TUTORIAL 3

Designing a Page Layout

Creating a Website for a Chocolatier

OBJECTIVES

Session 3.1
- Create a reset style sheet
- Explore page layout designs
- Center a block element
- Create a floating element
- Clear a floating layout
- Prevent container collapse

Session 3.2
- Use CSS grid styles
- Define a grid layout
- Place items within a grid
- Work with grid areas

Session 3.3
- Explore positioning styles
- Work with relative positioning
- Work with absolute positioning
- Work with overflow content

Case | *Pandaisia Chocolates*

Anne Ambrose is the owner and head chocolatier of *Pandaisia Chocolates*, a chocolate shop located in Essex, Vermont. You have been asked to assist on the redesign of the company's website. Anne has provided you with three pages from the website to start your work. She has written all of the content, compiled the necessary images and graphics, and written some of the text and color styles. She needs you to complete the project by designing the page layout using the CSS layout properties.

STARTING DATA FILES

tutorial
pc_about_txt.html
pc_home_txt.html
pc_info_txt.html
pc_grids_txt.css
pc_home_txt.css
pc_reset_txt.css
+ 22 files

review
pc_specials_txt.html
pc_specials_txt.css
+ 12 files

code1
code3-1_txt.html
code3-1_float_txt.css
+ 11 files

code2
code3-2_txt.html
code3-2_layout_txt.css
+ 1 file

code3
code3-3_txt.html
code3-3_scroll_txt.css
+ 10 files

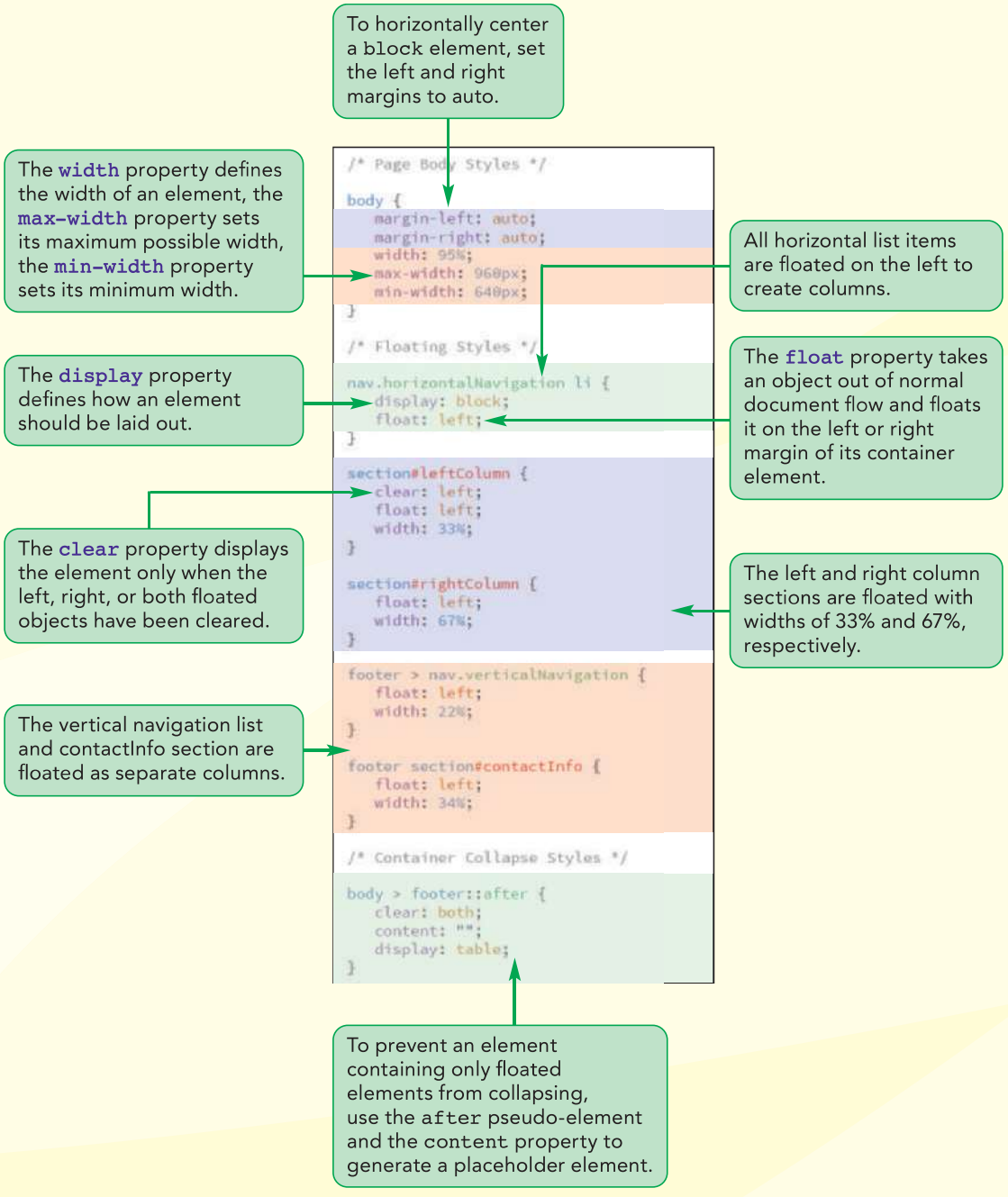
code4
code3-4_txt.html
debug3-4_txt.css
+ 2 files

case1
sp_home_txt.html
sp_layout_txt.css
+ 13 files

case2
ss_dday_txt.html
ss_layout_txt.css
+ 4 files

demo
demo_grid1.html
demo_grid2.html
demo_positioning.html
+ 6 files

Session 3.1 Visual Overview:



Page Layout with Floating Elements

Introducing the display Style

The study of page layout starts with defining how an individual element is presented on the page. In the first tutorial, you learned that HTML elements are classified into block elements, such as paragraphs or headings, or into inline elements, such as emphasized text or inline images. However, whether an element is displayed as a block or as inline depends on the style sheet. You can define the display style for any page element with the following `display` property

```
display: type;
```

where *type* defines the display type. A few of the many *type* values are shown in Figure 3–1.

