



CHAPTER 5

Tables and CSS Layout

CHAPTER OBJECTIVES

- ▶ Learn the difference between data tables and layout tables.
- ▶ Use `table` elements to implement data tables.
- ▶ Use CSS to format table cells with alignment and padding.
- ▶ Partition a table with the `thead` and `tbody` elements.
- ▶ Appreciate the benefits of web accessibility, and make your tables more web accessible.
- ▶ Learn how to span cells across multiple columns or multiple rows.
- ▶ Use CSS to implement layout tables.
- ▶ Use CSS to position an element relative to its containing block.
- ▶ Use CSS to position an element relative to its normal flow.

CHAPTER OUTLINE

- 5.1 Introduction
- 5.2 Table Elements
- 5.3 Formatting a Data Table: Borders, Alignment, and Padding
- 5.4 CSS Structural Pseudo-Class Selectors
- 5.5 `thead` and `tbody` Elements
- 5.6 Cell Spanning
- 5.7 Web Accessibility
- 5.8 CSS `display` Property with Table Values
- 5.9 Absolute Positioning with CSS Position Properties
- 5.10 Relative Positioning
- 5.11 Case Study: A Downtown Store's Electrical Generation and Consumption

5.1 Introduction

In Chapter 4, we focused on different ways to organize a web page's content. We discussed organizational elements that have clear physical manifestations—list and figure elements. We also discussed several organizational elements that are less clear-cut in terms of their physical manifestations—header, footer, nav, section, article, and so on. In this chapter, we present another organizational construct, but this construct is so popular and important that it merits an entire chapter. In this chapter, we discuss tables.

At its core, a table is a group of cells organized in a two-dimensional structure with rows and columns. Normally, we think of a table's cells as holding data, but tables can also be used purely for presentation purposes. To use a table for presentation purposes, you position content at particular locations on the web page using a row-column layout scheme. An example would be putting a navigation menu at the left, putting pictures at the right, and putting contact information at the bottom. We devote about half of this chapter to tables whose purpose is to hold data. We devote most of the rest of the chapter to tables whose purpose is to provide a row-column layout scheme.

Data tables very often hold numbers, but they can hold text and other types of content as well. The following data table holds text. More specifically, it holds descriptions for the 16 personality types defined by the Myers–Briggs Type Indicator (MBTI) instrument.¹ We created the table using Microsoft Word. One of the projects at the end of this chapter asks you to implement it using HTML.

In the MBTI table, the letters (ST, IJ, etc.) at the top and at the left are headers that show how a person's preferences indicate the person's personality type. The letters represent the contrasting pairs: Introversion (I) / Extroversion (E); Judging (J) / Perceiving (P); Sensing (S) / Intuition (N); Thinking (T) / Feeling (F). So in answering the MBTI questions, if a person indicates preferences

¹ Isabel Briggs Myers, *Introduction to Type: A Guide to Understanding Your Results on the MBTI Instrument* (Mountain View, CA: CPP, Inc., 1998). The personality descriptions in Myers's book are quite a bit longer than the ones shown. We shortened the descriptions to save space.

	ST	SF	NF	NT
IJ	Quiet and serious	Quiet, friendly, and responsible	Insightful, committed to their values	Independent-minded and hard-working
IP	Tolerant and analytical	Quiet, friendly, and sensitive	Idealistic and loyal	Has a desire to develop logical explanations
EP	Flexible and pragmatic	Outgoing, friendly, and accepting	Enthusiastic and imaginative	Quick, ingenious, and stimulating
EJ	Practical and realistic	Conscientious and cooperative	Warm, empathetic, and responsive	Decisive and quick to assume leadership

for sensing, thinking, introversion, and judging, then the table would suggest that the person's personality type is quiet and serious. For more details, see <https://www.opp.com/en/tools/MBTI/MBTI-personality-Types>.

Unlike data tables, layout tables are not limited to holding data—they are allowed to hold any type of content. Their purpose is to position that content with a row-column layout scheme. Consider this graphic:



To implement this graphic as part of a web page, you would probably want to use a two-row, three-column layout table. The "SEASONS ON THE HILL" heading would be implemented as a table caption. The pictures and season names would be positioned by placing them in the layout table's six cells.

We begin this chapter by describing data tables and the HTML table elements used to implement them such as `table`, `tr`, `th`, `td`, and so on. As part of our discussion of data tables, we describe cell spanning, where adjacent cells are merged to form larger cells. Next, we discuss web accessibility techniques that make it easier for disabled users to understand data table content. We then move on to a discussion of layout tables. We first describe how to implement layout tables using CSS's `display` property with various values such as `table`, `table-caption`, `table-row`, and so on. We then describe how to implement layout tables using CSS's `position` property. We show how to use the `position` property for absolute positioning, where an element gets positioned relative to its containing block, and we show how to use the `position` property for relative positioning, where an element gets positioned relative to its normal position in the web page.

5.2 Table Elements

Let's start by looking at a simple data table—the wind disasters table in **FIGURE 5.1**. It's a data table in that it displays data, the names of famous tornadoes and hurricanes, in a row-column format. Headers are not required for a data table, but they are common, and the wind disasters table has two column headers, labeled “Tornadoes” and “Hurricanes.”

To create a data table, start with a `table` container element, fill the `table` element with a `tr` element for each of its rows, and fill each `tr` element with `th` elements for header cells and `td` elements for data cells. **FIGURE 5.2** shows the code used to implement Figure 5.1's Wind Disasters web page. Note Figure 5.2's `table` element and its four `tr` elements. The top `tr` element contains `th` elements for the column header cells. The bottom three `tr` elements contain `td` elements for the data cells.

If you'd like to display a title for a table, embed a `caption` element within the `table` container. For example, note the `caption` element in Figure 5.2 and the resulting “Wind Disasters”

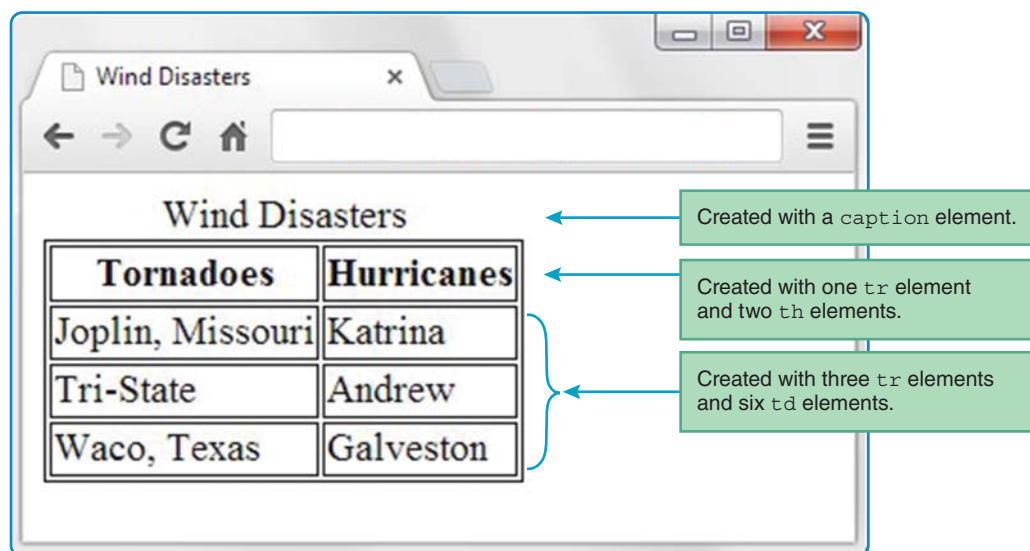


FIGURE 5.1 Wind Disasters web page

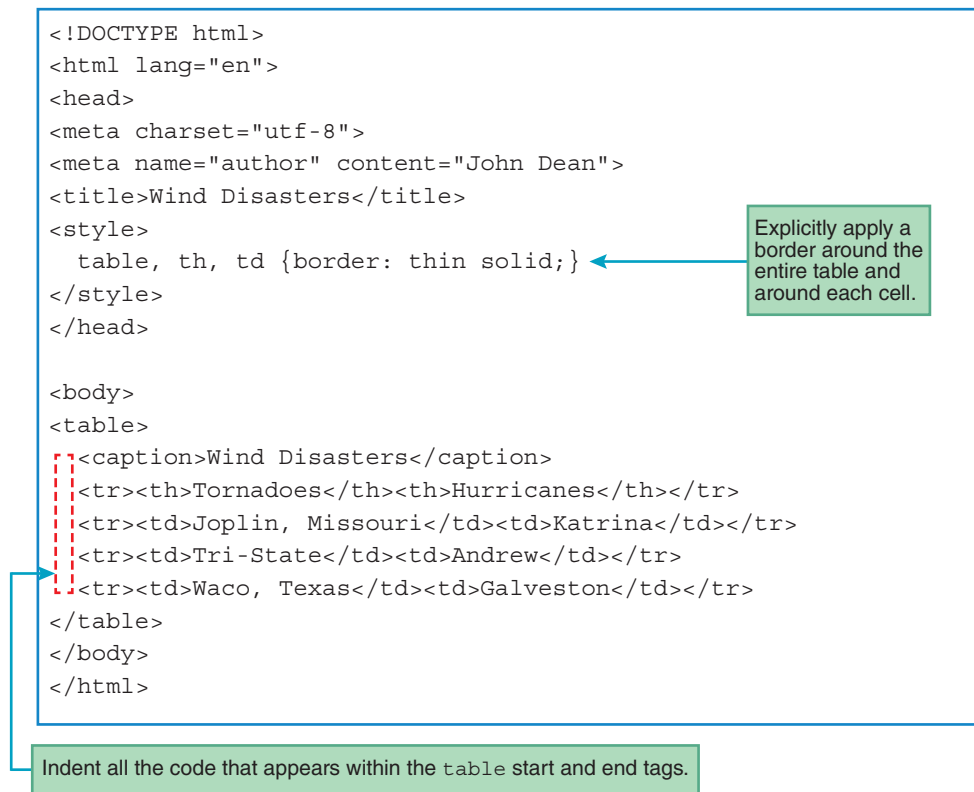


FIGURE 5.2 Source code for Wind Disasters web page

title in Figure 5.1. If you include a `caption` element within a `table` container, the `caption` element must be the first element within the table. As you'd expect, a table's caption displays above the table's grid by default. If you want the caption's text displayed at the bottom, you can use the following CSS type selector rule:

```
caption {caption-side: bottom;}
```

By default, browsers use boldface font for table header cells. You can see this behavior in Figure 5.1, where the “Tornadoes” and “Hurricanes” headers (implemented with `th` elements) are bolder than the text values below the headers. That default behavior should make sense because table headers are often boldfaced in business reports.

Beginning web programmers sometimes have trouble deciding when to use `th` elements and when to use `td` elements. Use `th` for a cell that is a description for other cells' content; use `td` for all other cells in the table.

The `table` element is a block element. As you learned earlier, unless all the code in a block element can fit on one line (and that's very unlikely with a `table` element), you should indent all of the block element's contained code. For example, in Figure 5.2, note how the `caption` and `tr` elements are indented inside the `table` element's start and end tags.

5.3 Formatting a Data Table: Borders, Alignment, and Padding

In this section, we'll describe how to format data tables in terms of their borders, cell alignment, and cell padding. Before HTML5, older versions of HTML allowed you to specify those presentation features using HTML attributes. But in sticking to its goal of keeping content and presentation separate, HTML5 has made those attributes obsolete. The solution? As usual, you should use CSS for presentation.

To specify whether or not you want borders for a table, you should use CSS's `border-style` property. To specify the border's width, you should use CSS's `border-width` property. For example, here's the CSS type selector rule used in the Wind Disasters web page:

```
table, th, td {border: thin solid;}
```

Oops. Why are there no `border-style` and `border-width` properties? That's a trick question. You might recall from Chapter 3 that `border` is a shorthand property that handles a set of border-related properties. In this example, the `border` property's first value is `thin`, which goes with the `border-width` property, and the `border` property's second value is `solid`, which goes with the `border-style` property. With `table`, `th`, and `td` all listed in the rule, the resulting web page displays a thin solid border around the entire table (except for the table's caption), around each header cell, and around each data cell. As a sanity check, glance back at Figure 5.1's Wind Disasters web page and verify that those borders exist.

Now onto the next two formatting features—cell alignment and cell padding. If you add no CSS to a table, then you'll end up using the browser's default CSS values. Table header cells (`th`) have a default alignment of center and a default weight of bold. Table data cells (`td`) have a default alignment of left. Both `th` and `td` cells have a default padding of none. Look at the Wind Disasters web page in Figure 5.1 and verify that the table uses those default CSS values. To adjust the horizontal alignment of text in table cells, use CSS's `text-align` property with a value of `left`, `right`, or `center`. To adjust the padding around the text in table cells, use CSS's `padding` property with a pixel value (e.g., `5px`).

Look at the Wind Disasters web page in **FIGURE 5.3**. Note how the header text ("Tornadoes" and "Hurricanes") is left aligned. Note how there's padding around every cell's text. Also note the border widths—the outer border is thicker than the cell borders, and there's an even thicker border below the header cells. To hone your problem-solving skills, see if you can figure out what CSS rules need to be added in order to implement those formatting features. Please do not continue reading until you try.

