins>
    created equal.</span>
  <br>
  <span class="sentence2">That they are endowed by their Creator
    with certain unalienable Rights, that among these are Life,
    Liberty and the pursuit of Happiness.</span>
</p>
</body>
</html>
```

A line height of 2em generates double spacing.

This specifies font size and then line height.

FIGURE 3.17 **Source code for Declaration of Independence web page**

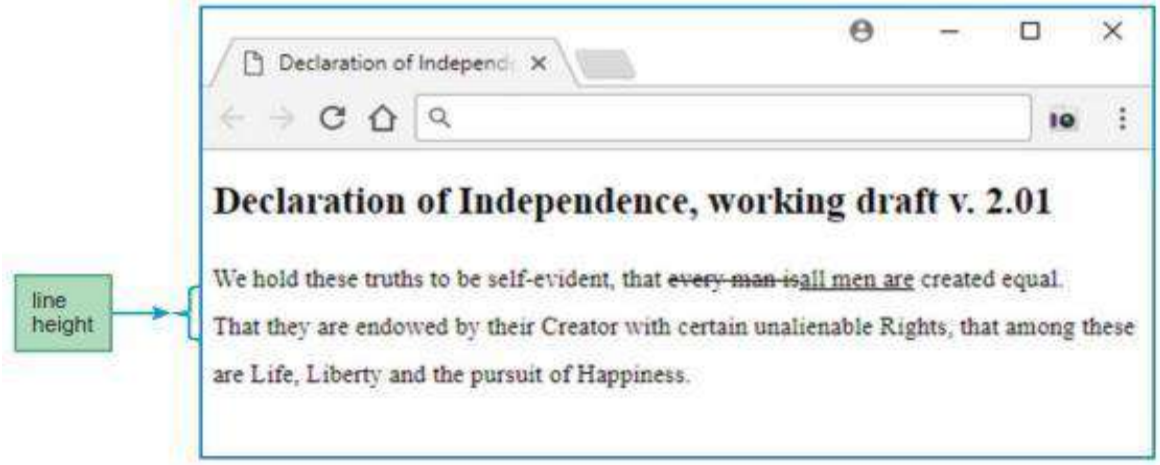


FIGURE 3.18 **Declaration of Independence web page**

Text Properties

with text properties, we'll focus on appearance characteristics of groups of characters. Specifically, here's what's on the agenda—text-align, text- decoration, text-transform, and text-indent.

`text-align` property

The text-align property specifies the horizontal alignment for a block of text If you use justify for the text-align property, the browser stretches all the lines in a block of text, except for the block of text's bottom line. The bottom line uses left justification. That behavior mimics what you see for paragraphs in newspapers and magazines, and that's why justify is used primarily for p elements.

text-align Values	Description
left	Align the text at the left.
right	Align the text at the right.
center	Center the text.
justify	Stretch the lines so that each line extends to the left edge and the right edge.

text-decoration property

The text-decoration property specifies something decorative that is added to text.

text-decoration Values	Description
none	This displays normal text (no decoration added).
underline	Draw a line below the text.
overline	Draw a line above the text.
line-through	Draw a line through the text.
blink	This causes the text to blink.

Because underlining is so common, you should memorize the following technique for generating an underline:

```
.underlined {text-decoration: underline;}
```

text-transform property

The text-transform property controls the text's capitalization.

text-transform Values	Description
none	The text renders the same as the original text.
capitalize	Transform the first character of each word to uppercase.
uppercase	Transform all characters to uppercase.
lowercase	Transform all characters to lowercase.

You might want to provide uppercase and lowercase buttons on your web page that allow users to dynamically change the page so it displays all uppercase or all lowercase. You can implement that with JavaScript and the text-transform property.

text-indent property

The text-indent property specifies the size of the indentation of the first line in a block of text. The block's second and third lines (and so on) are unchanged; that is, they do not get indented. If you want to adjust all the lines in a block of text, use the margin property, not the text-indent property. You'll learn about the margin property later in this chapter.

The most appropriate way to specify a value for the text-indent property is to use em units. Here's an example type selector rule that uses the text-indent property:

```
p {text-indent: 4em;}
```

Border Properties

As expected, the border properties allow You to specify the appearance of borders that surround elements. 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.

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;}
```

border-width property

The border-width property specifies the width of the border that surrounds the matched element.

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;
}
```

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. 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.

```
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.

`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.

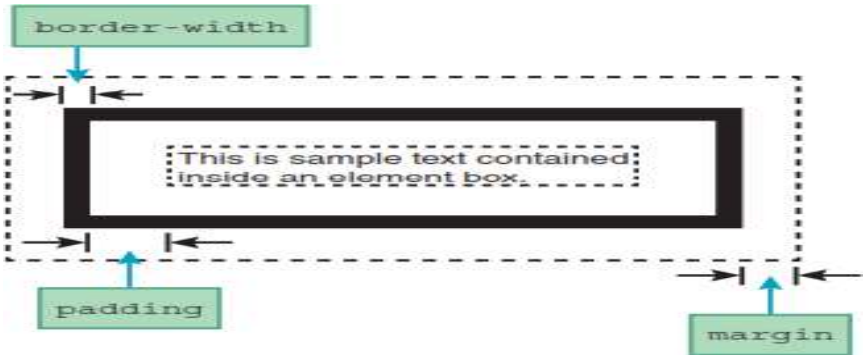
`border` Shorthand property

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;}
```

Element Box, `padding` Property, `margin` Property

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 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.

`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.

```
.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

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 borer. 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;
}
```

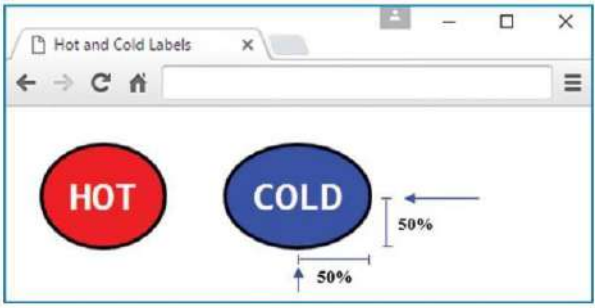


FIGURE 3.20 **Hot and Cold web page**

```
<!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.

The padding and margin properties both use values of 20px, which provides significant space on the interior and exterior of the borders. 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: inline block property-value pair to the label CSS rule. 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.

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 mar gin 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

- 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.

Question bank

1. Briefly explain CSS Syntax and Style
2. Suppose you want to apply CSS to text that doesn't coincide with any of the HTML5 elements. What should you do? Give an illustrative example.
3. What are the different methods used specify the RGB Values for Color explain any one of them.
4. Discuss font-style, font-variant,
5. What are the different methods used specify the RGB Values for Color explain any one of them.
6. Discuss font-style, font varient. font-weight properties of CSS.
7. Explain Text-align, text-decoration text-transform, and text-indent properties of CSS.
8. Explain briefly Element Box, padding Property, margin Property
9. with an example explain different levels of style sheets.
1. list the different selectors available in CSS and explain in detail.
2. Explain the need of cascade in CSS. Illustrate three principles of cascade with suitable CSS script segments.
3. Explain class selector of CSS with relevant scripts.
4. Define CSS and list out its benefits with explanation
5. What are selectors? List and explain selectors with an example.
6. With example explain the location of styles.
7. Write the division based HTML sematic structure elements.
8. Define CSS. Explain the location of styles.
9. Illustrate the CSS box model besuge to label each of the components of the box.