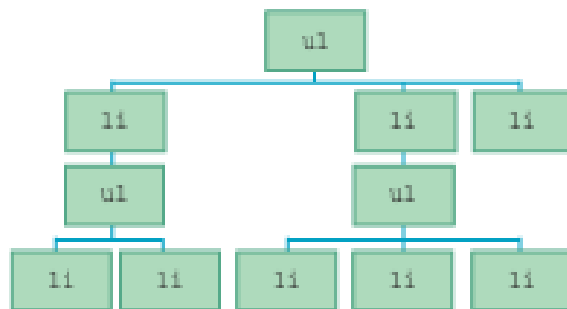


UNIT-IV

Descendant selector

A descendant selector is when you specify a series of two or more selectors separated by spaces. For each pair of adjacent selectors, the browser searches for a pair of elements that match the selectors such that the second element is contained within the first element's start tag and end tag. When an element is inside another element's start tag and end tag, we say that the element is a descendant of the outer element. To better understand the descendant selector, let's look at an example. The following structure shows how the Work Day web page's ul and li elements are related:



The ul element at the top is for the outer list. The three li elements below it are for the morning, afternoon, and evening list items. Each of the first two list elements contains a sublist, built with its own ul and li elements. The descendant relationship between two elements mimics the descendant relationships you can find in a family tree. Imagine that this structure shown is a family tree of bacteria organisms. Why bacteria? Because bacteria have only one parent, just as HTML elements have only one parent. All the elements below the top ul element are considered to be descendants of the top ul element. On the other hand, only the three li elements immediately below the top ul element are considered to be child elements of that ul element. So for an element to be a child of another element and not just a descendant, it has to be immediately below the other element.

Here's the syntax for a descendant selector rule: space-separated-list-of-elements {property1: value; property2: value;} In the following style container, note how the second and third rules use that syntax: `<style>ul {list-style-type: disc;} ulul {list-style-type: square;} ululul {list-style-type: none;} </style>` In applying the three preceding rules to a web page, the browser would use the first rule to generate bullet symbols for list items at the outer level of an unordered outline. It would use the second rule to generate square symbols for list items at the first level of nesting within an unordered outline. And it would use the third rule to display no symbols for list items at the next level of nesting within an unordered outline.

If you ever have two or more CSS rules that conflict, the more specific rule wins.² That's a general principle of programming and you should remember it—the more specific rule wins. In the preceding style container, the first rule (the one with ul for its selector) applies to all ul elements, regardless of where they occur. On the other hand, the second rule (the one with ulul for its selector) applies only to ul elements that are descendants of another ul element. Those rules conflict because they both apply to ul elements that are descendants of another ul element. For those cases, the second rule wins because the second rule is more specific. The third rule introduces another conflict—this time when there is a ul element that is a descendant of another ul element, and that other ul element is a descendant of a third

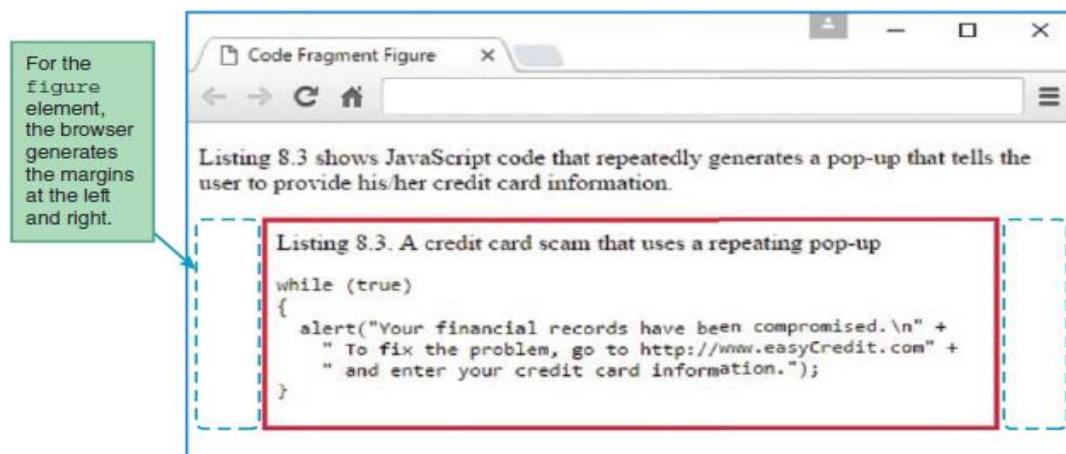
element. For those cases, the third rule wins because the third rule is more specific. In the preceding examples, we use descendant selectors to specify the different levels for nested lists. But be aware that you can use descendant selectors for any element types where one element is contained in another element. In expository writing,³ if you use a new word in a paragraph, it's common practice to italicize the word and then define it. What descendant selector CSS rule could you use to support this practice? Try to come up with this on your own before you look down. . . . Assuming you have spent sufficient time trying to figure it out on your own, now you're allowed to proceed. Here's the answer: `p dfn {font-style: italic;}`