Have you got the CSS figured out? To add left alignment and padding to every cell, use this type selector rule:

```
th, td {  
    text-align: left;  
    padding: 10px;  
}
```

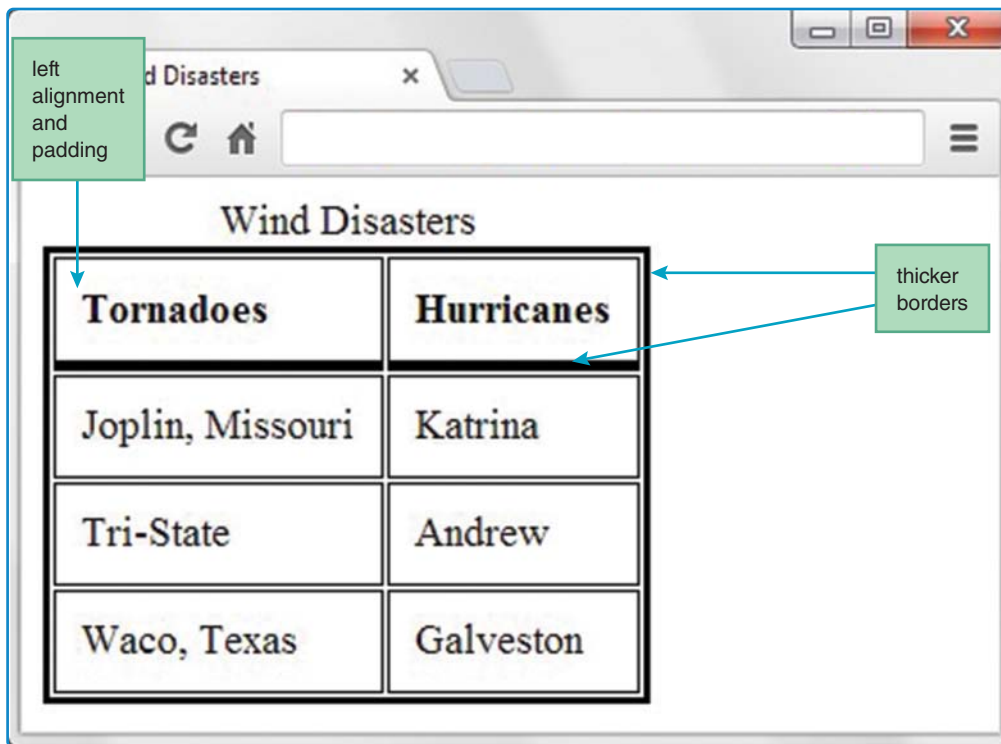


FIGURE 5.3 Wind Disasters web page with improved formatting

This rule specifies left alignment for the `td` element, even though the `td` element uses left alignment by default. That's OK. The code is *self-documenting*—it explicitly shows the web developer's intent without needing a comment.

Figure 5.2's style container contains the following CSS rule, which creates thin border lines around the entire table and also around each cell:

```
table, th, td {border: thin solid;}
```

To assign a medium width to the table's outer border and to assign a thick width to the header cells' bottom borders, add the following type selector rules below the rule:

```
table {border-width: medium;}
th {border-bottom-width: thick;}
```

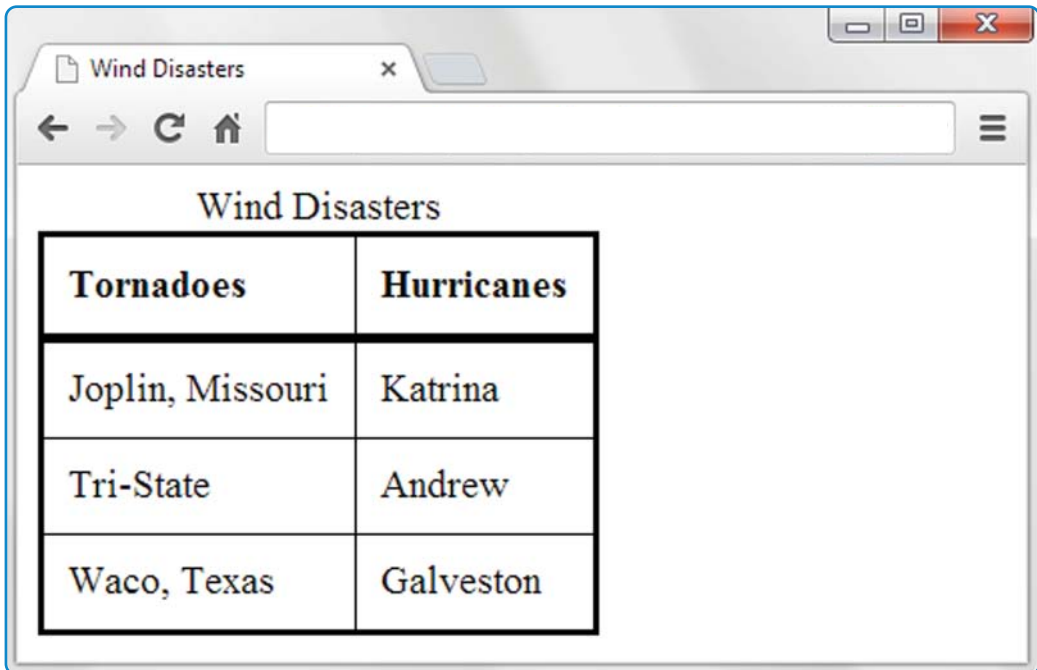
Note that all three rules deal with borders. The first and second rules both provide values for the table element's `border-width` property. The second rule's `border-width` property is explicit (`border-width: medium`), whereas the first rule's `border-width` property is built into the border shorthand property (`border: thin`). So will the table's border width be medium or thin? If two CSS rules refer to the same property, the rule that appears later overrides the prior rule's property value. Therefore, because the `border-width: medium` rule appears later, it wins and the table's border width will be medium.

The first and third rules provide values for the `th` element's `border-bottom-width` property. The third rule's `border-bottom-width` property is explicit (`border-bottom-width: thick`), whereas the first rule's `border-bottom-width` property is built into the `border` shorthand property (`border: thin`). Because the `border-bottom-width: thick` rule appears later, it wins and the `th` elements' bottom border widths will be `thick`.

In Figure 5.3, note how there's a gap between each of the borders. More specifically, there's a gap between the table's exterior border and the individual cells' borders, and there's a gap between the borders for adjacent cells. If you'd like to eliminate those gaps and merge the borders, use the `border-collapse` CSS property with a value of `collapse`, like this:

```
table {border-collapse: collapse;}
```

The default value for `border-collapse` is `separate`, and the `separate` value causes gaps to appear between adjacent borders. On the other hand, if you add the CSS rule to the “improved formatting” Wind Disasters web page shown in Figure 5.3, here's the result:



5.4 CSS Structural Pseudo-Class Selectors

The previous chapter described lists, and this chapter describes tables and tabular formatting. All of these things involve collections of elements. When you have a collection of elements, sometimes you want to display one or more of those elements differently from the rest. You could do that by including `class` attributes in each element that you want to display differently. You would then use the `class` attribute's value as a class selector in a CSS rule. But if the number of elements that you want to display with a special format is large, then quite a few `class="value"` code insertions would be required.

When there is regularity in the locations of certain elements within a collection of elements, you can avoid the `class="value"` code insertions described and, instead, implement that functionality with a structural pseudo-class CSS rule. *Pseudo-classes* conditionally select elements from a group of elements specified by a standard selector. For example, the following code uses a standard `tr` type selector to select all the `tr` elements in a web page, and the `:first-of-type` pseudo-class checks each of those elements to see if it is a first `tr` element within a particular table:

```
tr:first-of-type {background-color: palegreen;}
```

For each conditionally selected `tr` element (i.e., for each first-row `tr` element), the browser displays the element with a pale green background color.

A pseudo-class is called a “pseudo-class” because using a pseudo-class is similar to using a `class` attribute, but the two entities are not identical. A pseudo-class is like a class selector in that it matches particular instances of elements (that’s what happens with elements that use `class` attributes). But they are different from class selectors in that they don’t rely on the `class` attribute.

When we describe pseudo-class selectors, we’ll very often use the terms *sibling* and *parent*. For the preceding CSS rule, we formally say that the pseudo-class checks for a first `tr` element from among a group of sibling `tr` elements. *Sibling* elements are elements that have the same parent element. An element is considered to be a parent of another element if it contains the other element with just one level of nesting. This mimics the notion of a human parent. A human is a parent of its children, but is not a parent of its grandchildren.

The W3C defines 12 structural pseudo-classes, but we’ll focus on three of the most popular ones; they are `:first-of-type`, `:last-of-type`, and `:nth-of-type()`. See **FIGURE 5.4** for a short description of each one. As indicated in the figure, all pseudo-classes start with a colon. The purpose of a pseudo-class is to qualify a standard selector. More specifically, it provides a condition for selecting an element(s) from among the elements selected by a standard selector.

We’ve already discussed `:first-of-type`. Now let’s discuss its partner, `:last-of-type`. As you might guess, the `:last-of-type` pseudo-class checks each of the elements selected by a standard selector to see if the element is a last element from among a group of sibling elements. So the following example selects `li` elements that are at the bottom of unordered lists:

```
ul > li:last-of-type {background-color: palegreen;}
```

Pseudo-Class	Description
<code>:first-of-type</code>	Selects first element in sibling group of a particular type.
<code>:last-of-type</code>	Selects last element in sibling group of a particular type.
<code>:nth-of-type()</code>	Uses parentheses value to select an element or group of elements.

FIGURE 5.4 Popular structural pseudo-class selectors

Note the `ul > li` child selector notation, which means that an `li` element is selected only if it is a child of a `ul` element. Note that there are no spaces on either side of the pseudo-class's colon. That's a style rule. The rationale is that having no spaces serves as a visual reminder that the `last-of-type` pseudo-class qualifies the `li` selector. In the CSS rule, it would be legal to qualify `ul` with its own pseudo-class. If you did so, there should be no space between `ul` and the newly added pseudo-class.

Now for the more challenging pseudo-class, which is `:nth-of-type()`. Unlike the other pseudo-classes so far, the `:nth-of-type()` pseudo-class has parentheses. Inside the parentheses, you provide a value that indicates which element or group of elements you want to select. For example, in the following CSS rule, we put 3 in the parentheses to select the third data cell within a row of sibling data cells:

```
td:nth-of-type(3) {text-align: right;}
```

That rule matches every third `td` element within each row of `td` elements and causes those `td` elements to be right-aligned. Effectively, that causes tables to display their third columns with right-aligned data (which works nicely for money values). As an alternative to putting a number in the parentheses, you can specify `even` or `odd`. For example, in the following CSS rule, we put `even` in the parentheses to select every even-numbered row:

```
tr:nth-of-type(even) {background-color: lightblue;}
```

That rule causes tables to display their even-numbered rows (second, fourth, and so on) with a light blue background color.

As an alternative to putting a number, `even`, or `odd` in the parentheses, you can use an expression of the form $an + b$, where a and b are constant integers and n is a variable named n . By using such an expression, you can specify interleaved groups of elements. This is better explained with an example:

```
tr:nth-of-type(5n+2) {background-color: red;}
```

That rule selects every fifth `tr` element starting with the second row. In other words, it selects rows 2, 7, 12, 17, and so on. The way it works is you plug in values for n by starting with n equals 0 and incrementing n by 1 each time. So when n equals 0, row 2 is selected. When n equals 1, row 7 is selected. Make sense?

FIGURE 5.5 shows code for a simple example that presents electrical power generated by a small store's rooftop photovoltaic solar collectors, plus that store's immediate electrical consumption and electrical and thermal storage. This example includes three structural pseudo-class rules. The first uses `nth-of-type` selectors to right-align data in the third and fourth columns of the table. The second uses `:first-of-type` to color the background of the table's header row pale green. The third uses `:nth-of-type(2n+3)` to color alternate data rows pale goldenrod.

FIGURE 5.6 shows what this code displays. The optional case study at the end of this chapter expands a variation of this example and includes other material before and after this table.

```

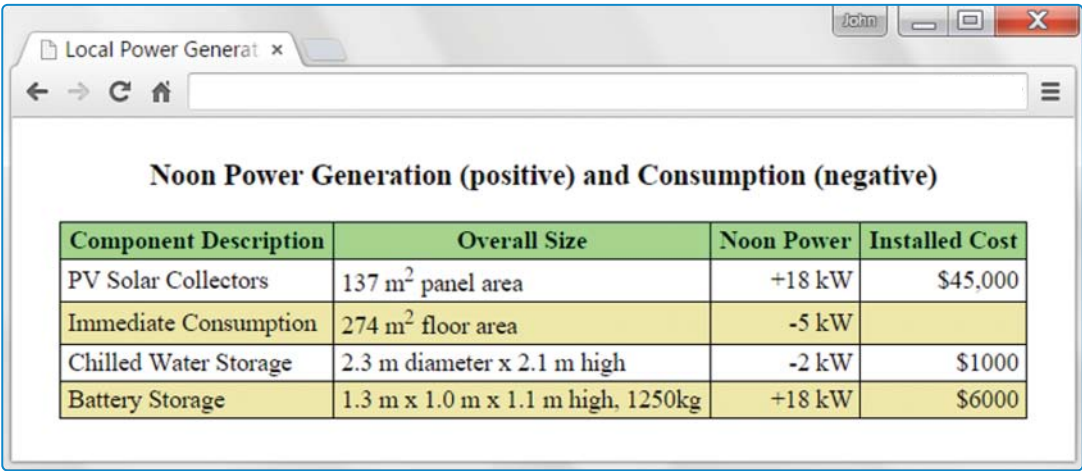
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="author" content="John Dean">
<title>Local Power Generation and Consumption</title>
<style>
  h3 {text-align: center;}
  table {border-collapse: collapse; margin: 0 auto;}
  th, td {border: thin solid; padding: 2px 5px;}
  td:nth-of-type(3), td:nth-of-type(4) {text-align: right;}
  tr:first-of-type {background-color: palegreen;}
  tr:nth-of-type(2n+3) {background-color: palegoldenrod;}
</style>
</head>

<body>
<table>
  <caption>
    <h3>Noon Power Generation (positive) and Consumption (negative)</h3>
  </caption>
  <tr>
    <th>Component Description</th> <th>Overall Size</th>
    <th>Noon Power</th> <th>Installed Cost</th>
  </tr>
  <tr>
    <td>PV Solar Collectors</td> <td>137 m<sup>2</sup> panel area</td>
    <td>+18 kW</td> <td>$45,000</td>
  </tr>
  <tr>
    <td>Immediate Consumption</td> <td>274 m<sup>2</sup> floor area</td>
    <td>-5 kW</td><td> </td>
  </tr>
  <tr>
    <td>Chilled Water Storage</td> <td>2.3 m diameter x 2.1 m high</td>
    <td>-2 kW</td> <td>$1000</td>
  </tr>
  <tr>
    <td>Battery Storage</td> <td>1.3 m x 1.0 m x 1.1 m high, 1250kg</td>
    <td>+18 kW</td> <td>$6000</td>
  </tr>
</table>
</body>
</html>

```

These rules use structural pseudo-class selectors.