Figure 3–1 Some values of the display property

Display Value	Appearance
`block`	Displayed as a block
`table`	Displayed as a web table
`inline`	Displayed inline within a block
`inline-block`	Treated as a block placed inline within another block
`run-in`	Displayed as a block unless its next sibling is also a block, in which case, it is displayed inline, essentially combining the two blocks into one
`inherit`	Inherits the display property of the parent element
`list-item`	Displayed as a list item along with a bullet marker
`none`	Prevented from displaying, removing it from the rendered page

For example, to supersede the usual browser style that displays images inline, you can apply the following style rule to display all of your images as blocks:

```
img {display: block;}
```

If you want to display all block quotes as list items, complete with list markers, you can add the following style rule to your style sheet:

```
blockquote {display: list-item;}
```

TIP

You also can hide elements by applying the style `visibility: hidden;`, which hides the element content but leaves the element still occupying the same space in the page.

You can even prevent browsers from displaying an element by setting its `display` property to `none`. In that case, the element is still part of the document structure but it is not shown to users and does not occupy space in the displayed page. This is useful for elements that include content that users shouldn't see or have no need to see.

You'll use the `display` property in creating a reset style sheet.

Creating a Reset Style Sheet

You learned in the last tutorial that your browser applies its own styles to your page elements unless those styles are superseded by your own style sheet. Many designers prefer to work with a "clean slate" and not have any browser style rules creep into the final design of their website. This can be accomplished with a **reset style sheet** that supersedes the browser's default styles and provides a consistent starting point for page design.

You'll create a reset style sheet for the Pandaisia Chocolates website. The first style rule in your sheet will use the `display` property to display all of the HTML 5 structural elements in your web page as blocks. While current browsers already do this, there are some older browsers that do not recognize or have predefined display styles for elements as such `header`, `article`, or `footer`. By including the `display` property in a reset style sheet, you add a little insurance that these structural elements will be rendered correctly.

To create a reset style sheet:

1. Use the text editor or HTML editor of your choice to open the **pc_reset_txt.css** file from the html03 ▸ tutorial folder. Enter ***your name*** and ***the date*** in the comment section of the file and save the document as **pc_reset.css**.
2. Within the Structural Styles section, insert the following style rule to define the display properties of several HTML 5 structural elements:

```
article, aside, figcaption, figure,
footer, header, main, nav, section {
   display: block;
}
```

Figure 3–2 highlights the new style rule in the document.

Figure 3–2 Displaying structural elements as blocks

```
/* Structural Styles */

article, aside, figcaption, figure,
footer, header, main, nav, section {
   display: block;
}
```

You will complete the reset style sheet by adding other style rules that set default padding and margins around commonly used page elements, define some basic typographic properties, and remove underlining from hypertext links found within navigation lists.

To complete the reset style sheet:

1. Within the Typographic Styles section, insert the following style rule to define the typographic styles for several page elements:

```
address, article, aside, blockquote, body, cite,
div, dl, dt, dd, em, figcaption, figure, footer,
h1, h2, h3, h4, h5, h6, header, html, img,
li, main, nav, ol, p, section, span, ul {

   background: transparent;
   font-size: 100%;
   margin: 0;
   padding: 0;
   vertical-align: baseline;
}
```

2. Add the following style rules to remove list markers from list items found within navigation lists:

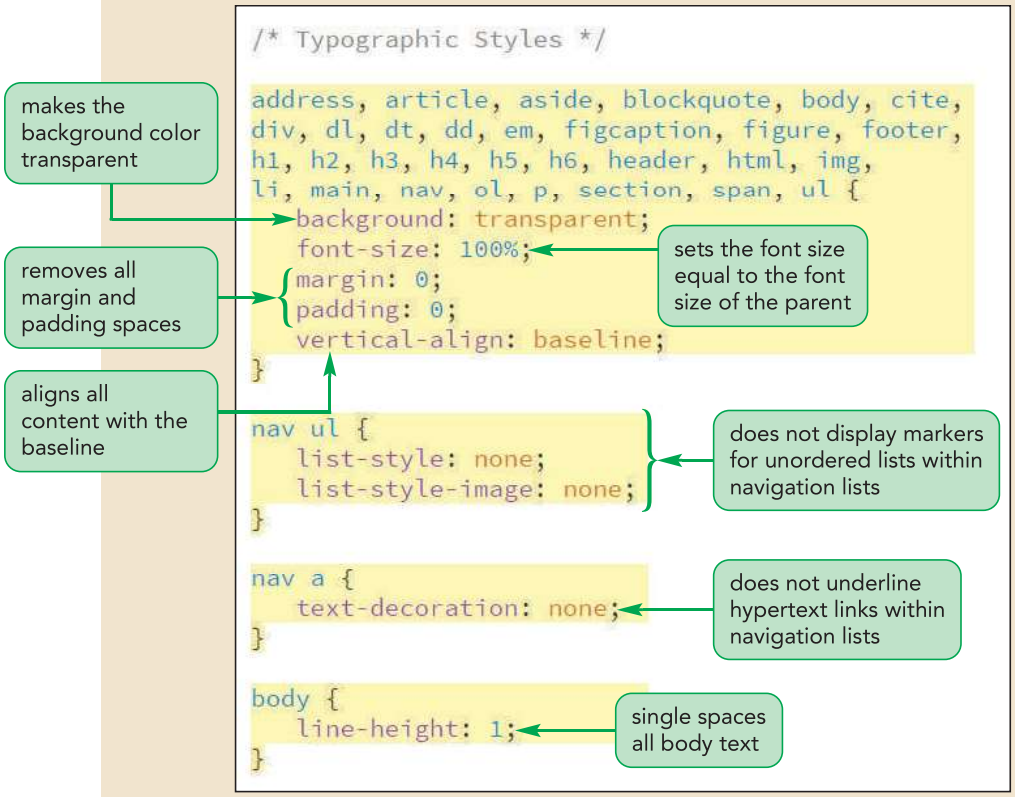
```
nav ul {
   list-style: none;
   list-style-image: none;
}

nav a {
   text-decoration: none;
}
```

3. Set the default line height to 1 (single-spaced) by applying the following style rule to the page body:

```
body {
   line-height: 1;
}
```

Figure 3–3 describes the new style rules in the document.