Figures:

In this section, you'll learn how to implement a figure. Typically, a figure holds text, programming code, an illustration, a picture, or a data table. As with all figures, the figure element's content should be self-contained, and it should be referenced from elsewhere in the web page.

Figure with a Code Fragment

Take a look at FIGURE . It uses the figure element to display a listing of programming code that's offset from the regular flow of the web page. The programming code is in JavaScript, which you'll learn about in later.



For now, there's no need to worry about what the JavaScript code means. Instead, just focus on the figure element's syntax. In FIGURE , note the figure element's start tag and end tag. Also, note the figcaption element inside the figure container. As its name implies, the figcaption element causes the browser to display a caption for a figure. In the browser window's red-bordered figure, you can see the caption at the top of the figure.

In above Figure browser window, note how the red-bordered figure has expansive equal-sized

margins at the left and right. The browser generates those margins by default for the figure element. But what's not a default is the visibility of the figure element's border. To make the border visible with a reddish color and a reasonable amount of padding inside it, it was necessary to add this CSS rule:

```
figure {  
border: solid crimson;  
padding: 6px;  
}
```

Figure with an ImageNext, let's use the figure element to display a picture with a caption. For an example, see the Final Tag web page in FIGURE and its source code in FIGURE. The figure element code and the figcaption element code should look familiar. That code is the same as for the Code Fragment Figure web page, except that the figcaption element is at the bottom of the figure

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
<meta charset="utf-8">  
<meta name="author" content="John Dean">  
<title>Code Fragment Figure</title>  
<style>  
  figure {  
    border: solid crimson;  
    padding: 6px;  
  }  
</style>  
</head>  
  
<body>  
<p>  
  Listing 8.3 shows JavaScript code that repeatedly generates a pop-up  
  that tells the user to provide his/her credit card information.  
</p>  
<figure>  
  <figcaption>Listing 8.3. A credit card scam that uses a repeating  
  pop-up</figcaption>  
  <pre><code>while (true)  
{  
  alert("Your financial records have been compromised.\n" +  
    " To fix the problem, go to http://www.easyCredit.com" +  
    " and enter your credit card information.");  
}</code></pre>  
</figure>  
</body>  
</html>
```

If you have a figcaption element,
it must be inside a figure element.

With the pre element, its enclosed text should be at the left.
In other words, there is no attempt to follow the usual practice
of indenting inside a block element (pre is a block element).

container instead of at the top. Consequently, in the browser window, you can see that the caption displays below the picture. The Final Tag web page's primary focus is its picture. To display a picture, you'll need to use the `img` element. We will present the `img` element formally in Chapter 6, but for now, we'll introduce just a few details to explain what's going on in the Final Tag web page. Here's the relevant source code:

```
<imgsrc="finalTag.png" alt="&lt;/life&gt; headstone">
```

Note the `img` element's `src` attribute—that's how you specify the location and name of an image file. If you don't specify a path in front of the image file's name, then the default is to look for the file in the same directory that holds the web page's `.html` file. Later, you'll see how to load a picture from a different directory, but we're keeping things simple here with the web page and the picture file in the same directory. In the preceding code fragment, note the `img` element's `alt` attribute. The HTML5 standard requires that for every `img` element, you provide an `alt` (for alternative) attribute. The `alt` attribute's value should normally be a description of the picture, and it serves two purposes. It provides *fallback content* for the image in case the image is unviewable. As you'll learn in Chapter 5, fallback content is particularly useful for visually impaired users who have *screen readers*. Screen readers can read the `alt` text aloud using synthesized speech. The preceding code fragment's `alt` value is rather odd-looking. It contains character references for the `<` and `>` symbols. When those character references are replaced with their symbols, the result looks like this:



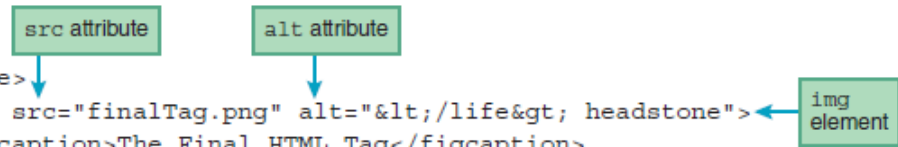
The Final HTML Tag

</life> headstone

That text provides an accurate description of the picture's content, so the alt value is appropriate.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="author" content="John Dean">
<title>Final Tag Figure</title>
<style>
  body {text-align: center;}
</style>
</head>

<body>
<figure>
  
  <figcaption>The Final HTML Tag</figcaption>
</figure>
</body>
</html>
```



Organizational Elements:

Those organizational structures are pretty straightforward because they have physical manifestations—list items in an outline and a caption above or below a figure. The rest of this chapter covers organizational elements that don't have obvious physical manifestations. Their purpose is to group web page content into sections so that you can use CSS and JavaScript to manipulate their content more effectively. Here are the organizational elements you'll be introduced to:

- ▶ ▶ section
- ▶ ▶ article
- ▶ ▶ aside
- ▶ ▶ nav
- ▶ ▶ header
- ▶ ▶ footer

There's usually no need to use these organizational elements for small web pages, but when you have a multipage website, you should try to use them consistently. For example, you should use a common header for all web pages on a particular website. Being consistent will make your web pages look uniform, and that will give your users a comfortable feeling. In addition, consistency leads to web pages that are easier to maintain and update.

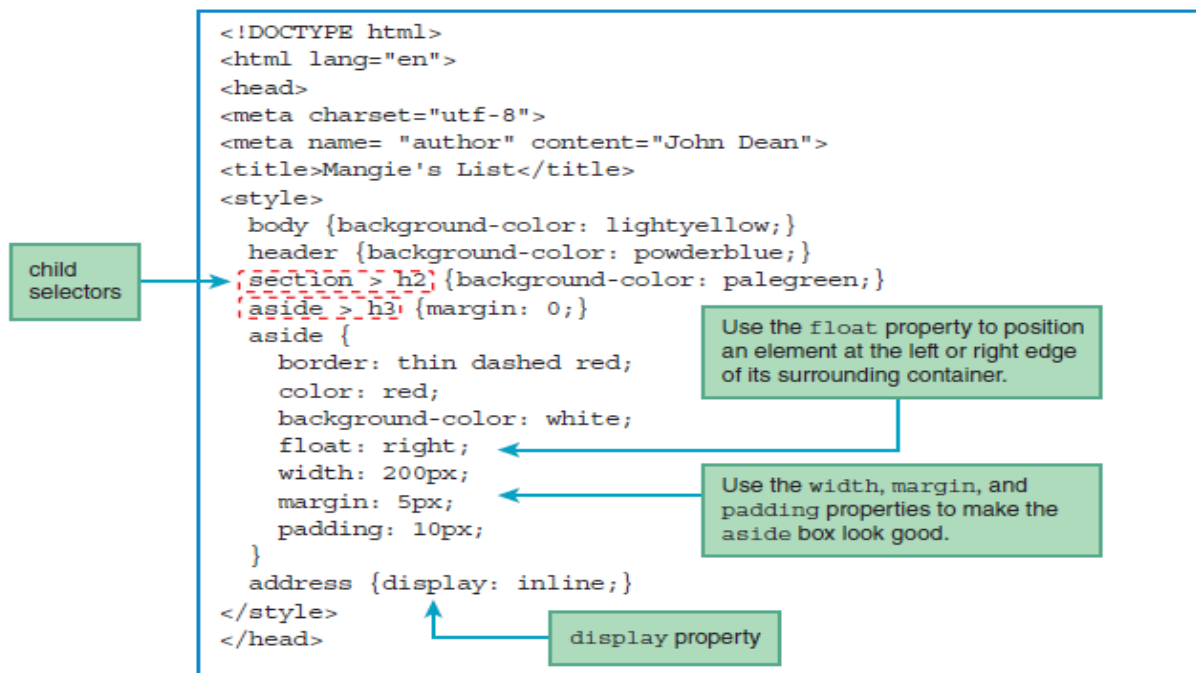
We could explain the organizational elements by showing code fragments or a series of small web pages, but that wouldn't illustrate the concepts very well. It's probably better to jump in with a complete web page where there are different areas of content that can be