FIGURE 5.5 Source code for Power Table web page



Component Description	Overall Size	Noon Power	Installed Cost
PV Solar Collectors	137 m ² panel area	+18 kW	\$45,000
Immediate Consumption	274 m ² floor area	-5 kW	
Chilled Water Storage	2.3 m diameter x 2.1 m high	-2 kW	\$1000
Battery Storage	1.3 m x 1.0 m x 1.1 m high, 1250kg	+18 kW	\$6000

FIGURE 5.6 Power Table web page

5.5 `thead` and `tbody` Elements

Normally, you'll put table header cells at the top of a table's columns, but sometimes you'll also want to put them at the left of each row. For example, in the Global Temperatures web page in [FIGURE 5.7](#), note the year values in header cells at the left. If you have header cells at the left, very often you'll want to differentiate those header cells from the ones at the top. The preferred way to differentiate is to put the top cells' row (or rows) in a `thead` element and put the subsequent rows in a `tbody` element. In the Global Temperatures web page, why would you need to differentiate between the header cells at the left and the ones at the top? So you can apply different CSS background-color rules to the different groups of header cells—midnight blue for the top cells and violet red for the left-side cells.

Take a look at the Global Temperatures `thead` and `tbody` code in [FIGURE 5.8](#). The `thead` element contains a `tr` element and three `th` elements within the `tr` element. The `tbody` element contains several `tr` elements, with each `tr` element holding a `th` element and two `td` elements. Here are simplified versions of the descendant selector rules used to color `thead`'s header cells differently from `tbody`'s header cells:

```
thead th {background-color: midnightblue;}
tbody th {background-color: mediumvioletred;}
```

In [Figure 5.8](#), note the indentations for the `thead`, `tbody`, and `tr` containers. They are all block elements, and unless all the code in a block element can fit on one line, you should indent all of the block element's contained code. For example, note how the `tr` elements are indented inside the `thead` and `tbody` containers. Also, inside the first `tr` container, note how the three `th` elements are indented even more so.

Besides using `thead` and `tbody`, there are other ways to distinguish the top header cells from the left-side header cells. For example, you could use `class` attributes with one value for

Average Annual Global Temperatures

Year	Temp Rank	Avg Temp (°F)
2016	1	58.98
2015	2	58.77
2014	3	58.53
2013	5	58.37
2012	9	58.33

thead holds the first row

tbody holds the subsequent rows

FIGURE 5.7 Global Temperatures web page

Note: The web page's table data refers to global land-ocean surface air temperatures. Data from the National Oceanic and Atmospheric Administration (NOAA), "Global Climate Report—Annual 2016," National Oceanic and Atmospheric Administration (NOAA), January 2017, <https://www.ncdc.noaa.gov/sotc/global/201613>.

the top header cells and a different value for the left-side header cells. However, that would lead to cluttered code—a `class` attribute for every `th` cell. On the other hand, the Global Temperatures web page uses `thead` and `tbody`, which means less clutter because no `class` attributes are necessary.

Let's now examine a few noteworthy CSS rules from the Global Temperatures web page that are unrelated to `thead` and `tbody`. Here's the first such rule:

```
body {display: flex; justify-content: center;}
```

That rule tells the browser to center all the elements in the `body` container horizontally within the browser window's borders. Go back to Figure 5.8 and confirm that the `body` container has only one child element—the `table` element. So the `table` gets centered. Go back to Figure 5.7 and confirm that the table is indeed centered. In the CSS rule, the `display: flex;` property-value pair creates a *flexbox* layout (also called a *flexible box* layout). It provides the ability to add certain formatting features to a standard block element. The formatting feature we're interested in now is horizontal centering, and the `justify-content: center` property-value pair takes care of that.

```

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="author" content="John Dean">
<title>Global Temperatures</title>
<style>
  body {display: flex; justify-content: center;}
  table, th, td {border: none;}
  th, td {padding: 10px;}
  thead th {
    background-color: midnightblue;
    color: white;
    vertical-align: bottom;
  }
  tbody th {
    background-color: mediumvioletred;
    color: white;
  }
  td {
    background-color: mistyrose;
    text-align: center;
  }
</style>
</head>

<body>
<table>
  <caption>Average Annual Global Temperatures</caption>
  <thead>
    <tr>
      <th>Year</th>
      <th>Temp<br>Rank</th>
      <th>Avg<br>Temp (&deg;F)</th>
    </tr>
  </thead>
  <tbody>
    <tr><th>2016</th><td>1</td><td>58.98</td></tr>
    <tr><th>2015</th><td>2</td><td>58.77</td></tr>
    <tr><th>2014</th><td>3</td><td>58.53</td></tr>
    <tr><th>2013</th><td>5</td><td>58.37</td></tr>
    <tr><th>2012</th><td>9</td><td>58.33</td></tr>
  </tbody>
</table>
</body>
</html>

```

To center a block element (like `table`), apply this CSS code to the element's parent container.

To position text vertically within its container, use the `vertical-align` property.

If a row's content is too long to fit on one line, then put indented cell elements on separate lines.

FIGURE 5.8 Source code for Global Temperatures web page

In the CSS rule, note that the selector is `body`, not `table`. To center an element, you apply the rule to the element's parent container, not to the element itself.

Because the flexbox layout is fairly new (the W3C introduced it to its CSS specification in 2016), older browsers don't support it. Therefore, you should be familiar with this alternative technique, which is pervasive throughout the web page universe:

```
table {margin: 0 auto;}
```

The code shows two values for the `margin` property—`0` and `auto`. You might recall from Chapter 3 that if you provide two values, the first value specifies the top and bottom margins and the second value specifies the left and right margins. So with the first value being `0` in the CSS rule, there are no margins above and below the table. The `auto` value requires some additional explanation. For any block element (including a `table` element), if the left margin and right margin are both set to `auto`, that will force the browser to make the margins equal, which forces the browser to center the block element. Thus, the CSS rule causes the table to be centered.

Here's a simplified version of another rule from the Global Temperatures web page that deserves some attention:

```
thead th {vertical-align: bottom;}
```

What is the `vertical-align` property for? Before answering that question, look back at Figure 5.7's Global Temperatures web page. Note how the top heading values are aligned at the bottom of their cells. Using a `bottom` value for the `vertical-align` property causes a cell's text to be aligned at the bottom. If you need top or middle vertical alignment, use the `vertical-align` property with a value of `top` or `middle`, respectively.

Note this additional CSS rule from the Global Temperatures web page, copied here for your convenience:

```
table, th, td {border: none;}
```

The `border: none` property-value pair means that the browser will not draw border lines. That means the web page's background color appears where the border lines would normally appear. For the Global Temperatures web page, the cells use a different background color than the web page's background. So with `border none`, the borders display as white lines, from the web page's default white background color. Although the default is for a table to have no visible borders, it's fairly unusual to stick with that default. Thus, we include the rule to make it clear to someone looking at the code that having no borders is intentional. This is a form of self-documentation.

There is one final noteworthy item in the code for the Global Temperatures web page. The table's first row contains this `th` container code:

```
<th>Avg<br>Temp (&deg;F)</th>
```

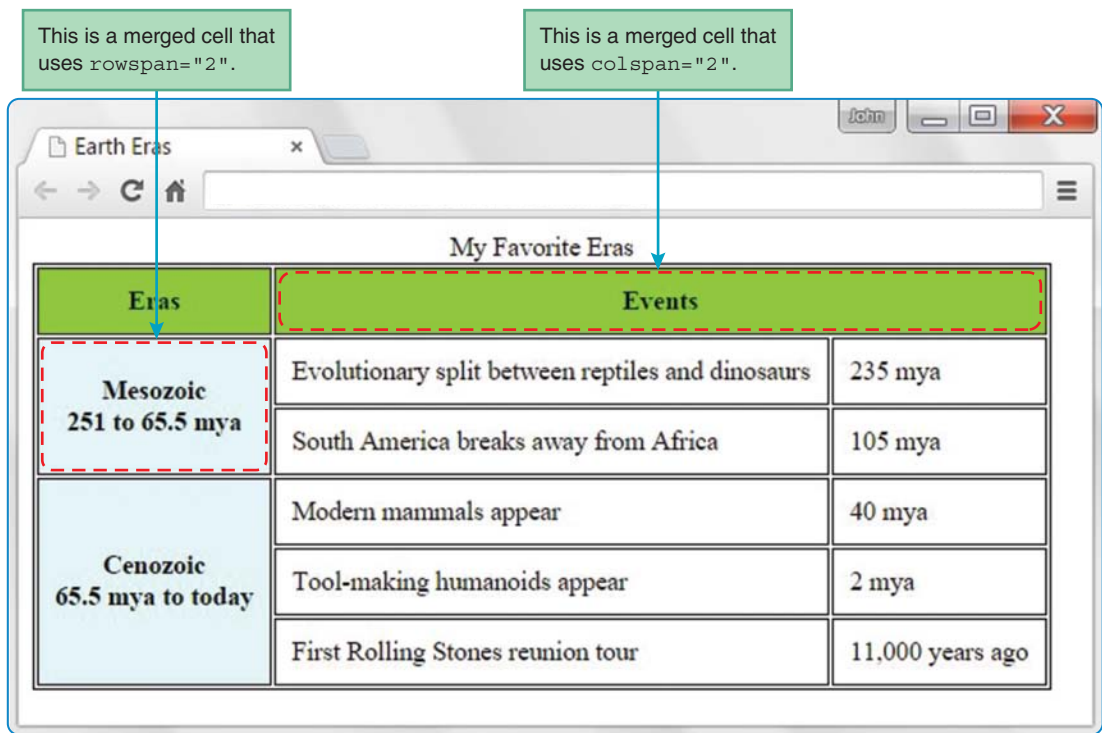


FIGURE 5.9 My Favorite Eras web page

What is `°`? It’s a character reference for the degree character (°). To verify, see the “Avg Temp (°F)” header in Figure 5.7.

5.6 Cell Spanning

So far, all of our data table examples have used a standard grid pattern, with one cell for each row-column intersection. But sometimes data tables will have cells that span more than one of the intersections in a standard grid. For example, see the My Favorite Eras table in [FIGURE 5.9](#). We implemented it using a table element with six rows and three columns. The Events cell at the top is a merged version of two cells in the first row. The Mesozoic cell at the left is a merged version of two cells in the first column. Below the Mesozoic cell, the Cenozoic cell is a merged version of the next three cells in the first column.

If you want to create a merged cell that spans more than one column, you’ll need to add a `colspan` attribute to a `th` or `td` element. [FIGURE 5.10](#) shows the code for the My Favorite Eras web page. In particular, examine the code for the table’s first row, and note `colspan="2"`, which creates a merged cell that spans two columns. We’ve copied the code here for your convenience:

```
<tr><th>Eras</th><th colspan="2">Events</th></tr>
```



```

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="author" content="John Dean">
<title>Earth Eras</title>
<style>
  table {border: thin solid;}
  th, td {border: thin solid; padding: 10px;}
  thead th {background-color: lawngreen;}
  tbody th {background-color: lightcyan;}
</style>
</head>

<body>
<table>
  <caption>My Favorite Eras</caption>
  <thead>
    <tr><th>Eras</th><th colspan="2">Events</th></tr>
  </thead>
  <tbody>
    <tr>
      <th rowspan="2">Mesozoic<br>251 to 65.5 mya</th>
      <td>Evolutionary split between reptiles and dinosaurs</td>
      <td>235 mya</td>
    </tr>
    <tr>
      <td>South America breaks away from Africa</td>
      <td>105 mya</td>
    </tr>
    <tr>
      <th rowspan="3">Cenozoic<br>65.5 mya to today</th>
      <td>Modern mammals appear</td>
      <td>40 mya</td>
    </tr>
    <tr><td>Tool-making humanoids appear</td><td>2 mya</td></tr>
    <tr>
      <td>First Rolling Stones reunion tour</td>
      <td>11,000 years ago</td>
    </tr>
  </tbody>
</table>
</body>
</html>

```

FIGURE 5.10 Source code for My Favorite Eras web page

Note how this row has two cells—an Eras header cell and an Events header cell. On the other hand, the table's next row has three cells—a Mesozoic header cell and two data cells:

```
<tr>
  <th rowspan="2">Mesozoic<br>251 to 65.5 mya</th>
  <td>Evolutionary split between reptiles and dinosaurs</td>
  <td>235 mya</td>
</tr>
```

The different number of cells in the two rows should make sense when you realize that the first row's second cell is formed by spanning two cells in the table's original grid pattern.

In this code fragment, note `rowspan="2"` in the first cell. If you want to create a merged cell that spans more than one row, add a `rowspan` attribute to the cell's `th` or `td` element. Thus, as shown in Figure 5.9, the Mesozoic header cell spans two rows.

Here's the code for the table's next row:

```
<tr>
  <td>South America breaks away from Africa</td>
  <td>105 mya</td>
</tr>
```

Remember that the prior row has three cells. Can you figure out why this row has only two cells rather than three? It's because the prior row's first cell (the Mesozoic header cell) spans down and replaces the next row's first cell.

Refer back to Figure 5.9's My Favorite Eras web page. In the light blue header cells at the left, note how each era's name appears on a separate line from its range of years. We induced that separation by adding `
` elements after the words "Mesozoic" and "Cenozoic." If there were no `
` elements, then each era's name and range of years would appear on one line, unbroken, and the left-side column would widen to accommodate the longer lines of text. Having such a wide left-side column would look rather odd because of all the unused space within the Mesozoic and Cenozoic header cells. An important takeaway from this is that a table column will conform to the width of the column cell with the widest content. An exception to that rule occurs when a table's natural width is greater than the browser window's width. In that case, the browser will shrink one or more of the table's columns so that the entire table displays in the browser window. In shrinking a column(s), the browser will initiate line wrap in the cell(s) with the widest content. To get a better appreciation for this phenomenon, you should experience it for yourself. Load the My Favorite Eras web page in a browser. Use your mouse to decrease the browser's width, and as you do so, in the first data cell, you should see "dinosaurs" wrap to a second line. The browser chooses to wrap that data cell's text because that data cell contains the widest content from among all the cells in the second column. The rules that determine which columns to shrink first are slightly different for the different browsers. The rules can get pretty complicated, but don't worry—there's no need to understand them fully just yet.