Figure 3–3 Completing the reset style sheet



4. Save your changes to the file.

This is a very basic reset style sheet. There are premade reset style sheets freely available on the web that contain more style rules used to reconcile the various differences between browsers and devices. Before using any of these reset style sheets, you should study the CSS code and make sure that it meets the needs of your website. Be aware that some reset style sheets may contain more style rules than you actually need and you can speed up your website by paring down the reset sheet to use only the elements you need for your website.

The first page you will work on for Pandaisia Chocolates is the site's home page. Anne has already created a typographical style sheet in the pc_styles1.css file. Link to the style sheet file now as well as the pc_reset.css style sheet you just created and the pc_home.css style sheet that you will work on for the remainder of this session to design the page layout.

To get started on the Pandaisia Chocolates home page:

1. Use your editor to open the **pc_home_txt.css** file from the html03 ▸ tutorial folder. Enter ***your name*** and ***the date*** in the comment section of the file and save the document as **pc_home.css**.
2. Use your editor to open the **pc_home_txt.html** file from the same folder. Enter ***your name*** and ***the date*** in the comment section and save the file as **pc_home.html**.
3. Within the document head, directly after the `title` element, insert the following `link` elements to link the home page to the pc_reset.css, pc_styles1.css and pc_home.css style sheets:

```
<link href="pc_reset.css" rel="stylesheet" />
<link href="pc_styles1.css" rel="stylesheet" />
<link href="pc_home.css" rel="stylesheet" />
```

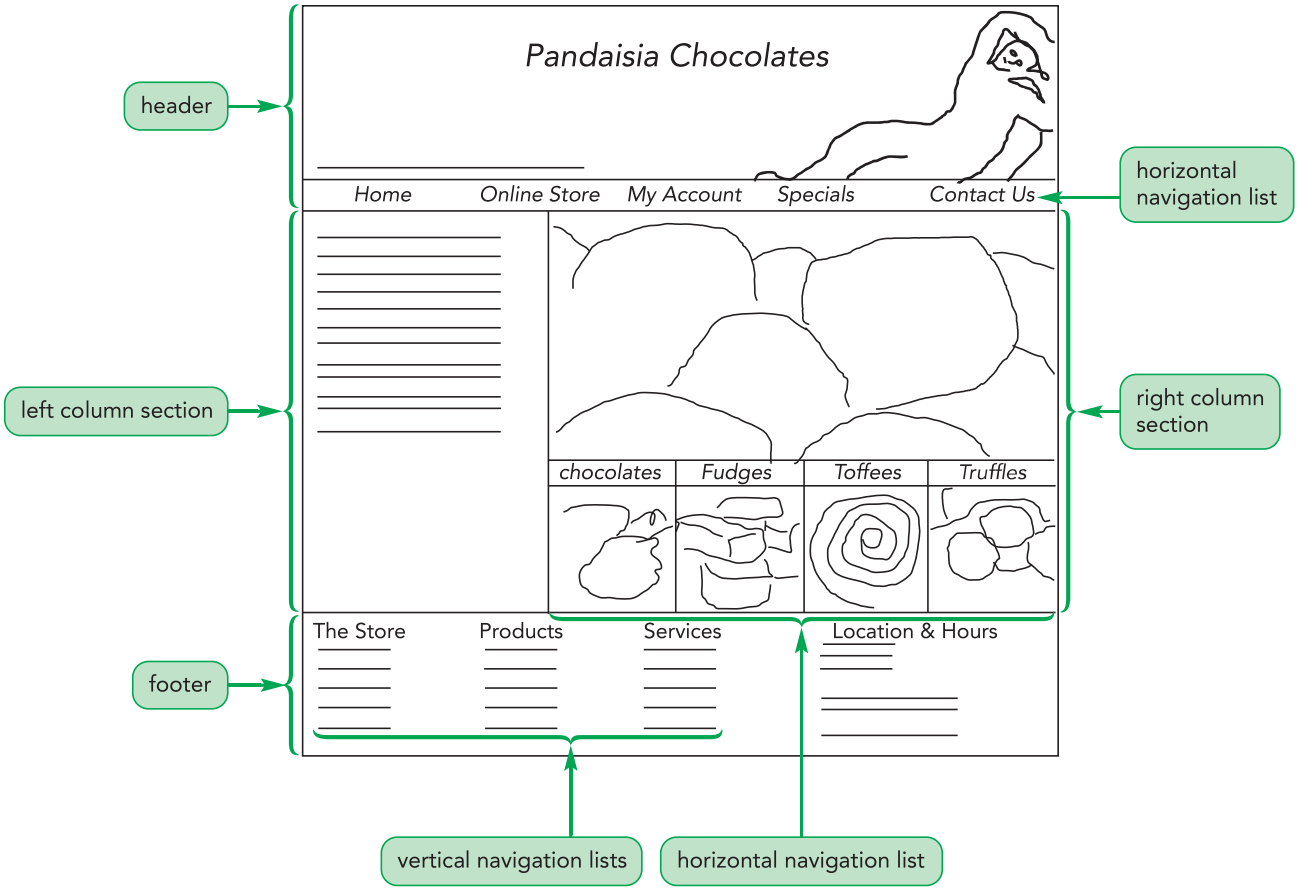
4. Take some time to study the content and structure of the pc_home.html document. Pay particular attention to the use of ID and class names throughout the document.
5. Save your changes to the file. You might want to keep this file open as you work with the pc_home.css style sheet so that you can refer to its content and structure.

TIP

The reset style sheet should *always* be the first style sheet listed before any other style sheets to ensure that your default styles are applied first.

Anne has sketched the general layout she wants for the home page, shown in Figure 3–4. Compare the pc_home.html file content to the sketch shown in Figure 3–4 to get a better understanding of how the page content relates to Anne's proposed layout.

Figure 3–4 **Proposed home page layout**



Before creating the page layout that Anne has sketched out for you, you'll examine different types of layout designs.

Exploring Page Layout Designs

One challenge of layout is that your document will be viewed on many different devices with different screen resolutions. When designing for the web, you're usually more concerned about the available screen width than screen height because users can scroll vertically down the length of the page, but it is considered bad design to make them scroll horizontally.

A page designer needs to cope with a wide range of possible screen widths ranging from wide screen monitors with widths of 1680 pixels or more, down to mobile devices with screen widths of 320 pixels and even less. Complicating matters even more is that a screen width represents the maximum space available to the user, but some space is always taken up by toolbars, sidebar panes, and other browser features. In addition, the user might not even have the browser window maximized to fill the entire screen. Thus, you need a layout plan that will accommodate a myriad of screen resolutions and browser configurations.