compartmentalized. Take a look at FIGURE web page. It showcases Mangie's List, a tongue-in-cheek service that features reviews of dining and clothing venues from a manly man's perspective. You can see two headings at the top with a light blue background color. The headings are surrounded by a header container. Below the headings you can see two links that are surrounded by a nav (stands for navigation) container. Then comes dining content and clothing content, each with its own section container. At the right, you can see a red box, which is implemented with an aside container. Finally, at the bottom, you can see content enclosed in a footer container.

FIGURES 4.11A and 4.11B show the Mangie's List source code. We'll describe the code in

detail in the upcoming sections, but for now just peruse the callouts that show where the organizational elements are used.





Header and footer Elements:

header Element The header element is for grouping together one or more heading elements (h1-h6) such that

the group of heading elements form a unified header for nearby content. Normally, the header

is associated with a section, an article, or the entire web page. To form that association, the

header element must be positioned within its associated content container. Typically, that means

at the top of the container, but it is legal and sometimes appropriate to have it positioned lower.

To identify them because we use CSS to apply a light blue background color to the header's

content. Here's the code for the CSS rule and for the header container:

```
<style>
header {background-color: powderblue;}
...
</style>
<header>
<h1>Mangie's List</h1>
<h2><q>Simply the best reviews anywhere!</q></h2>
</header>
```

As an alternative, we could have used this CSS rule:

```
h1, h2 {background-color: powderblue;}
```

footer Element (with **address** Element Inside It)

The footer element is for grouping together information to form a footer. Typically, the footer

holds content such as copyright data, author information, or related links. The footer should be

associated with a section, an article, or the entire web page. To form that association, the

footer element must be positioned within its associated content container. Typically, that means at

the bottom of the container, but it is legal and sometimes appropriate to have it positioned elsewhere.

For the Mangie's List web page, we use a footer element for contact information. Here's the

relevant code:

```
<footer>
```

```
Questions? Email <address>mangie@gmail.com</address>.
```

```
</footer>
```

Note how the footer container has an address element inside of it. The address element is for contact information. Here, we show an e-mail address, but the address element also works for phone numbers, postal addresses, and so on. If the address element is within an article

container, then the address element supplies contact information for the article. Otherwise, the address element supplies contact information for the web page as a whole.

display Property, User Agent Style Sheets

The address element is a block element, so by default, browsers display it on a line by itself. But sometimes (actually, pretty often), you're going to want to display an address in an inline manner within a sentence. If you look at the Mangie's List web page, you can see that the address is embedded within the footer's sentence. To implement that inline behavior, the web page uses this CSS rule:

```
address {display: inline;}
```

Child Selectors

There are still a few Mangie's List CSS details that need to be covered. In this section, we tackle child selectors. In introducing child selectors, it's helpful to compare them to descendant selectors.