Having different rules about which columns to shrink first can lead to slightly different layouts for users with different browsers. That inconsistency might run counter to your tendency to want to make everything look identical on all browsers. Consistency is indeed a worthy goal. Nonetheless, sometimes attempting to achieve that goal is not worth the effort.

5.7 Web Accessibility

In this section, we'll digress a bit and discuss *web accessibility*—a subject that is important for tables specifically, but also for programming in general. Web accessibility means that disabled users can use the Web effectively. Most web accessibility efforts go toward helping users with visual disabilities, but web accessibility also attempts to address the needs of users with hearing, cognitive, and motor skills disabilities.

Many countries have laws that regulate accessibility for websites. For example, <https://www.access-board.gov/guidelines-and-standards/communications-and-it> describes web accessibility guidelines that U.S. government agencies are required to follow.

To promote the social good (through equal opportunities for disabled people) and to promote their own businesses, many companies have policies that require web developers and software purchasers to follow web accessibility standards. Such policies can promote the company's business not only by attracting people who fall into the traditional disabled categories, but also by providing added value to other people. For example, accessible websites tend to be better for users with slow Internet connections and for users who need glasses.

Typically, visually impaired users have screen readers to read web pages. A *screen reader* is software that figures out what the user's screen is displaying and sends a text description of it to a speech synthesizer. The speech synthesizer then reads the text aloud.

The easiest way to understand a table is to look at it. If you can't see the table and you rely on someone else reading the table's content to you, you'll probably have a harder time understanding what's going on. Likewise, because their output is purely auditory and not visual, screen readers are a bit challenged when it comes to describing a table's content. To overcome that challenge, screen readers rely on the fact that most data tables have header cells in the first row or the first column. When screen readers see such "simple" data tables, they assume that each header in the first row describes the data cells that are below it. Likewise, if there are headers in the first column, screen readers assume that each of those headers describes the data cells that are at the right of the header.

If you have a data table in which one or more header cells are not in the first row or column (i.e., it's not a simple table), then you should consider adding code to make the table more web accessible. In particular, you should consider embedding a `details` element in the table's `caption` element. The `details` element provides a description of the table's content so that a screen reader can read the description and get a better understanding of the nature of the table's organization.

The Grading Weights table in **FIGURE 5.11** has header cells in its second row, so the table is a good candidate for a web accessibility makeover.² In the figure, note the right-facing triangle under the table's title. If the user clicks the triangle, the browser will display "help" details that describe the table's content. The triangle and the text that describes the table both come from a `details` element embedded in the table's `caption` element. Take a look at **FIGURE 5.12A** and find the `details` element and its enclosed text. Screen readers will use that text to describe the table's organization. The HTML5 standard requires that you preface the `details` element's text

²The Grading Weights web page is excerpted from the #1 hit television series "Extreme Makeover: Web Page Accessibility Edition."

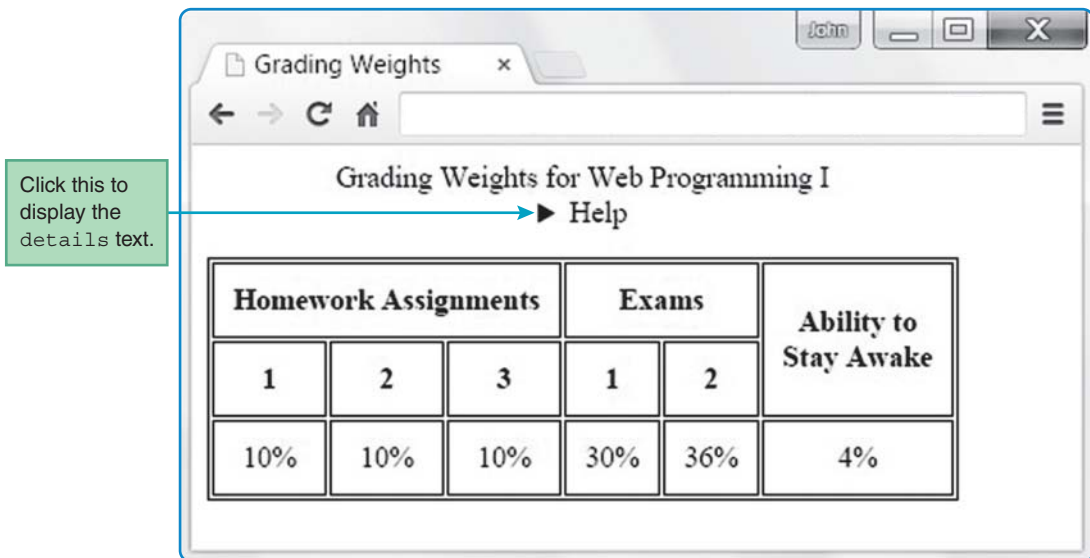


FIGURE 5.11 Grading Weights web page

with a summary element. In Figure 5.12A, the `summary` element contains one word—“Help.” In Figure 5.11, you can see that “Help” serves as a label for the clickable details triangle.

The `details` element is new with HTML5, and, as such, older browsers don’t support it. Besides the `details` element, another way to help screen readers understand complicated data tables (where one or more header cells are not in the first row or first column) is to use the `headers` attribute. With a `headers` attribute, you specify the header cell(s) that each data cell or subheader cell is associated with. For example, in the Grading Weights table, the 36% data cell is associated with the Exams header cell and also the 2 subheader cell immediately above the 36% cell. So to help with web accessibility, the 36% cell has a `headers` attribute that specifies those two header cells. Here’s the relevant code from **FIGURE 5.12B**:

```
<td headers="exams exam2">36%</td>
```

In this `headers` attribute, the “exams” and “exam2” values match the `id` values for the Exams header cell and the 2 subheader cell immediately above the 36% cell. Here’s the relevant code from Figure 5.12B:

```
<th colspan="2" id="exams">Exams</th>
...
<th id="exam2" headers="exams">2</th>
```

In this `headers` attribute, the “exams” value matches the `id` value for the Exams header cell. This stuff can be kind of tricky, so spend time studying the Grading Weights web page and source code until you’re comfortable with it.

The `details` element and `headers` attribute make things easier for screen readers by indicating how the cells are organized by rows and columns and how the data cells relate to the header

```

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="author" content="John Dean">
<title>Grading Weights</title>
<style>
  table, th, td {border: thin solid;}
  th, td {
    text-align: center;
    padding: 10px;
  }
  caption {margin-bottom: 15px;}
</style>
</head>
<body>
  <table>
    <caption>
      Grading Weights for Web Programming I
      <details>
        <summary>Help</summary>
        The first 3 columns show weights for the 3 homework assignments.
        The next 2 columns show weights for the 2 exams.
        The last column shows the weight for staying awake during class.
      </details>
    </caption>
  </table>

```

details element, embedded in a caption element

summary element, embedded in the details element

FIGURE 5.12A Source code for Grading Weights web page

cells. But what if a `table` element is used for layout purposes, and the cells are not organized by rows and columns? If a screen reader reads such a table and doesn't know that it's a layout table, there's a good chance that it will provide unhelpful (and possibly confusing) information. For example, before verbalizing each row's content, it might preface the content with a row number, even though the user won't care about row numbers. To avoid this problem, if you have a table element being used for layout purposes, you should add a `role` attribute to the `table` element's start tag, like this:

```
<table role="presentation">
```

That tells the screen reader that the table is for presentation/layout purposes, not for storing data, and the screen reader should then be able to do a better job describing the table's content.

Even though `role="presentation"` is part of the HTML5 specification, the W3C is not particularly fond of it. The W3C states that you should implement layout with CSS and not

```

<tr>
  <th colspan="3" id="hw">Homework Assignments</th>
  <th colspan="2" id="exams">Exams</th>
  <th rowspan="2" id="awake">Ability to<br>Stay Awake</th>
</tr>
<tr>
  <th id="hw1" headers="hw">1</th>
  <th id="hw2" headers="hw">2</th>
  <th id="hw3" headers="hw">3</th>
  <th id="exam1" headers="exams">1</th>
  <th id="exam2" headers="exams">2</th>
</tr>
<tr>
  <td headers="hw hw1">10%</td>
  <td headers="hw hw2">10%</td>
  <td headers="hw hw3">10%</td>
  <td headers="exams exam1">30%</td>
  <td headers="exams exam2">36%</td>
  <td headers="awake">4%</td>
</tr>
</table>
</body>
</html>

```

FIGURE 5.12B Source code for Grading Weights web page

with the `table` element. That wasn't always the case. In older versions of HTML, the `table` element was used not only for holding data, but also for layout. Consequently, many web developers have gotten used to using the `table` element for layout purposes. Knowing that it's hard to "teach an old dog new tricks," the W3C realizes that this nonconforming practice will continue for the foreseeable future. The `role="presentation"` solution somewhat mitigates the problem.

This section provided a brief introduction to the rather large field of web accessibility. If you'd like additional details, check out the W3C's accessibility page at <https://www.w3.org/WAI/intro/accessibility.php>.

5.8 CSS `display` Property with Table Values

In the previous section, we told you not to use the `table` element for layout tables, but if you do so, you should use `role="presentation"` to avoid incurring the wrath of the W3C police. Now it's time to discuss how to implement layout tables the right way—using CSS rather than the `table` element. There are two main ways to implement layout tables with CSS. If you want the layout boundaries to grow and shrink the way they do for an HTML `table` element, then use the CSS `display` property with table values. On the other hand, if you want the layout boundaries to be fixed (no growing or shrinking), then use CSS position properties. In this section, we discuss the first technique, using the CSS `display` property with table values, and in the next section, we discuss the second technique, using CSS position properties.

The display Property's Table Values

In Chapter 4, you learned about the CSS `display` property. Specifically, you used a `display: inline` property-value pair to display an `address` element (which is normally a block element) in the flow of its surrounding sentence. The `display` property can be used for much more than just inlining block element content. It can also be used to emulate the various parts of a table. Review **FIGURE 5.13**. It shows values for the `display` property that enable elements to behave like the parts of a table. Figure 5.13's first `display` property value is `table`. The `table` value enables an element, like a `div` element, to behave like a table. Here's how you can do that:

```
<style>
  .table {display: table;}
  ...
</style>

<body>
<div class="table">
  ...
</div>
</body>
```

In this code, the selector name, `table`, is a good descriptive name, but you don't have to use it for your selector. You can use any selector name you want, but it should be descriptive.

In Figure 5.13, the descriptions for the `display` property values are pretty straightforward. The only description that needs clarification is the one for `table-header-group`. The `table-header-group` value causes its rows to display before all other rows and after any table captions (even if the `table-header-group` element code appears below any `table-row` element code). If a table contains multiple elements with `table-header-group` values, only the first such element behaves like a `thead` element; the others behave like `tbody` elements.

Table Values for the display Property	Description
<code>table</code>	Used to mimic a <code>table</code> element.
<code>table-caption</code>	Used to mimic a <code>caption</code> element.
<code>table-row</code>	Used to mimic a <code>tr</code> element.
<code>table-cell</code>	Used to mimic a <code>td</code> element or a <code>th</code> element.
<code>table-header-group</code>	Used to mimic a <code>thead</code> element.
<code>table-row-group</code>	Used to mimic a <code>tbody</code> element.

FIGURE 5.13 Table values for the CSS `display` property

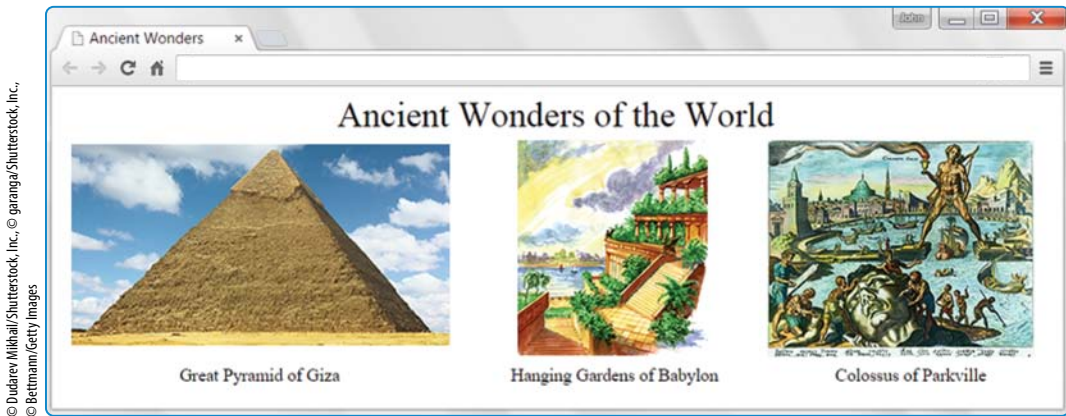


FIGURE 5.14 Ancient Wonders web page

One thing you might notice that's missing from Figure 5.13—there are no `display` property values that mimic the functionality of `rowspan` or `colspan`. Sorry, if you want that functionality, the most straightforward solution is to use the real-deal `rowspan` and `colspan` attributes in conjunction with a `table` element. As you might recall, you're supposed to use the `table` element only for data tables and not for table layout. But we'll let you in on a little secret: Browsers don't care. So if you don't mind getting razzed by Tim Berners-Lee at your next W3C cocktail party, you can use the `table`, `rowspan`, and `colspan` elements for table layout. If you do so, as was suggested in Section 5.7, you should add `role="presentation"` to the `table` element's start tag.

Example Web Page

Take a look at the Ancient Wonders web page in **FIGURE 5.14**. The pictures and the labels under the pictures are displayed using a two-row, three-column layout scheme. You could implement that layout scheme with either a `table` element or with CSS, so which is more appropriate? Because the web page does not contain data,³ you should use CSS. To have the table's column widths accommodate the widths of the pictures, you should use the CSS `display` property with table values. Using the `display` property with table values makes the web page easy to maintain. As the web developer, if you decide to replace one of the pictures with a wider or narrower picture, the picture's column will grow or shrink to accommodate the new picture.