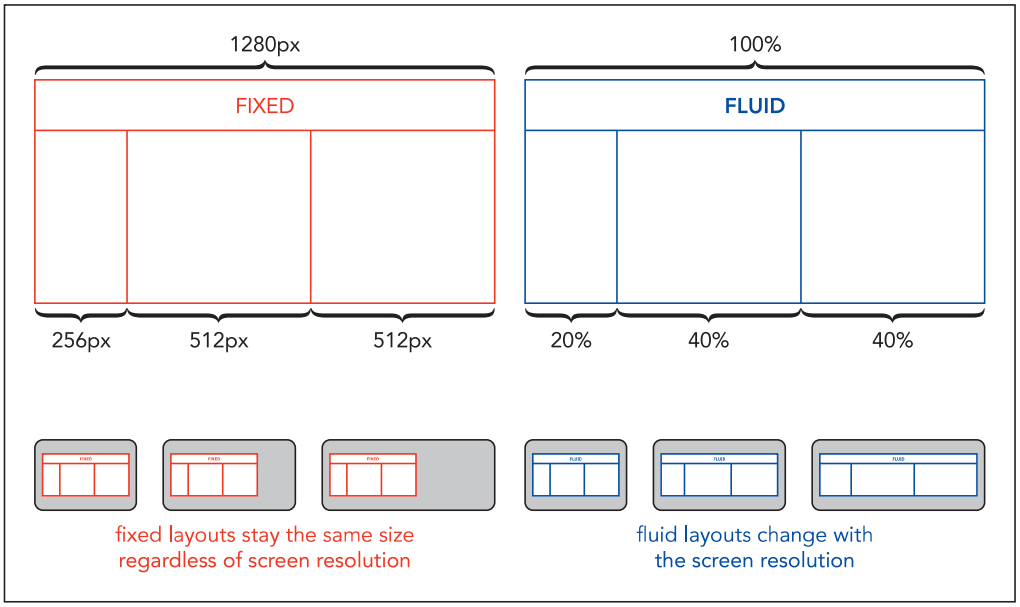
Fixed, Fluid, and Elastic Layouts

Web page layouts fall into three general categories: fixed, fluid, and elastic. A **fixed layout** is one in which the size of the page and the size of the page elements are fixed, usually using pixels as the unit of measure. The page width might be set at 960 pixels and the

width of the company logo set to 780 pixels. These widths are set regardless of the screen resolution of the user's device and this can result in the page not fitting into the browser window if the device's screen is not wide enough.

By contrast, a **fluid layout** sets the width of page elements as a percent of the available screen width. For example, the width of the page body might be set to fill 90% of the screen and the width of the company logo might be set to fill 80% of that page body. Under a fluid layout, the page resizes automatically to match the screen resolution of the user's device. Figure 3–5 shows how a three-column layout might appear in both a fixed and a fluid design.

Figure 3–5 **Fixed layouts vs. fluid layouts**



With different devices accessing your website, it's usually best to work with a fluid layout that is more adaptable to a range of screen resolutions. Fixed layouts should only be used when you have more control over the devices that will display your page, such as a web page created specifically for a digital kiosk at a conference.

Another layout design is an **elastic layout** in which all measurements are expressed in em units and based on the default font size used in the page. If a user or the designer increases the font size, then the width, height, and location of all of the other page elements, including images, change to match. Thus, images and text are always sized in proportion to each other and the layout never changes with different font sizes. The disadvantage to this approach is that, because sizing is based on the font size and not on the screen resolution, there is a danger that if a user sets the default font size large enough, the page will extend beyond the boundaries of the browser window.

Finally, the web is moving quickly toward the principles of **responsive design** in which the layout and design of the page change in response to the device that is rendering it. The page will have one set of styles for mobile devices, another for tablets, and yet another for laptops or desktop computers. You'll explore how to implement responsive design in Tutorial 5.

Because width is such an integral part of layout, you will start designing the Pandaisia Chocolates home page by defining the width of the page body and elements within the page.

Working with Width and Height

The width and height of an element are set using the following `width` and `height` properties

```
width: value;
height: value;
```

where *value* is the width or height using one of the CSS units of measurement or as a percentage of the width or height of the parent element. For example, the following style rule sets the width of the page body to 95% of the width of its parent element (the browser window):

```
body {width: 95%;}
```

Usually, you do not set the height value because browsers automatically increase the height of an element to match its content. Note that all block elements, like the `body` element, have a default width of 100%. Thus, this style rule makes the `body` element width slightly smaller than it would be by default.

Setting Maximum and Minimum Dimensions

You can set limits on the width or height of a block element by applying the following properties

```
min-width: value;
min-height: value;
max-width: value;
max-height: value;
```

where *value* is once again a length expressed in one of the CSS units of measure (usually pixels to match the measurement unit of the display device). For example, the following style rule sets the width of the page body to 95% of the browser window width but confined within a range of 640 to 1680 pixels:

```
body {
   width: 95%;
   min-width: 640px;
   max-width: 1680px;
}
```

Maximum and minimum widths are often used to make page text easier to read. Studies have shown that lines of text that are too wide are difficult to read because the eye has to scan across a long section of content and that lines of text that are too narrow with too many line returns break the flow of the material.

REFERENCE

Setting Widths and Heights

- To set the width and height of an element, use the styles

```
width: value;
height: value;
```

where *value* is the width or height in one of the CSS units of measurement or a percentage of the width or height of the parent element.

- To set the minimum possible width or height, use the styles

```
min-width: value;
min-height: value;
```

- To set the maximum possible width or height, use the styles

```
max-width: value;
max-height: value;
```

Set the width of the page body for the Pandaisia Chocolates home page to 95% of the browser window ranging from 640 pixels to 960 pixels. Also display the company logo image as a block with its width set to 100% so that it extends across the page body. You do not have to set the height of the logo because the browser will automatically scale the height to keep the original proportions of the image.