Remember how descendant selectors work? That's when you have two selectors separated by a space, and the browser searches for a pair of elements that match the selectors such that the second element is a descendant of the first element (i.e., the second element is contained anywhere within the first element). A *child selector* is a more refined version of a descendant selector. Instead of allowing the second element to be any descendant of the first element, the second

element must be a child of the first matched element (i.e., the second element must be within the first element, and there are no other container elements inside the first element that surround the second element).

The syntax for a child selector is the same as the syntax for a descendant selector, except that

> symbols are used instead of spaces. Here's the syntax:

```
list-of-elements-separated-with->'s {property1: value; property2: value;}
```

For an example, look at this CSS rule from the Mangie's List web page:

```
section > h2 {background-color: palegreen;}
```

CSS Inheritance:

CSS *inheritance* is when a CSS property

value flows down from a parent element to one or more of its child elements. That should

sound familiar. It parallels the inheritance of genetic characteristics (e.g., height and eye color)

from a biological parent to a child.

Some CSS properties are inheritable and some are not. To determine whether a particular

CSS property is inheritable, go to Mozilla's list of CSS keywords at <https://developer.mozilla.org>

/en-US/docs/Web/CSS/Reference and click on the property you're interested in. That should take you to a description of that property, including its inheritability. Of the CSS properties covered so far in this book, here are the ones that are inheritable:

- ▶ ▶ color
- ▶ ▶ font (and all of its more granular properties, like font-size)
- ▶ ▶ line-height
- ▶ ▶ list-style (and all of its more granular properties, like list-style-type)
- ▶ ▶ text-align
- ▶ ▶ text-transform

To explain CSS inheritance, we'll refer once again to the Mangie's List web page. Specifically, we'll refer to this aside element:

```
aside {  
border: thin dashed red;
```

```
color: red;
background-color: white;
float: right;
width: 200px;
padding: 10px;
margin: 5px;
}
```

In this rule, the only inheritable CSS property is color. If you specify papayawhip (look it up; it's real) for a body element's color, all the elements inside the body container would inherit that color. That would cause the browser to use that color when displaying all the text within the

web page. So for the Mangie's List web page, the red color gets inherited by the h3 element that is a child of the aside element.

Inheritance is blocked for an inheritable property when an element explicitly specifies a new value for that property. In other words, if a parent element and its child element have two different CSS rules with the same property specified, then the child element's property-value pair (and not the parent element's property-value pair) gets applied to the child element. Formally, we say that the child element's property-value pair *overrides* the inherited property-value pair. So, for the Mangie's List web page, if color: blue; was specified for the h3 element inside the aside element, then the browser would display blue text for the h3 element.

For the other properties shown in the preceding aside type selector rule, their values do not flow down via inheritance. Thus, inside the aside element, the h3 element does not get its own border. But it does get a background color of white, and it gets floated to the right. Why? Those properties are not inherited, but by applying those properties to the aside element, the h3 child element is naturally affected.

Bitmap Image Formats: GIF, JPEG, PNG

There are two basic categories of image files—bitmap image files and vector graphics files.

We'll have more to say about vector graphics files soon enough, but for now we'll focus on bitmap

image files. With bitmap image files, an image is comprised of a group of pixels. For example, *anicon*, which is simply a small image file, typically has 16 rows with 16 pixels in each row. Within a bitmap image file, every pixel gets mapped to a particular color value, and each color value is a sequence of bits (where a bit is a 0 or a 1). For a browser to display a bitmap image, it displays each pixel's mapped color. This reliance on mapping color bit values to pixels is the basis for the

name *bitmap image*.

The three most common formats for bitmap image files (also called raster image files) on the Web are GIF, JPEG, and PNG. You can see brief descriptions of those formats. We'll provide more details shortly, but first, you should be aware of two other file formats—BMP

(for bitmap) and TIFF (for tagged image file format). They're both very popular for graphics applications, but they're generally not used with web pages. Why? BMP files are too large, and TIFF files cannot be viewed by web browsers without a plug-in. Since we're focusing on webpages, we'll refrain from providing details about them.