³ The WHATWG states, "The `table` element represents data with more than one dimension." The WHATWG doesn't define what "data" is, so there is some ambiguity as to when it's appropriate to use the `table` element. But one thing that's clear is the desire of the standards committees to have web developers use a particular element only when its content coincides with the meaning that the element's tag name implies. Although it's a close call, in our opinion most nonprogrammers would refer to the Ancient Wonders web page as three pictures with labels, and they would not refer to it as a table. Therefore, given that assessment, we feel it would be inappropriate to use the `table` element for the Ancient Wonders web page.

In Figure 5.14, note that there are no borders for the table's exterior and no borders for the table's cells. That's the default for tables created with CSS. If we had wanted borders, we would have applied the CSS `border` property to the element we designate as a table and to the elements we designate as table cells. Now let's examine how we designate table elements with the CSS `display` property.

In **FIGURE 5.15's** Ancient Wonders style container, note the CSS rules that use the `.table`, `.caption`, and `.row` class selectors. In the body container, note how those selector names are used to implement a table using `div` elements—`<div class="table">`, `<div class="caption">`, and `<div class="row">`. The `div` element is good for implementing table components because, with one exception, it's generic. That means it doesn't add any formatting features of its own. The exception to that rule is that browsers generate newlines around `div` elements. That works great for implementing tables, captions, and rows because each of those entities is supposed to be surrounded by newlines.

Our implementation of the table cells in the Ancient Wonders web page requires some extra attention. As shown in Figure 5.13, to implement table cells with the `display` property, you need to use the `table-cell` value. In the Ancient Wonders table, each cell in the first row holds a picture, so our first implementation effort attempted to use the pictures' `img` elements as the targets for a `table-cell` rule, like this:

```
img {display: table-cell;}
```

Testing shows that this does not work. That should make sense when you think about what a table cell is supposed to be—a container for content. The `img` element is a void element, not a container, so it's inappropriate to try to use CSS to turn it into a table cell. The solution is to surround the `img` elements with `span` containers and use `span` as the target for a `table-cell` rule. Here's the most straightforward way to apply a `table-cell` value to a `span` element:

```
span {display: table-cell;},
```

If we were to use the CSS rule, then every `span` element would be implemented as a table cell. That's OK for the current version of the Ancient Wonders web page, but as a web developer, you should think about making your web pages maintainable. That means you should accommodate the possibility that you or someone else adds to your web page sometime in the future. If a `span` element is added that's not part of a table, the CSS rule would attempt to make it behave like a table cell, which would be inappropriate. In general, to avoid this kind of problem, you should not use a generic element, `span` or `div`, as the type for a type selector CSS rule.

So rather than using a type selector rule with `span`, we use a more elegant technique. We use a child selector rule that matches every element that is a child of a `row` element. This is justified because it's reasonable to assume that within a `row` element, every child element is a data cell. Here's the relevant rule from Figure 5.12's Ancient Wonders source code:

```
.row > * {display: table-cell;}
```

```

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="author" content="John Dean">
<title>Ancient Wonders</title>
<style>
  .table {
    display: table;
    border-spacing: 20px;
  }
  .caption {
    display: table-caption;
    font-size: xx-large;
    text-align: center;
  }
  .row {display: table-row;}
  .row > * {
    display: table-cell;
    text-align: center;
  }
</style>
</head>
<body>
<div class="table">
  <div class="caption">Ancient Wonders of the World</div>
  <div class="row">
    <span></span>
    <span></span>
    <span></span>
  </div>
  <div class="row">
    <span>Great Pyramid of Giza</span>
    <span>Hanging Gardens of Babylon</span>
    <span>Colossus of Parkville</span>
  </div>
</div>
</body>
</html>

```

For table behavior.

For caption behavior.

For tr behavior.

This causes all children of .row elements to behave like table cells (i.e., td or th elements).

This causes this div element to behave like a table.

This causes this div element to behave like a caption.

This causes this div element to behave like a row.

FIGURE 5.15 Source code for Ancient Wonders web page

You might recall that the `>` symbol is known as a combinator because it combines two selectors into one. The selector at the left, `.row`, matches all the elements in the Ancient Wonders web page that have `class="row"`. The universal selector at the right, `*`, matches any element. When

the two selectors are combined with `>`, the resulting child selector matches every element that is a child of a row element.

As explained earlier, the Ancient Wonders web page uses `div` elements to implement the table's table, caption, and row components. On the other hand, the Ancient Wonders web page uses `span` elements to implement the table's cells (which hold the table's pictures and labels). Like the `div` element, the `span` element is generic. Actually, even more generic. Browsers do not surround `span` elements with line breaks. That's a good thing for table cells because they should be displayed inline, without line breaks surrounding them.

The border-spacing Property

By default, tables created with the `display` property are displayed with no gaps between their cells. For the Ancient Wonders web page, that default behavior would have led to pictures that were touching. To avoid that ugliness, you can use the `border-spacing` property, and that's what we did in the Ancient Wonders `style` container:

```
.table {  
  display: table;  
  border-spacing: 20px;  
}
```

Go back to Figure 5.14 and note the space between the pictures and text in the Ancient Wonders web page. That is due to the `border-spacing` code.

When you use the `border-spacing` property, you should apply it to the entire table, not to the table's individual cells. Consequently, in the Ancient Wonders web page, we apply the `border-spacing` property shown to a `div` element that forms the entire table. In that example, we apply the `border-spacing` property to a table created with CSS. As an alternative, you can apply the `border-spacing` property to an old-fashioned HTML `table` element, and the effect is the same—space gets added between the table's cells.

The `border-spacing` property allows you to specify horizontal and vertical cell spacing separately. Here's an example:

```
border-spacing: 15px; 25px;
```

The first value, `15px`, specifies horizontal spacing, and the second value, `25px`, specifies vertical spacing. Horizontal spacing refers to the width of the gap between adjacent cells in the same row. Vertical spacing refers to the height of the gap between adjacent cells in the same column.

The `border-spacing` property adds space outside each cell's border. On the other hand, the `padding` property adds space inside each cell's border. If you want to see how the two properties differ, feel free to enter the Ancient Wonders web page code into your favorite web authoring tool, add `border-spacing` and `padding` property-value pairs to the table cell selector rule, and view the resulting web page.

In the past, we used the `margin` property to specify the space outside an element's border. So, can we use the `margin` property to specify the space outside an individual table cell's border? No, the `margin` property has no effect when used with elements that are defined to be table cells. That anomaly should make sense when you think about it. If the `margin` property was able to specify the space outside individual table cell borders, then you could specify different gap sizes between every pair of adjacent cells Yikes. What an unsightly table that would be! So the moral of the story is to use the `border-spacing` property to specify consistent horizontal and vertical gaps between table cells.

5.9 Absolute Positioning with CSS Position Properties

If you want table layout where content controls the size of the table's cells, then use the CSS `display` property with table values. But if you want table layout where the table's cell sizes are fixed, you should use CSS position properties. For **FIGURE 5.16**'s Imprint Express web page, which technique would be better—using the CSS `display` property with table values or using CSS position properties?

The Imprint Express web page implements an online newspaper, and newspapers should normally have columns with uniform widths. Using the CSS `display` property with table values wouldn't satisfy that goal. On the contrary, that technique would cause the column widths to be different. Why is that, you ask? Because the right column's natural width is greater than the left column's natural width. By "natural width," we mean the width of the columns if the window were wide enough to display both columns' content in a single row. The right column

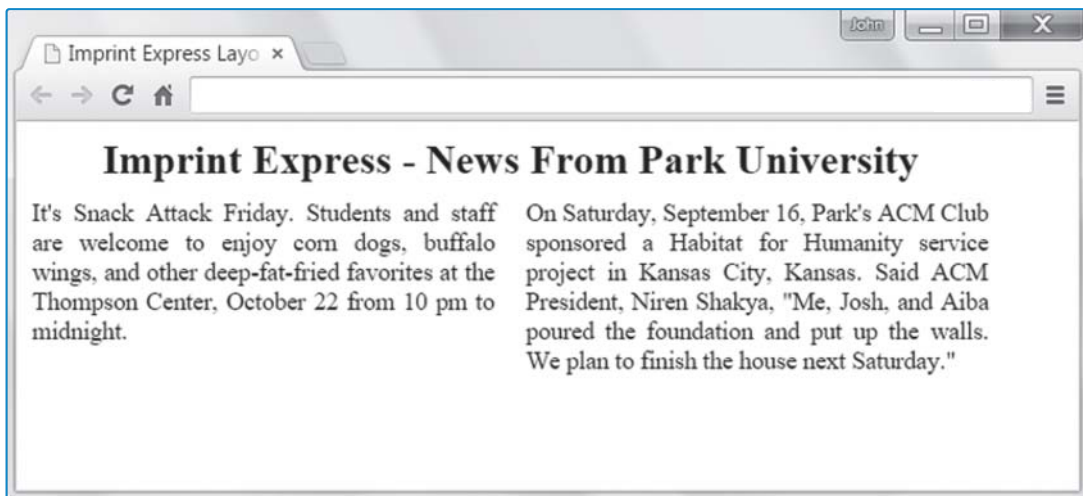


FIGURE 5.16 Imprint Express web page

contains more text than the left column, so the right column would be wider. If you then narrowed the window to induce line wrap, the browser would scale down both column widths, but it would continue to display the right column with a greater width than the left column. Note: You could force the two columns' natural widths to be the same by inserting `
` tags at the break points shown in the browser window ("staff," "buffalo," "the," etc.), but that would be a nightmare to maintain.

Thus, for the Imprint Express web page, using the CSS `display` property with table values would be inappropriate. Let's now see how CSS position properties would be more appropriate.

Browsers use a default layout scheme known as *normal flow*. With normal flow, within a container, block elements are laid out from top to bottom, one block element per row, and phrasing elements are laid out from left to right in the available space. To remove an element from the normal flow, you can use the CSS position properties shown in **FIGURE 5.17**. In the figure, note that the first position property is named "position." It's confusing to have a group of CSS properties referred to collectively as "position properties" and then have one of those properties—the most important one—use the name "position." Use the `position` property with a value of `absolute` to change the layout scheme from the default normal flow to *absolute positioning*. With absolute positioning, you use the `top` and `left` properties to specify where

CSS Position Properties	Description
<code>position: absolute</code>	Tells the browser to position the element relative to the containing block. Normally, the containing block is the web page's <code>html</code> element.
<code>left: value*</code>	Specifies the distance between the containing block's left edge and the element's left edge (where left edge means at the left of any padding, border, or margin).
<code>top: value*</code>	Specifies the distance between the containing block's top edge and the element's top edge (where top edge means above any padding, border, or margin).
<code>width: value*</code>	Specifies the width of the invisible box that constrains the element's contents. The width does not include padding, border, or margin.
<code>height: value*</code>	Specifies the height of the invisible box that constrains the element's text. The height does not include padding, border, or margin.

FIGURE 5.17 CSS position properties

* For the `left`, `top`, `width`, and `height` properties, you should normally provide number-of-pixels values, with `px` for the unit. But as an alternative, you can provide percentage-of-the-containing-block values, with `%` for the unit.

the target element's top-left corner is positioned relative to the element's *containing block*. Typically, an element's containing block resolves to the web page's `html` element. We'll talk later about how to designate a different containing block than the `html` element, but don't worry about that for now.

Note the bottom two properties in Figure 5.17—`width` and `height`. For an absolute positioned element, if you do not specify the element's width (with the `width` property), then the element's width will be determined by the natural width of the element's content. If the content is a picture, that's appropriate, because normally you will want the picture to display with its original dimensions. On the other hand, if the content is a significant amount of text, you should probably specify the element's width. Why? If you don't specify the width, the browser will try to widen the element to display all the text in a single line without any breaks. The `height` property is like the `width` property in that you might want to omit it (e.g., for pictures), but in many cases, you'll want to include it. If you want absolute positioned boxes to be aligned vertically at their bottoms, then for each box you should include not only the `top` property (with the same value for each box), but also the `height` property (with the same value for each box).

Now back to the Imprint Express web page. Before showing you how we implement that web page's layout with CSS position properties, let's design the layout with a drawing. See the drawing in **FIGURE 5.18**. Can you visualize the title positioned in the top region and the two newspaper articles positioned in the left and right regions?

Normally, when designing a web page with CSS position properties, you should start by drawing the layout using paper and a pencil. In your drawing, show pixel measurements for the dimensions of your web page's regions (as in Figure 5.18). By looking at a drawing that's roughly scaled to pixel measurements, you can get a pretty good idea of whether the resulting web page's layout will be aesthetically pleasing. It's good to know that in advance before writing the code and wasting time with recoding efforts if the result looks bad.

To position the Imprint Express web page's left and right newspaper articles using the layout shown in Figure 5.18, you could use `.left-article` and `.right-article` selector rules like this:

```
.left-article {                .right-article {
  position: absolute;          position: absolute;
  top: 50px; left: 10px;       top: 50px; left: 330px;
  width: 300px; height: 180px; width: 300px; height: 180px;
}                               }
```

As a sanity check, verify that the rules' pixel values agree with the figure's pixel values. For instance, in the rules, the `top` property has a value of `50px`, and in the figure, the measurements above the two newspaper article regions add up to 50 ($10 + 30 + 10 = 50$).

The problem with the rules shown is that there's quite a bit of *code redundancy*. Both rules contain identical (redundant) values, except for the `left` property's value. With code redundancy, if you later decide to make a change (or someone else decides to make a change), you have to remember to make the change in both places. Thus, code redundancy makes web page maintenance more difficult.