To set the initial dimensions of the page:

1. Return to the **pc_home.css** file in your editor and add the following style rule to the Body Styles section:

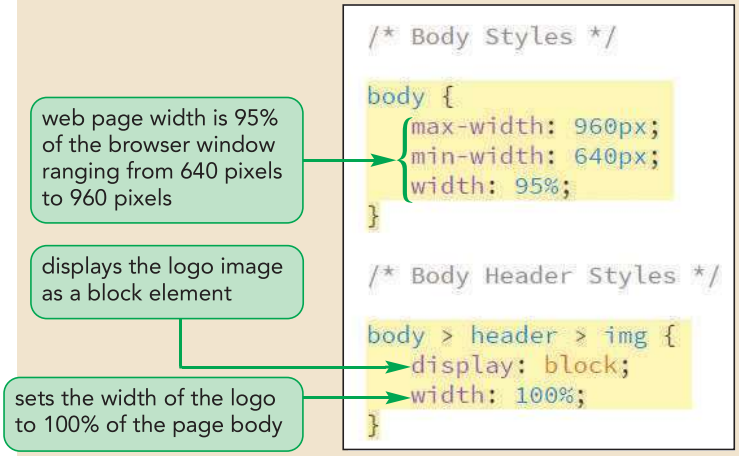
```
body {
   max-width: 960px;
   min-width: 640px;
   width: 95%;
}
```

2. Within the Body Header Styles section, insert the following style rule to set the display type and width of the logo image:

```
body > header > img {
   display: block;
   width: 100%;
}
```

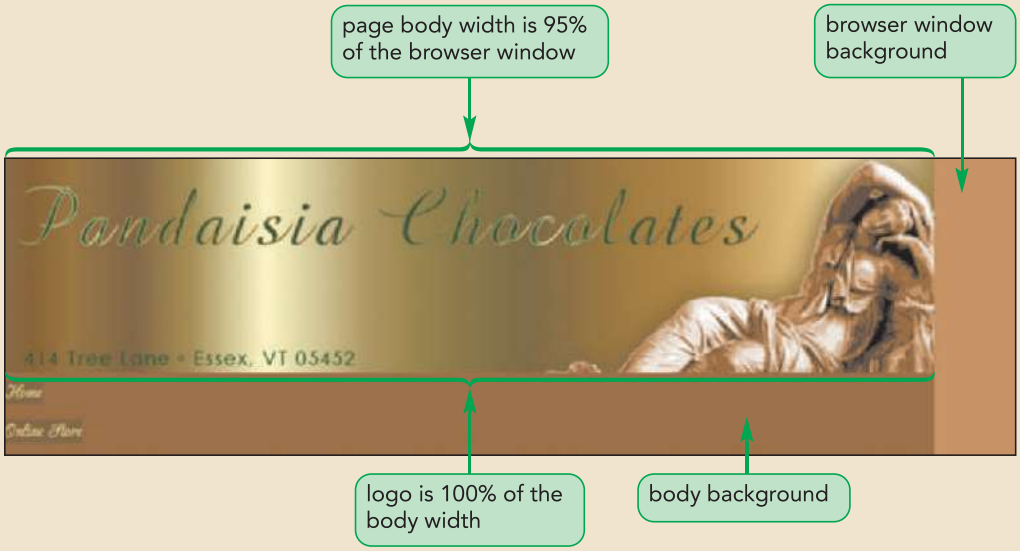
Figure 3–6 highlights the newly added style rules in the style sheet.

Figure 3–6 Setting the width of the page body and logo



3. Save your changes to the file and then open the **pc_home.html** file in your browser. Figure 3–7 shows the current layout of the page body and logo.

Figure 3–7 Initial view of the body header



4. Change the width of your browser window and verify that the size of the page body and the size of the logo resize as needed within the range of 640 to 960 pixels.

The page body is currently placed on the left margin of the browser window. Anne would like it centered horizontally within the browser window.

Centering a Block Element

Block elements can be centered horizontally within their parent element by setting both the left and right margins to `auto`. Thus, you can center the page body within the browser window using the style rule:

```
body {
   margin-left: auto;
   margin-right: auto;
}
```

Modify the style rule for the page body to center the Pandaisia Chocolates home page horizontally by setting the left and right margins to `auto`.

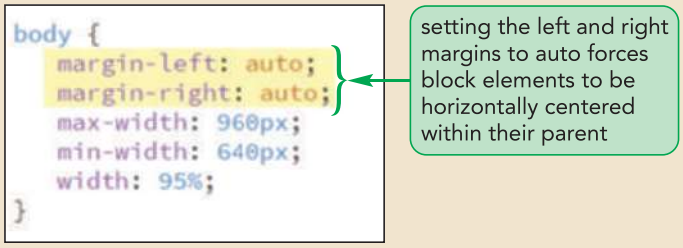
To center the page body horizontally:

1. Return to the **pc_home.css** file in your editor and, within the style rule for the `body` selector, insert the properties:

   ```
   margin-left: auto;
   margin-right: auto;
   ```

   Figure 3–8 highlights the newly added styles.

Figure 3–8 Centering the page body



2. Save your changes to the file and then reload the pc_home.html file in your browser. Verify that the page body is now centered within the browser window.

INSIGHT

Working with Element Heights

The fact that an element's height is based on its content can cause some confusion. For example, the following style rule appears to set the height of the header to 50% of the height of the page body:

```
body > header {height: 50%;}
```

However, because the total height of the page body depends on the height of its individual elements, including the body header, there is circular reasoning in this style rule. You can't set the page body height without knowing the height of the body header and you can't set the body header height unless you know the height of the page body. Most browsers deal with this circularity by leaving the body header height undefined, resulting in no change in the layout.

Heights need to be based on known values, as in the following style rules where the body height is set to 1200 pixels and thus the body header is set to half of that or 600 pixels.

```
body {height: 1200px;}
body > header {height: 50%;}
```

It is common in page layout design to extend the page body to the height of the browser window. To accomplish this, you set the height of the `html` element to 100% so that it matches the browser window height (a known value defined by the physical properties of the screen) and then you set the minimum height of the page body to 100% as in the following style rules:

```
html {height: 100%;}
body {min-height: 100%;}
```

The result is that the height of the page body will always be at least equal to the height of the browser window, but it will extend beyond that if necessary to accommodate extra page content.