Bitmap Image Formats	Description
GIF	Good for limited-color images such as line drawings, icons, and cartoon-like illustrations.
JPEG	Good for high-quality photographs.
PNG	Flexible; good for limited-color images and also high-quality photographs.

GIF Image File Format

In creating a GIF file from an original picture, the original picture's colors are mapped to an 8-bit palette of colors. That means each pixel uses 8 bits, and those 8 bits determine the pixel's color. And the entire set of colors forms the image's *color palette*. Each image has its own color palette with its own set of colors. So for the peanut butter image seen here you'd need to capture colors such as red, brown,

and blue. Each of those colors would have an 8-bit sequence associated with them, such as 01011010 for red, 11011001 for brown, and 00010110 for blue. With 8 bits for each color value, there are 256 different ways to arrange the 0's and 1's. You can prove this to yourself by writing all the different permutations of eight bits, or you can just remember that the number of permutations is equal to 2 raised to the power of the number of bits, where $2^8 = 256$. That means each GIF image file can handle a maximum of 256 distinct colors. If a

picture has more than 256 distinct colors, then when creating the GIF file from the picture, some of the colors won't be stored accurately.

Instead they'll be stored with similar colors that are part of the GIF file's color palette, and that leads to the GIF file's image being a degraded version of the original photograph. Color degradation is nonexistent or imperceptible for limited-color images such as line drawings, icons, or cartoon-like illustrations, and that's why GIF files are good for those types of things and not good for color photographs.



JPEG Image File Format

the JPEG image file format. JPEG stands for Joint Photographic Experts Group, and JPEG is pronounced "jay-peg." JPEG files use a filename extension of .jpeg or .jpg.

In creating a JPEG file from an original picture, the original picture's colors are mapped to a 24-bit palette of colors. With 24 bits, there are approximately 16 million permutations of 0's and 1's in each color value ($2^{24} = 16,777,216$). That means approximately 16 million unique colors can be represented, and that's more colors than the human eye can discern. So unless you're an eagle with the ability to distinguish between more than 16,000,000 colors, you should be in good shape with the quality of colors in JPEG files. In other words, for humans, there is effectively no information loss in 24-bit palette JPEG images. Note this picture of the New York Catskills' famed fall foliage:

Fall on Alma Pond in the Catskills Photo courtesy of Allegany County, New York.



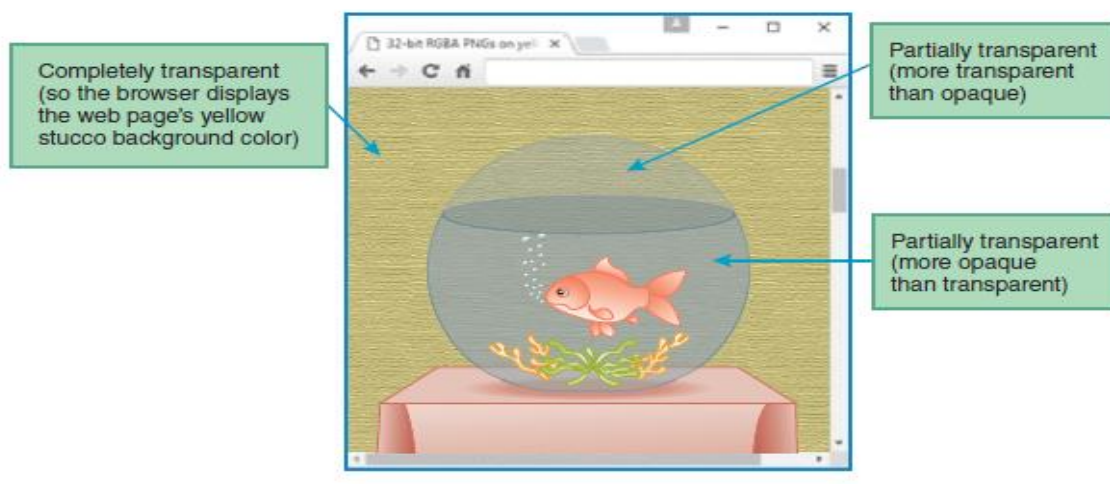
PNG Image File Format

PNG stands for Portable Network Graphics, and PNG is pronounced "ping." PNG files use a filename extension of .png.