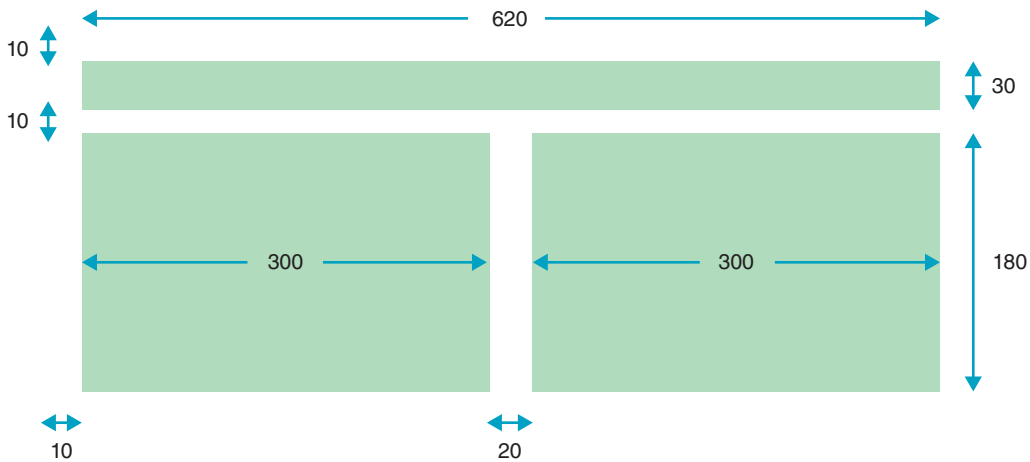


FIGURE 5.18 Layout drawing for the Imprint Express web page

In the actual web page, we avoid code redundancy by putting the property-value pairs that are common to both newspaper article regions in a CSS rule that applies to both of the regions. We also put the `left` property-value pairs in CSS rules that apply to each newspaper article region separately. To see those rules and the rest of the Imprint Express web page code, examine [FIGURE 5.19](#). In particular, note how we use an `article` type selector for the rule that applies to both of the newspaper article regions. Note also how we use `article.left` and `article.right` class selectors for the rules that apply to each newspaper article region separately. Prefacing `.left` and `.right` with `article` is not required, but we use the `article` prefix as a form of self-documentation. Also, it allows you to have `left` and `right` class attributes for an element different from the `article` element where the two sets of `left` and `right` class attributes have different presentation characteristics.

In Figure 5.19's code for the Imprint Express web page, note this CSS rule:

```
* {margin: 0; padding: 0;}
```

We use the universal selector, `*`, to assign a value of 0 to every element's margin and padding areas. In doing so, we enable the CSS position property values to be applied without interference from default margins or borders. If we did not zero out the margin and padding values, then the title's `h1` element would use its default nonzero value for the top margin, and that would cause the title to move down and slightly overlap the two articles. That wouldn't be good.

There's one additional noteworthy item in the Imprint Express web page code. Normally, for the `font-size` property, we've been using values with `em` units (where `em` is the height of the letter M in the current context). For the Imprint Express web page, we use units of `pt` (for points) for the `font-size` properties because the web page is supposed to mimic a newspaper. We use 20-point font for the heading and 12-point font for the article text because those are standard sizes for newspapers.

```

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="author" content="John Dean">
<title>Imprint Express Layout</title>
<style>
  * {margin: 0; padding: 0;}
  .title {
    position: absolute;
    left: 10px; top: 10px;
    width: 620px; height: 30px;
    text-align: center;
    font-size: 20pt;
  }
  article {
    position: absolute;
    top: 50px;
    width: 300px; height: 180px;
    text-align: justify;
    font-size: 12pt;
  }
  article.left {left: 10px;}
  article.right {left: 330px;}
</style>
</head>

<body>
<h1 class="title">Imprint Express - News From Park University</h1>
<article class="left">
  It's Snack Attack Friday. Students and staff are welcome to
  enjoy corn dogs, buffalo wings, and other deep-fat-fried favorites
  at the Thompson Center, October 22 from 10 pm to midnight.
</article>
<article class="right">
  On Saturday, September 16, Park's ACM Club sponsored a Habitat for
  Humanity service project in Kansas City, Kansas. Said ACM President,
  Niren Shakya, "Me, Josh, and Aiba poured the foundation and put up
  the walls. We plan to finish the house next Saturday."
</article>
</body>
</html>

```

Set all elements' margins and paddings to 0 so the content positioning relies solely on the absolute position values without interference from default margins or borders.

This article type selector rule specifies the CSS properties that apply to both articles.

These class selector rules specify the `left` property, which has a different value for each article.

FIGURE 5.19 Source code for Imprint Express web page

5.10 Relative Positioning

In the previous section, you learned about absolute positioning, where you specify `position: absolute` in a CSS rule in order to position an element relative to the top-left corner of its containing block. As an alternative, you can position an element relative to its normal flow within its surrounding content. That's called *relative positioning*. Take a look at the Divine Comedy web page in **FIGURE 5.20**, where we use relative positioning to move “height of ecstasy” up and “depths of despair” down.

In the Divine Comedy web page's CSS rules, note the `position: relative` property-value pairs. That's how relative positioning is established. In the `.down` CSS rule, note the property-value pairs `top: 2em` and `left: 2em`. Positive values for the `top` property move the element down, and positive values for the `left` property move the element to the right. Thus, in the web page, you can see that “depths of despair” moves 2 font size positions down and 2 font size positions to the right. When applying relative positioning to an element, the element

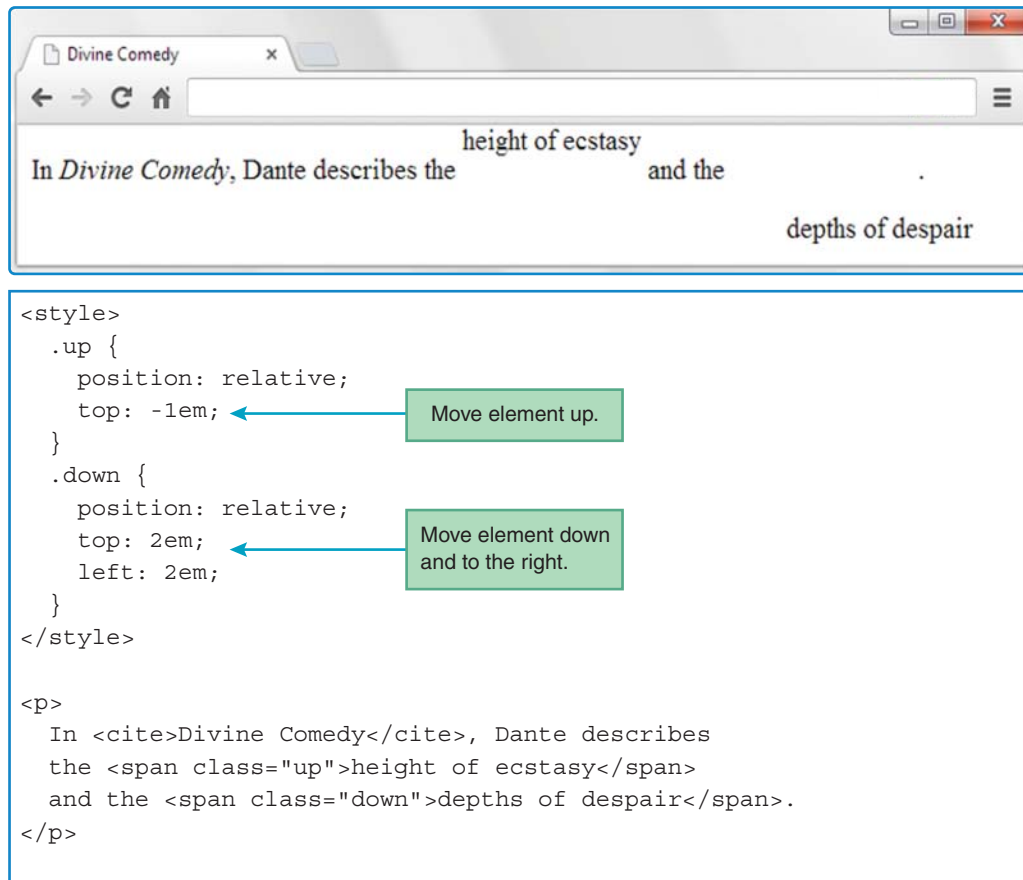


FIGURE 5.20 Divine Comedy web page and its relative positioning code

moves, but the content after the element behaves as if the element stayed in its normal flow position. The web page illustrates this phenomenon by positioning the sentence's period at the right of the normal flow position of "depths of despair." In the web page's `.up` CSS rule, note the property-value pair `top: -1em`. The `-1em` value causes "height of ecstasy" to move up by one font size position.

Using relative positioning to position an element within its normal flow can look strange, so it's not used all that often. But it can be useful for positioning the `body` element.

Centering the Entire Page Within a Browser Window

After creating the Imprint Express web page as shown in Figure 5.16, suppose you get the urge to center the page's contents. Centering the heading by itself would be easy. In the `h1` element's `.title` CSS rule, scrap all the absolute positioning property-value pairs and retain only the `text-align: center` property-value pair. The `text-align: center` property-value pair would work great for the `h1` element because the `text-align` property's target is plain text, and `h1` contains plain text. On the other hand, centering the two articles is not as easy. You could use the `text-align` property to center the text within each individual article's block, but that's not helpful. You need to center the blocks themselves within the browser window, with the two blocks side-by-side. To do that, the two `article` elements need to be wrapped in a container, and the container needs to be repositioned so its contents are centered.

In the Imprint Express web page, is there already a container that surrounds the two `article` elements? Yes, indeed! The `body` container surrounds them, and as an added bonus, it also surrounds the `h1` element. The goal is to reposition the `body` element within the browser window, so its enclosed elements (the `h1` element and the two `article` elements) become centered horizontally in the browser window.

Do you remember how we centered the `table` element in the Global Temperatures web page? We used CSS to turn the `table`'s parent container (the `body` element) into a flexbox, and then we applied `justify-content: center` to that parent container. We'll do the same thing here, but this time the element that needs centering is the `body` element. So, what is the `body` element's containing element? That's right; it's the `html` element. Here's what we need:

```
html {  
    display: flex;  
    justify-content: center;  
}
```

We want the `body`'s enclosed elements to use absolute positioning relative to the repositioned `body`. To do that, we need the `body` container to be the containing block for its enclosed elements. That sounds like it should be the default, but not so. In the previous section, we said that an element's containing block normally resolves to the web page's `html` element. For this particular situation, we don't want the containing block to be the `html` element. To force the `body` container to be the containing block for the `body`'s enclosed

elements, we need to apply a `position: relative` CSS rule to the `body` element. Specifically, here's what's needed:

```
body {  
    position: relative;  
    width: 620px;  
    height: 220px;  
}
```

In the rule, the `width` and `height` properties (whose values come from the dimensions shown in Figure 5.18) are necessary so the browser engine can establish a framework to which other elements can be added. For example, later you'll see how we position the web page's second newspaper article at the right side of the web page by specifying 0 for the property that indicates distance from the right side of the `body` element. Without the `width` value, the browser won't know the location of the right side.⁴ For this particular web page, there's no need to position any elements using the page's bottom edge as the starting point, so it's OK to omit the `height` property. On the other hand, you might want to keep it as a form of self-documentation.

To center the web page's contents, not only do you need a `body` CSS rule with `position: relative` as shown, you also need a CSS rule with `position: absolute` for each of the elements inside the `body` container. **FIGURE 5.21** shows the CSS code for the centered version of the Imprint Express web page. The CSS rules that start with `.title` and `article` apply to the `title` element and the `article` elements, respectively. To have the `body` container's centered position take effect, both rules include `position: absolute`.

In Figure 5.21, the `body` CSS rule is slightly different from the `body` CSS rule presented earlier. We add a `top` property with a value of 30 pixels in order to shift the `body` container down (notice the expanded blank area at the top of Figure 5.21's web page display). Also, to reduce clutter, we omit the `height` property.

By using relative positioning for the web page's `body`, we are able to simplify the CSS code for the elements inside the `body` container. Specifically, there is no need for `left` and `top` properties in the `title` element's CSS rule because the `title` element now uses `left` and `top` positioning from the repositioned `body` container. Likewise, there is no need for an `article.left` CSS rule because the left article's element now uses `left` positioning from the repositioned `body` container. For the `article.right` CSS rule, we could have used `left: 320px` to position the right article 320 pixels from the left edge of the centered `body`. As a more elegant alternative, we use `right: 0` to position the right article at the right edge of the centered `body`. The `right` property, not shown in Figure 5.17's table of CSS position properties, works as expected—it specifies the distance between the containing block's right edge and the current element's right edge.

⁴Without relative positioning, there's no need to specify the `body` element's dimensions because the browser can calculate the `body`'s default dimensions from the size of the browser window's *viewport* (where the viewport is the large area below the address bar where web page content displays). But with relative positioning, the page dimensions are 0's by default, and that's not helpful for positioning purposes.