Vertical Centering

Centering an element vertically within its parent element is not easily accomplished because the height of the parent element is usually determined by its content, which might not be a defined value. One solution is to display the parent element as a table cell with a defined height and then set the `vertical-align` property set to `middle`. For example, to vertically center the following h1 heading within the `div` element

```
<div>
   <h1>Pandaisia Chocolates</h1>
</div>
```

you would apply the style rule:

```
div {
   height: 40px;
   display: table-cell;
   vertical-align: middle;
}
```

Using this style rule, the h1 heading will be vertically centered.

To vertically center a single line of text within its parent element, set the line height of the text larger than the text's font size. The following style rule will result in an h1 heading with vertically centered heading text.

```
h1 {
   font-size: 1.4em;
   line-height: 2em;
}
```

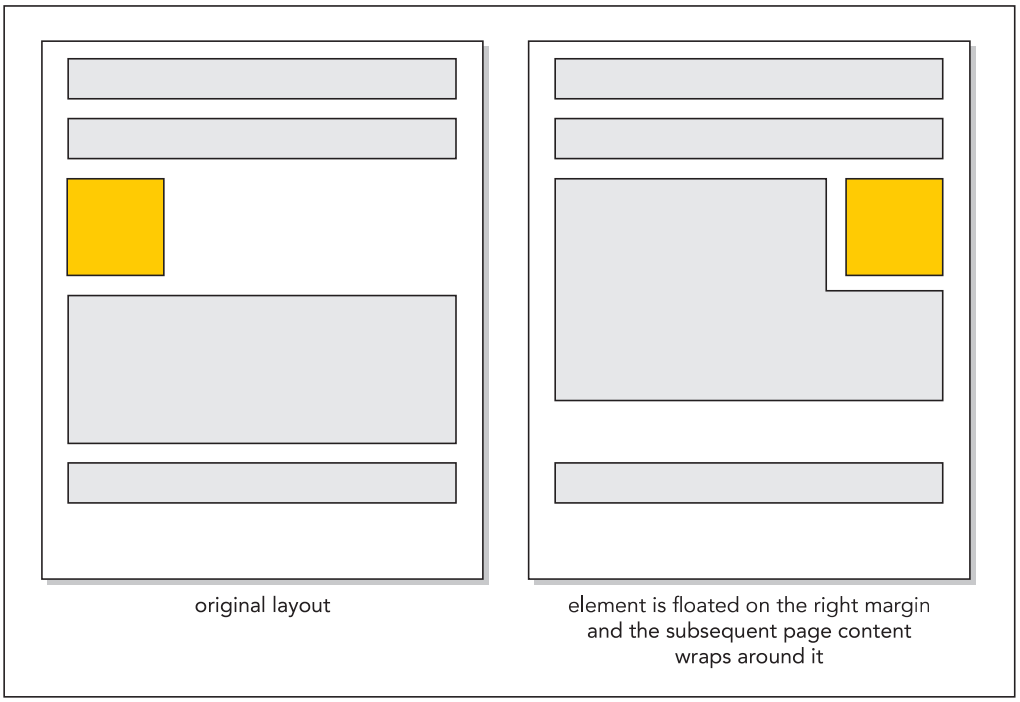
Note that this approach will only work for a single line of text. If the text wraps to a second line, it will no longer be vertically centered. Vertical centering is a common design challenge and there are several other workarounds that have been devised over the years. The simplest approach is to use CSS grid styles, a topic that we'll discuss in the next session.

Next, you will lay out the links in the navigation list. Anne wants the links displayed horizontally rather than vertically. You can accomplish this using CSS floats.

Floating Page Content

By default, content is displayed in the page in the order it appears within the HTML file as part of the normal document flow. **Floating** an element takes it out of position and places it along the left or right edge of its parent element. Subsequent content that is not floated occupies the space previously taken up by the floated element. Figure 3–9 shows a diagram of an element that is floated along the right margin of its container and its effect on the placement of subsequent content.

Figure 3–9 Floating an element

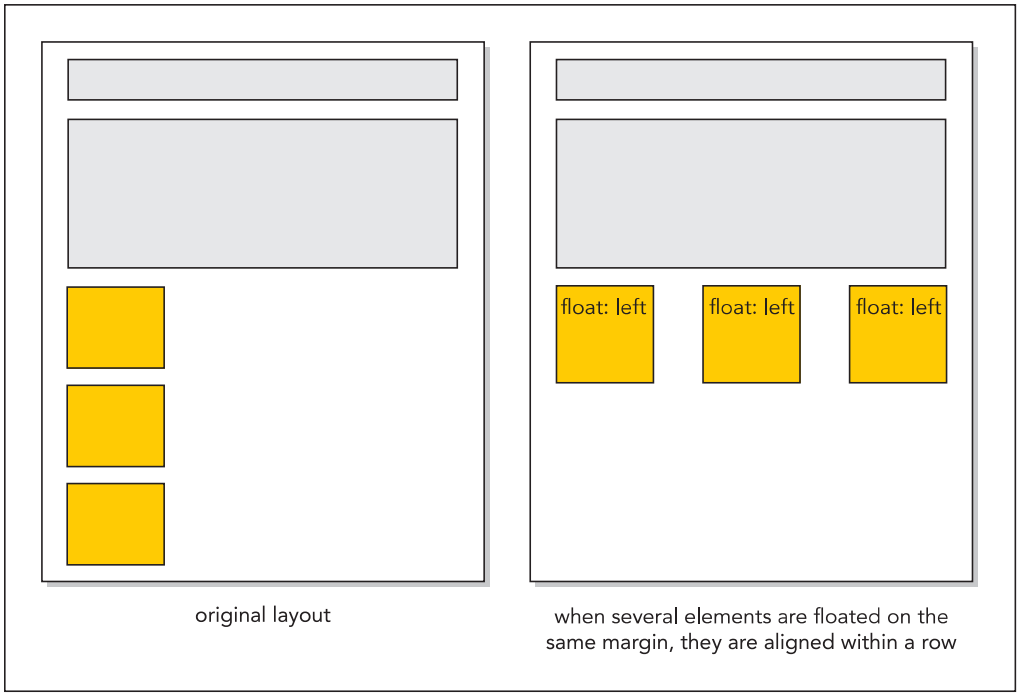


To float an element, apply the following `float` property

```
float: position;
```

where *position* is `none` (the default), `left` to float the object on the left margin, or `right` to float the object on the right margin. If sibling elements are floated along the same margin, they are placed alongside each other within a row as shown in Figure 3–10.