The PNG format was invented in 1996 as an open-source alternative to the GIF format because the GIF format was copyright protected with a patent owned by Unisys. So each time someone made a new GIF file, they were supposed to pay a license fee to Unisys. Oftentimes, GIF

file creators didn't bother to pay the license fee, which was illegal. To avoid such illicit activity, the web community eventually invented their own open-source format for image files, and the PNG format was born. After the PNG format's inception in 1996, the GIF copyright expired, so it's now legal to create and use GIF files without fear of retribution. Although the PNG format no longer serves as a licit alternative to the illicit use of the GIF format, the PNG format remains very popular, as it improved upon the GIF format in several ways.

The PNG format provides more flexibility in terms of transparency. You can create images with different levels of transparency for different parts of an image. GIF images can have only two levels of transparency—completely opaque or completely transparent. PNG images can have 256 levels of transparency



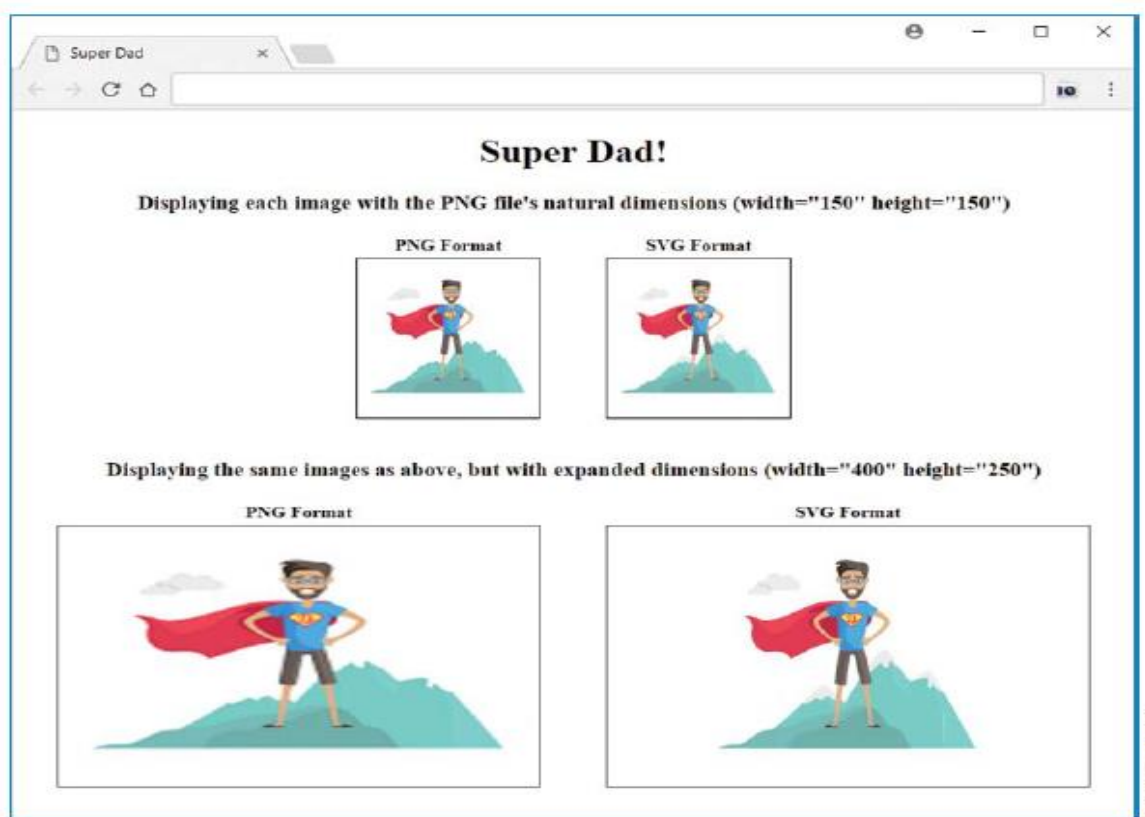
Vector Graphics:

GIF, JPEG, and PNG are formats for bitmap images. As discussed earlier, a bitmap image works with the help of a map that assigns a color to each pixel in the image's rectangular grid of pixels. If you attempt to resize a bitmap image (by using an `img` element's width and height attributes).

SVG Image Format:

SVG is the most popular type of vector graphics format, and that's the format we'll focus on. SVG stands for Scalable Vector Graphics. SVG files use a filename extension of `.svg`, but after compression, the resulting compressed file has an extension of `.svgz`.

As mentioned, one of the primary benefits of the SVG format over bitmap image formats is that there's no degradation when an SVG file is resized. To see what we're talking about, study **FIGURE** The two pictures at the top show what happens when a PNG file and an SVG file



In addition to enabling accurate resizing, the SVG format provides several other benefits over the bitmap image format. SVG files tend to be smaller, and that leads to faster web page downloads. An SVG file's

formulas are built with SVG code, and as with HTML code, you can

use JavaScript to dynamically manipulate any of the elements in the SVG file's code. you'll learn how to use JavaScript to manipulate the elements in an HTML page. At that point, you'll be ready to learn on your own how to use JavaScript to manipulate SVG files as well. With the ability to manipulate SVG files, you will then be able to animate your SVG images.

Unfortunately, the SVG format is not perfect. It does not lend itself well to accurately displaying most photographs. Most photographs have lots of different colors and lines, and it's difficult for (SVG-format) formulas to describe all of that complexity. The SVG format is not supported by older browsers, but this is becoming a non-issue, as all current and fairly recent

browsers do support the SVG format. Another drawback is that there are relatively few prebuilt SVG files to choose from (if you go to the Google Images site, you can find lots of bitmap images, but way fewer SVG images). Because it can be very difficult to find freely available SVG files that fit your needs, as a web

developer, you might want to install an SVG editor tool and create your own SVG files.

Displaying an SVG File with an `img` Element

There are several techniques for using SVG to display an image in a web page. If you already have an SVG file, the easiest way to display it is to use the standard `img` element with a `src` attribute that specifies the SVG file's name. FIGURE 6.13 shows the complete source code for the Super Dad web page. In the body container, note the second and fourth `img` elements, with their `superDad.svg` filenames.

Displaying an SVG Code Fragment with an `svg` Element

If you want to use SVG to display an image in a web page, but you don't have an SVG file, you can embed SVG code within an `svg` element. FIGURE shows a Voting Sticker web

```

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="author" content="John Dean">
<title>Super Dad</title>
<style>
  h1, h3 {text-align: center;}
  td {border: thin solid;}
  .center {display: flex; justify-content: center;}
  .gap {width: 50px; border: none;}
</style>
</head>

<body>
<h1>Super Dad!</h1>
<h3>Displaying each image with the PNG file's natural
  dimensions (width="150" height="150")</h3>
<div class="center">
  <table>
    <tr><th>PNG Format</th><th></th><th>SVG Format</th></tr>
    <tr>
      <td></td>
      <td class="gap"></td> <!-- can't adjust table cell margins -->
      <td></td>
    </tr>
  </table>
</div>
<br>
<h3>Displaying the same images as above, but with
  expanded dimensions (width="400" height="250")</h3>
<div class="center">
  <table>
    <tr><th>PNG Format</th><th></th><th>SVG Format</th></tr>
    <tr>
      <td></td>
      <td class="gap"></td> <!-- can't adjust table cell margins -->
      <td></td>
    </tr>
  </table>
</div>
</body>
</html>

```

SVG file

SVG file