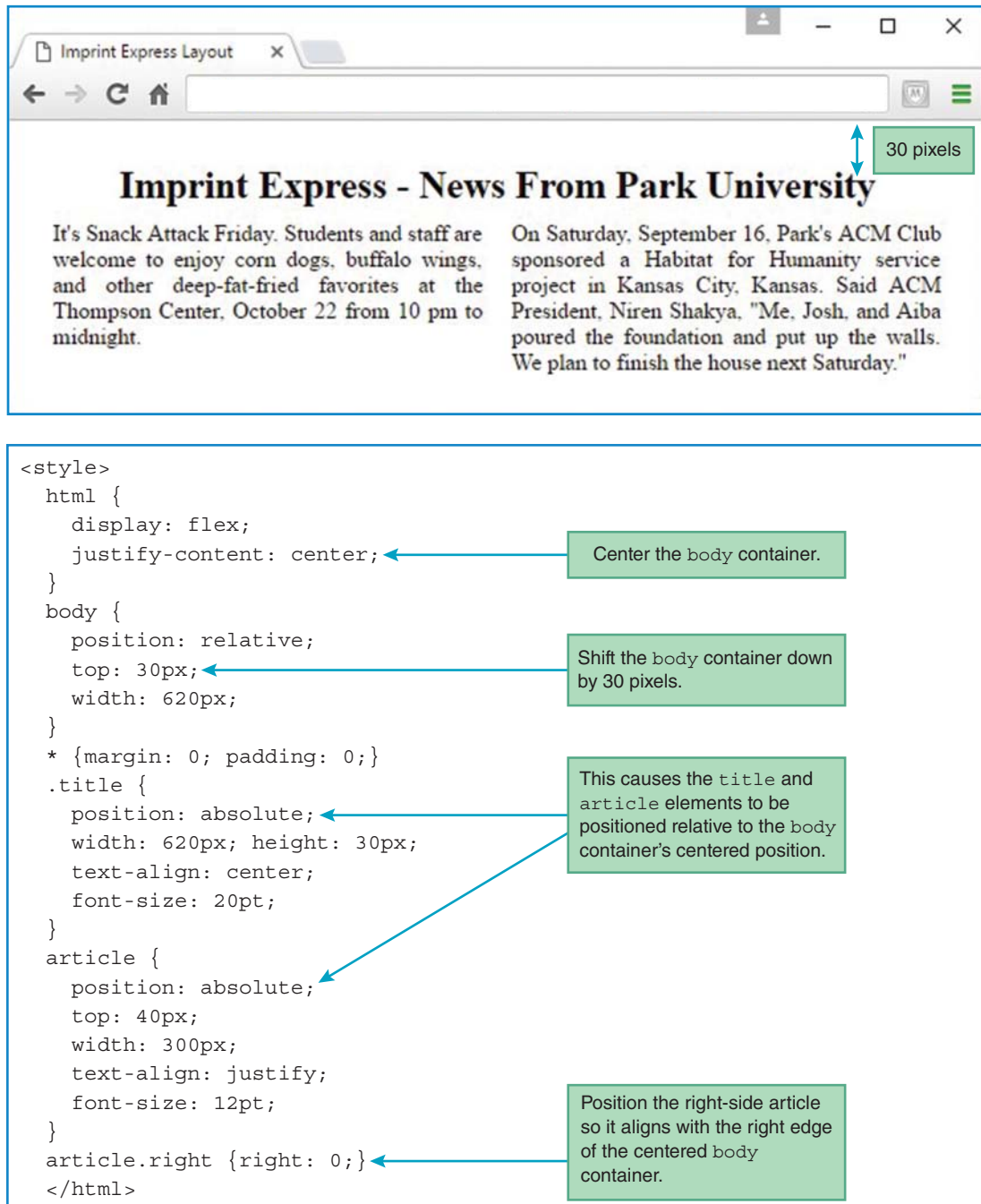


FIGURE 5.21 Centered Imprint Express web page and its CSS code

Relative Positioning for an Element Without Adjusting the Element's Position

So far, we've used the `position: relative` property-value pair to adjust an element's position relative to its normal flow position. You can also use the `position: relative` property-value pair without adjusting the element's position. In that case, the purpose of the `position: relative` property-value pair is to designate the element as a containing block for absolute positioned elements inside it. For example, suppose you want to display several business cards on a web page. Each business card displays a person's name, phone number, and email address. You implement each business card with a `div` element and within each `div`, you use `span` elements for the name, phone number, and email values. For each business card `div` element, you apply a `position: relative` CSS rule. For each `span` element, you apply a `position: absolute` CSS rule with values for the `left`, `top`, `width`, and/or `height` properties. Because of the surrounding `div` element's `position: relative` CSS rule, the `left` and `top` values position the `span` element relative to the `div` element. Each business card's `span` elements are positioned in the same places within the card, so you use the same set of CSS `span` rules for all the business cards. Sharing the CSS `span` rules (and sharing code in general) can lead to a more maintainable web page. If you have to change the positions of the `span` elements within each business card, you make the change in one place—in the shared set of CSS `span` rules—rather than making changes to separate CSS `span` rules for each business card.

To illustrate relative positioning in the context of a complete web page, we could provide the code for the aforementioned business card web page, but that would spoil the fun. This chapter has a project that describes the business card web page in greater detail, and we encourage you to implement it.

5.11 CASE STUDY: A Downtown Store's Electrical Generation and Consumption

The web page for this segment of our ongoing case study describes the smallest kind of electrical microgrid—the electrical system for a single small shop or restaurant. Rather than try to specify everything at the start, we'll develop this web page in a sequence of iterations. This strategy minimizes the number of new problems faced at each stage of the design process, and it produces alternative options for different hardware platforms.

For style consistency, we'll continue to format the page header with the external CSS file introduced in Chapter 3, `microgrid.css`. This file specifies the body's width, padding, and background color. After the header, the web page contains an introductory block of text. Next, a table contains the salient features of a typical electrical microgrid. After the table, there are two blocks of text for a system description and an identification of key assumptions used. At the bottom of the web page, there is a one-line summary of the savings produced by the local system's own renewable energy source.

In our first design iteration, we'll focus on the table and use simple paragraphs for the subsequent blocks of text. **FIGURE 5.22** shows the resulting Typical Downtown Property web page.

As in previous segments of this case study, the text in the top banner is an `h2` header. The table caption and the headers below the table are in `h3` elements. For formatting, they use `text-align: center`, as specified in the external CSS file. The text blocks are in `p` containers with `text-align: justify` formatting. As expected, the table uses a table container with a caption element, followed by `tr` elements with embedded `th` elements.

The only subtleties are in the `style` container's CSS code that formats the table:

```
.center {
  display: flex; justify-content: center;
}
table {border-collapse: collapse;}
th, td {
  border: thin solid;
  padding: 2px 5px;
}
td:nth-child(3), td:nth-child(4) {
  text-align: right;
}
```

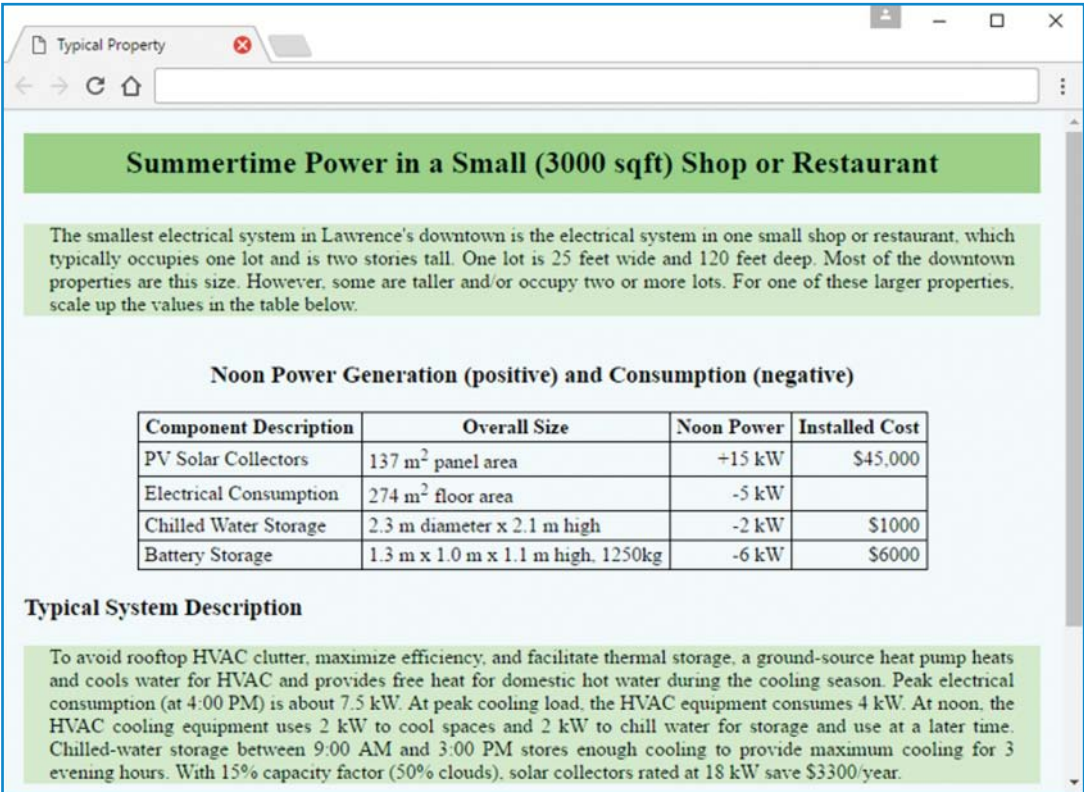
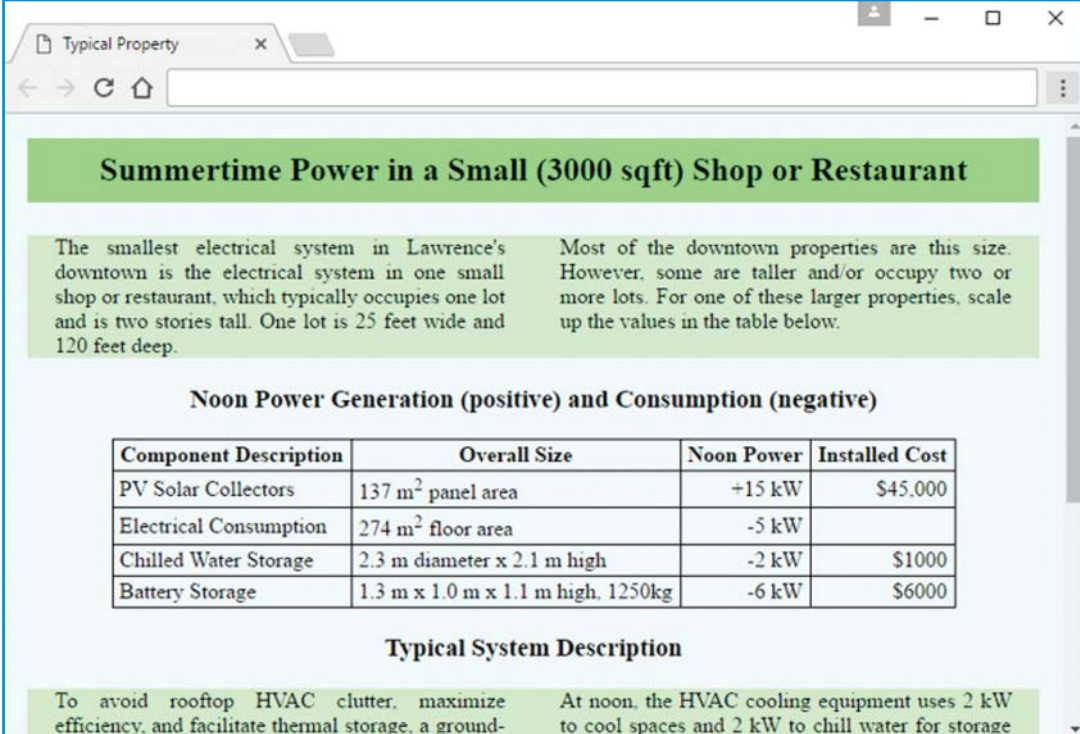


FIGURE 5.22 Case Study: Typical Downtown Property web page, first design iteration

As you learned earlier, to center an element (like a table) horizontally, you can position the element in a flexbox and apply `justify-content: center` to the flexbox. In this code, you can see how we use that CSS property-value pair as part of a `.center` selector rule. We apply that rule to a `div` container that surrounds the table. In the `table` rule, the `border-collapse: collapse` property-value pair makes the table cleaner. The `th, td` rule causes cell borders to be visible and adds padding for each cell. Finally, the `:nth-child()` pseudo-class selectors rule specifies right alignment for the numerical data in the last two columns without changing the default left alignment for the text data in the first two columns.

We suggest that you now write the code for this first-iteration Typical Downtown Property web page using the suggestions. No need to worry about the text that's out of view at the bottom of Figure 5.22. When you're done, you should retrieve the `typicalPropertyFirstCut.html` file from the book's website and compare its source code to yours.

If you load the Typical Downtown Property web page in a browser, you can see that the content remains fairly readable if you reduce the page's width. On the other hand, if you expand the page's width too much, you can get to the point where the lines of text become longer than what you'd want for a comfortable reading experience. If you feel a web page is likely to appear in wide windows, you might want to consider organizing the text in columns, as in a newspaper. **FIGURE 5.23** shows the same information presented in Figure 5.22, but each block of text is split into two columns.



Summertime Power in a Small (3000 sqft) Shop or Restaurant

The smallest electrical system in Lawrence's downtown is the electrical system in one small shop or restaurant, which typically occupies one lot and is two stories tall. One lot is 25 feet wide and 120 feet deep.

Most of the downtown properties are this size. However, some are taller and/or occupy two or more lots. For one of these larger properties, scale up the values in the table below.

Noon Power Generation (positive) and Consumption (negative)

Component Description	Overall Size	Noon Power	Installed Cost
PV Solar Collectors	137 m ² panel area	+15 kW	\$45,000
Electrical Consumption	274 m ² floor area	-5 kW	
Chilled Water Storage	2.3 m diameter x 2.1 m high	-2 kW	\$1000
Battery Storage	1.3 m x 1.0 m x 1.1 m high, 1250kg	-6 kW	\$6000

Typical System Description

To avoid rooftop HVAC clutter, maximize efficiency, and facilitate thermal storage, a ground-

At noon, the HVAC cooling equipment uses 2 kW to cool spaces and 2 kW to chill water for storage

FIGURE 5.23 Case Study: Typical Downtown Property web page, second design iteration

For our second design iteration, we'll split each block of text into two columns by replacing each `p` element in the first iteration with a `div` element containing a pair of subordinate `div` elements—one for each column. Thus, for example, for the first block of text, instead of using this:

```
<p>
  The smallest electrical system in Lawrence's downtown ...
  ... values in the table below.
</p>
```

We'll use this:

```
<div class="row">
  <div class="cell">
    The smallest electrical system in Lawrence's downtown ...
    ... occupies a single lot.
  </div>
  <div class="cell">
    One lot is 25 feet wide and 120 feet deep. Most of the ...
    ... values in the table below.
  </div>
</div>
```

This requires two CSS rules. So the `style` element at the top of the web page will get these two additional components:

```
.row {
  display: table-row;
}
.cell {
  display: table-cell;
  width: 40%;
  padding: 0px 20px;
  text-align: justify;
}
```

This causes the new outer `div` element to display like a table row, and it causes the new inner `div` elements to display like table cells. The two cells are the two columns. In the course of doing this, we also insert a `text-align: justify` property to straighten the ragged right sides in the text blocks.