Figure 3–10 **Floating multiple elements in a row**



Note that for the elements to be placed within a single row, the combined width of the elements cannot exceed the total width of their parent element, otherwise any excess content will automatically wrap to a new row.

REFERENCE

Floating an Element

- To float an element within its container, apply the style

  `float: position;`

  where *position* is `none` (the default), `left`, or `right`.

Anne wants you display the content of navigation lists belonging to the horizontalNavigation class within a single row. You will accomplish this by floating each item in those navigation lists on the left margin using the `float` property. Create this style rule now.

To be confined to a single row, the total width of floated elements cannot exceed the width of the container.

To lay out horizontal navigation list items:

1. Return to the **pc_home.css** file in your editor and go to the Body Header Styles section.
2. Because there are five links in the navigation list, you'll make each list item 20% of the width of the navigation list by adding the following style rule:

```
body > header > nav.horizontalNavigation li {
   width: 20%;
}
```

3. Insert the following style rule within the Horizontal Navigation Styles section to display every list item within a horizontal navigation list as a block floated on the left.

```
nav.horizontalNavigation li {
   display: block;
   float: left;
}
```

Figure 3–11 highlights the styles used with list items.

Figure 3–11 Floating items in the navigation list

```
/* Body Header Styles */

body > header > img {
   display: block;
   width: 100%;
}

body > header > nav.horizontalNavigation li {
   width: 20%;
}

/* Horizontal Navigation Styles */

nav.horizontalNavigation li {
   display: block;
   float: left;
}
```

sets the width of the list item to 20% of the width of the navigation list

floats the list item within every horizontal navigation list as a block on the left

4. Save your changes to the file and then reload the pc_home.html file in your browser. Figure 3–12 shows the revised layout of the navigation list in the page header.

Figure 3–12 Floating items in a horizontal navigation list

the width of each list item set to 20% and floated on the left margin

Anne doesn't like the appearance of the hypertext links in the navigation list. Because the links are inline elements, the background color extends only as far as the link text. She suggests you change the links to block elements and center the link text within each block.

To change the display of the hypertext links:

1. Return to the **pc_home.css** file in your editor.
2. Within the Horizontal Navigation Styles section, insert the following style rule to format the appearance of the hypertext links within the horizontal navigation lists:

```
nav.horizontalNavigation a {
   display: block;
   text-align: center;
}
```

Figure 3–13 highlights the style rule for the hypertext links.

Figure 3–13 Formatting hyperlinks in horizontal navigation lists

```
/* Horizontal Navigation Styles */

nav.horizontalNavigation li {
   display: block;
   float: left;
}

nav.horizontalNavigation a {
   display: block;
   text-align: center;
}
```

centers the link text within the block

displays the link as a block

3. Save your changes to the file and then reload the pc_home.html file in your browser.
4. Hover your mouse pointer over the links in the navigation list. Note that the link text is centered within its block and the background color extends fully across the block rather than confined to the link text. See Figure 3–14.

Figure 3–14 Links in the body header

each hypertext link displayed as a block with the link text centered within the block

Trouble? Don't worry about the jumble of elements displayed after the body header. You'll straighten out those objects next.

You have completed the design of the body header. Next, you will lay out the middle section of the home page.

INSIGHT

Creating Drop Caps with CSS

A popular design element is the **drop cap**, which consists of an enlarged initial letter that drops down into a body of text. To create a drop cap, you increase the font size of an element's first letter and float it on the left margin. Drop caps also generally look better if you decrease the line height of the first letter, enabling the surrounding content to better wrap around the letter. Finding the best combination of font size and line height is a matter of trial and error, and unfortunately, what looks best in one browser might not look as good in another. The following style rule works well in applying a drop cap to the first paragraph element:

```
p:first-of-type::first-letter {
   font-size: 4em;
   float: left;
   line-height: 0.8;
}
```

For additional design effects, you can change the font face of the drop cap to a cursive or decorative font.

Clearing a Float

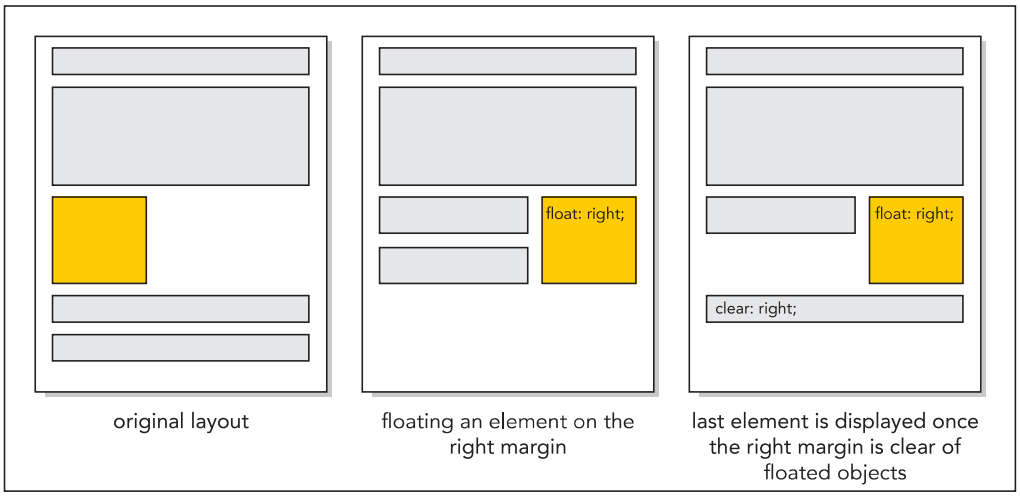
In some layouts, you will want an element to be displayed on a new row, clear of previously floated objects. To ensure that an element is always displayed below your floated elements, apply the following `clear` property:

```
clear: position;
```

where *position* is `left`, `right`, `both`, or `none`. A value of `left` displays the element only when the left margin is clear of floating objects. A value of `right` displays the element only when the right margin is clear. A value of `both` displays the element only when both margins are clear of floats. The default clear value is `none`, which allows the element to be displayed alongside any floated objects.