Now we suggest that you use this method to modify the text blocks in your first-iteration code so that your web page generates a display like that shown in Figure 5.23. When you're done, you should retrieve the `typicalProperty.html` file from the book's website and compare its source code to yours.

Review Questions

5.2 Table Elements

1. For each of the following elements, what is its HTML start tag?
 - a) A table row element
 - b) An element within a table row that holds data
 - c) An element in the top row of a table that describes the data in the column below

5.3 Formatting a Data Table: Borders, Alignment, and Padding

2. Using type selectors, write a CSS rule that creates a thin solid single-line border around all tables and uses a single line to separate adjacent columns and adjacent rows.
3. Using type selectors, write a CSS rule that puts 5 pixels of padding around the text in each table header and data cell.

5.4 CSS Structural Pseudo-Class Selectors

4. Write a CSS structural pseudo-class selector that selects the odd table row elements, starting with the third one.

5.5 `thead` and `tbody` Elements

5. What is the purpose of the `thead` element, and what is the purpose of the `tbody` element?
6. What does the following CSS rule do?

```
body {display: flex; justify-content: center;}
```

5.6 Cell Spanning

7. Assume the following `style` and `body` elements are part of a complete html5 document. Provide a sketch that shows what the code displays.

```
<style>
  th, td {padding: 5px 20px;}
  table, th, td {border: solid thin; border-collapse: collapse;}
</style>

<body>
<table>
  <tr> <th></th> <th colspan="2"></th> <th></th> </tr>
  <tr> <td></td> <td></td> <td></td> <td></td> </tr>
  <tr>
    <td rowspan="3"></td> <td></td> <td></td>
    <td rowspan="3"></td>
  </tr>
```

```
    <tr> <td></td> <td></td> </tr>
    <tr> <td></td> <td></td> </tr>
  </table>
</body>
```

5.7 Web Accessibility

8. What is a screen reader?

5.8 CSS `display` Property with Table Values

9. For each HTML element, provide a CSS property-value pair that implements its presentation functionality.
 - a) `table`
 - b) `caption`
 - c) `tr`
 - d) `td`
10. Why is the `div` element a good choice for CSS implementation of table and row components?
11. The `display: table-cell` property does not work well with elements in the embedded category like `img`, `audio`, and `video`. How can you work around this problem?

5.9 Absolute Positioning with CSS Position Properties

12. Describe how to apply absolute positioning to an element.
13. With absolute positioning, the `top` and `left` properties indicate the position of the target element's top-left corner relative to what?
14. What CSS properties establish offsets from container sides to corresponding component sides?

5.10 Relative Positioning

15. With relative positioning, the `top` and `left` properties indicate the position of the target element's top-left corner relative to what?
16. If you apply the `position: relative` property-value pair to a container, that container can then serve as a containing block for absolute positioned elements inside it. True or false.

Exercises

1. [after Section 5.3] Suppose you have a table implemented with a `table` element, and borders are displayed around each table cell and around the table's perimeter. Provide a CSS type selector rule that eliminates the gaps between each of the adjacent borders.

2. [after Section 5.3] There are quite a few syntax details in HTML5 and in CSS. This book covers a lot of those details, but certainly not all of them. One thing we skipped in this chapter is the `colgroup` element, which is used in conjunction with the `table` element. For this exercise, you'll need to learn about the `colgroup` element by looking it up on the W3C's website.

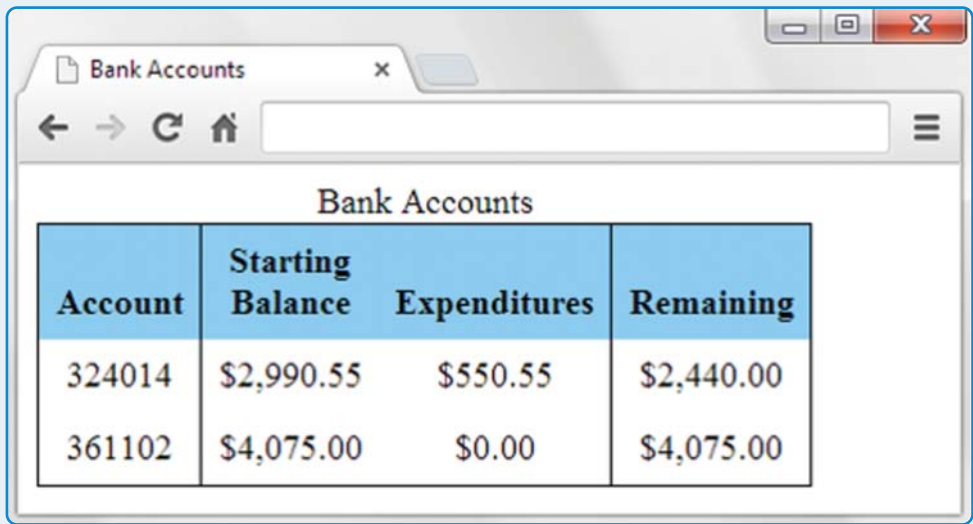
Given this code:

```
<style>
  table {border-collapse: collapse;}
  th, td {
    padding: .5em;
    text-align: center;
    vertical-align: bottom;
  }
  th {background: lightskyblue;}
</style>
</head>

<body>
<table>
  <caption>Bank Accounts</caption>
  <tr>
    <th>Account<br></th>
    <th>Starting<br>Balance</th>
    <th>Expenditures</th>
    <th>Remaining</th>
  </tr>
  <tr>
    <td>324014</td>
    <td>$2,990.55</td>
    <td>$550.55</td>
    <td>$2,440.00</td>
  </tr>
  <tr>
    <td>361102</td>
    <td>$4,075.00</td>
    <td>$0.00</td>
    <td>$4,075.00</td>
  </tr>
</table>
</body>
```

What `colgroup` element code and CSS code need to be added to this code in order to produce the following web page? As its name implies, the purpose of the `colgroup` element is to group together a set of columns. In this case, you must provide three `colgroup` elements—one for the first column, one for the second and third columns combined, and

one for the fourth column. In your answer, you must make it clear where your added code should be inserted in the code.



Account	Starting Balance	Expenditures	Remaining
324014	\$2,990.55	\$550.55	\$2,440.00
361102	\$4,075.00	\$0.00	\$4,075.00

3.

[after Section 5.4] What is the purpose of a pseudo-class?
4.

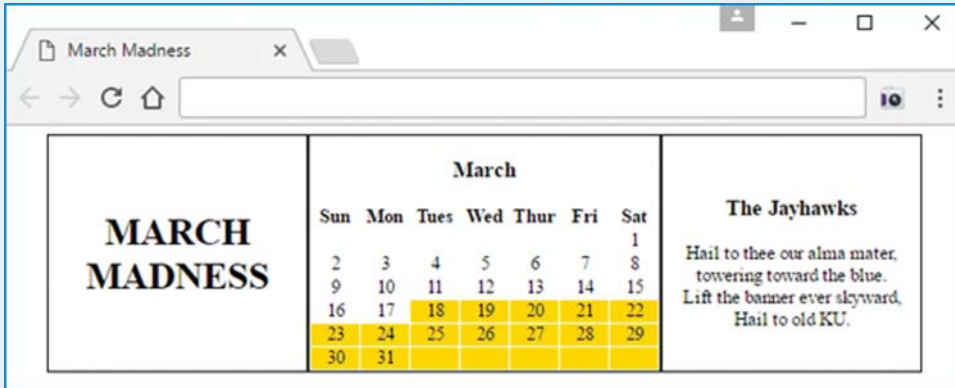
[after Section 5.4] Describe the effect of this CSS rule:

```
<CD>tr:nth-of-type(odd) {color: red;}</CD>
```
5.

[after Section 5.5] Refer to the Global Temperatures source code in Figure 5.8. Provide a modified table that uses two `tbody` elements—one for the two most recent years (2015 and 2016) and one for the earlier years. Assign "recentYears" to the `class` attribute of the `tbody` containing the two most recent years. Provide a CSS rule for the recent-years data cells that changes the text from black to red. Hint: Use a class selector in conjunction with a descendant selector.
6.

[after Section 5.7] Refer to the My Favorite Eras source code in Figure 5.10. Provide a modified table that is more accessible. Specifically, add an `id` attribute value to the `th` element at the head of each column and each row. Also, add a `header` attribute to the Mesozoic and Cenozoic `th` elements and to every `td` element.

7. [after Section 5.8] A table is embedded inside one cell of a larger structure formed using `display: table`, `display: table-row`, and `display: table-cell`. The embedded table is the middle cell in the three bordered cells in this screenshot:



This exercise also provides practice using structural pseudo-class selectors. The selectors apply gold background to the days on the calendar during which NCAA basketball tournament games are played. Your display will look like the screenshot, except for the calendar dates. Use the current year, which means March dates will probably fall on different days of the week. Also, the cell on the right should contain material related to *your* favorite team.

Provide complete source code for the March Madness web page. The following template provides most of the code, so use it as a starting point. In the left section, you'll need to add "MARCH MADNESS." In the middle section, you'll need to provide content for the calendar by adding March, Sun, Mon, ..., 1, 2, etc. In the right section, you'll need to replace "Jayhawks" and the fight song with text for your favorite team. In the structural pseudo-class CSS rule, you'll need to provide appropriate values in the selectors' parentheses. Figuring out those values is a bit tricky. Here's a hint: To do something to a particular row and all subsequent rows or a particular cell within a row and all subsequent cells, use the formula, $1n+\#$, where $\#$ is the starting row number or starting cell number.

```

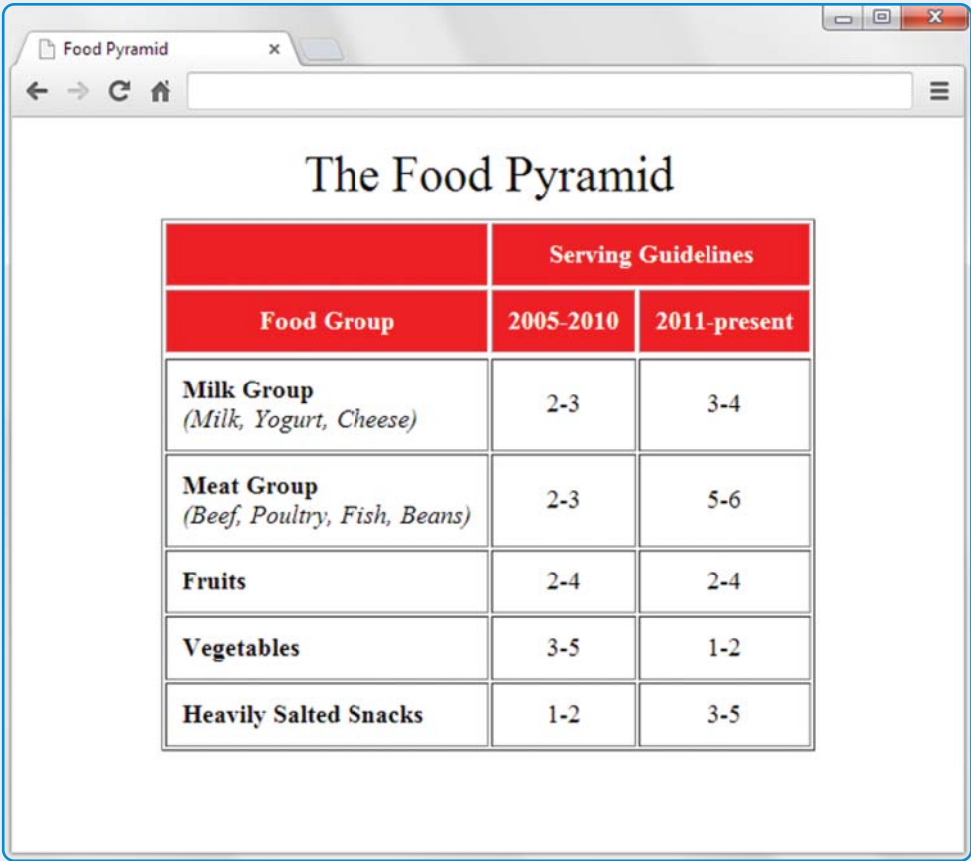
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="author" content="John Dean">
<title>March Madness</title>
<style>
  html {display: flex; justify-content: center;}
  body {display: table;}
  .row {display: table-row;}
  .cell {
    display: table-cell;
    border: thin solid;
    text-align: center;
    vertical-align: middle;
  }
  .end {width: 200px; padding: 10px;}
  th {width: 40px;}
  tr:nth-of-type() td:nth-of-type(), tr:nth-of-type() {
    background-color: gold;
  }
</style>
</head>

<body>
  <div class="row">
    <div class="end cell">
      <h1></h1>
    </div>
    <table class="cell">
      <caption><h3></h3></caption>
      <tr>
        <th></th><th></th><th></th><th></th><th></th><th></th><th></th>
      </tr>
      <tr>
        <td></td><td></td><td></td><td></td><td></td><td></td><td></td>
      </tr>
      <tr>
        <td></td><td></td><td></td><td></td><td></td><td></td><td></td>
      </tr>
      <tr>
        <td></td><td></td><td></td><td></td><td></td><td></td><td></td>
      </tr>
    </table>
  </div>

```


Projects

- 1. Create the following web page and name the file `foodPyramid.html`:



Note:

- ▶ Use a `table` element with appropriate other elements, including `thead` and `tbody` elements.
- ▶ The table should be horizontally centered with respect to the browser window.
- ▶ Look carefully—padding has been added to the table's caption.

2. Create the following web page and name the file `gymnastics.html`:



You are required to use appropriate organizational elements. Use a header container for the top two headings. Use two section containers—one for the female gymnasts content and one for the male gymnasts content. Use a figure container for the image and caption. You are required to use absolute positioning for the header, section, and figure containers.

Extra Credit

Modify the gymnastics page so that there is a 20-pixel gap around the page's contents. Hint: For an elegant implementation, you'll need to use the `position: relative;` property-value pair.

Modify the gymnastics page so that the female gymnast list items are numbered continuously and the male gymnast list items are numbered continuously. See the display for details.