Figure 3–15 shows how use of the `clear` property prevents an element from being displayed until the right margin is clear of floats. The effect on the page layout is that the element is shifted down and is free to use the entire page width since it is no longer displayed alongside a floating object.

Figure 3–15 Clearing a float



REFERENCE

Clearing a Float

- To display a non-floated element on a page with a floated element, use the following style so the non-floated element can clear the floated element

  `clear: position;`

  where *position* is `none` (the default), `left`, `right`, or `both`.

The next part of the Pandaisia Chocolates home page contains two `section` elements named leftColumn and rightColumn. Set the width of the left column to 33% of the body width and set the width of the right column to 67%. Float the sections side-by-side on the left margin, but only when the left margin is clear of all previously floated objects.

To float the left and right column sections:

1. Return to the **pc_home.css** file in your editor. Go to the Left Column Styles section and insert the style rule:

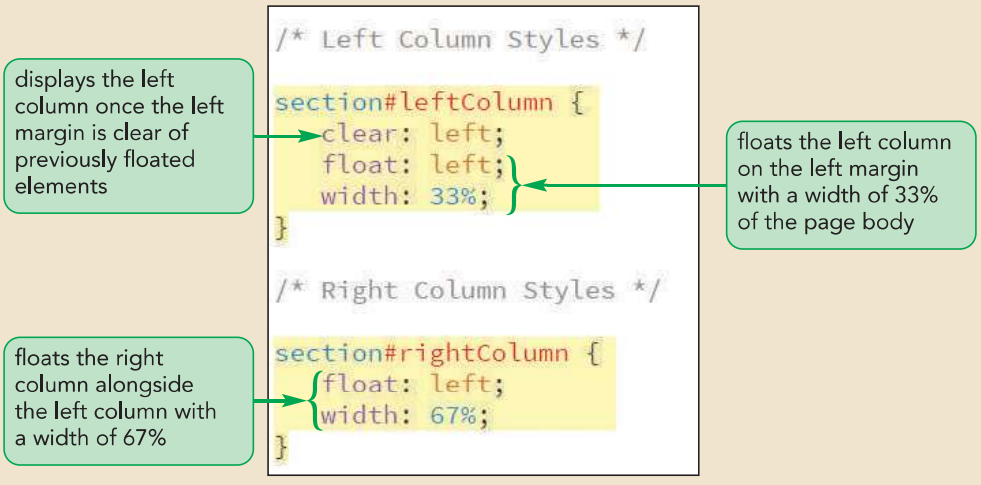
```
section#leftColumn {
   clear: left;
   float: left;
   width: 33%;
}
```

2. Within the Right Column Styles section, insert:

```
section#rightColumn {
   float: left;
   width: 67%;
}
```

Note that you do not apply the `clear` property to the right column because you want it to be displayed in the same row alongside the left column. Figure 3–16 highlights the style rules for the left and right columns.

Figure 3–16 Float the left and right column sections



The right column contains a horizontal navigation list containing four items, each consisting of an image and a label above the image. Anne wants the four items placed side-by-side with their widths set to 25% of the width of the navigation list. Anne also wants the images in the right column displayed as blocks with their widths set to 100% of their parent element.

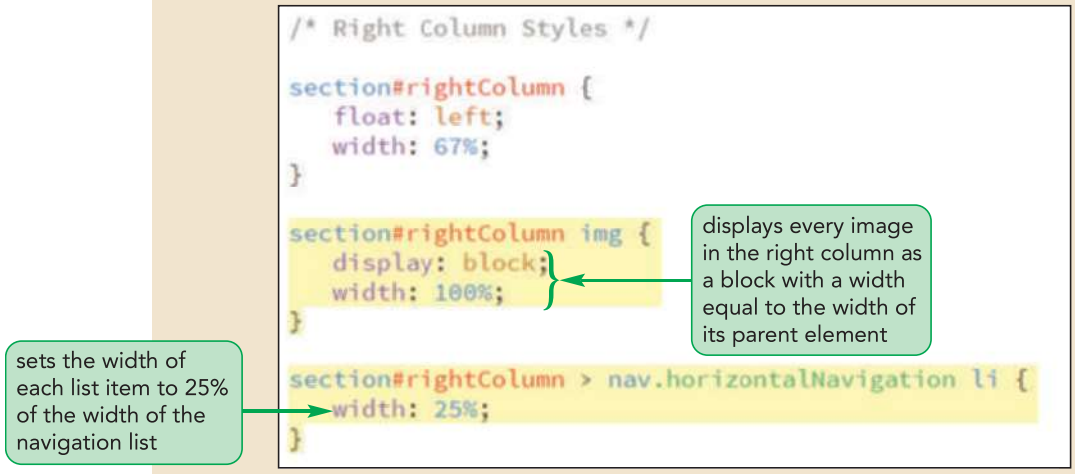
To complete the right column section:

1. Within the Right Column Styles section, insert the following style rules to format the inline images and list items:

```
section#rightColumn img {
   display: block;
   width: 100%;
}

section#rightColumn > nav.horizontalNavigation li {
   width: 25%;
}
```

Note that you do not have to include a style rule to float the items in the horizontal navigation list because you have already created that style rule in Figure 3–11. Figure 3–17 describes the new style rules in the style sheet.

Figure 3–17 Formatting the right column section



2. Save your changes to the file and then reload the pc_home.html file in your browser. Figure 3–18 shows the layout of the left and right column sections.

Figure 3–18 Layout of the left and right columns

Anne doesn't like that the text in the left column crowds the right column and page boundary. She suggests that you provide more interior space by increasing the padding in the left column.

To increase the left column padding:

1. Return to the **pc_home.css** file in your editor and go to the Left Column Styles section.
2. Insert the property **padding: 1.5em;** into the `section#leftColumn` style rule as shown in Figure 3–19.

Figure 3–19 Increasing the padding of the left column

```
/* Left Column Styles */

section#leftColumn {
   clear: left;
   float: left;
   padding: 1.5em;
   width: 33%;
}
```

increases the interior padding to 1.5em

3. Save your changes to the style sheet and then reload the pc_home.html file in your browser. Figure 3–20 shows the result of your change.

Figure 3–20 Page layout crashes with increased padding

© Brenda Carson/Shutterstock.com; Source: Facebook; Source: Twitter, Inc.

This simple change has caused the layout to crash. What went wrong?

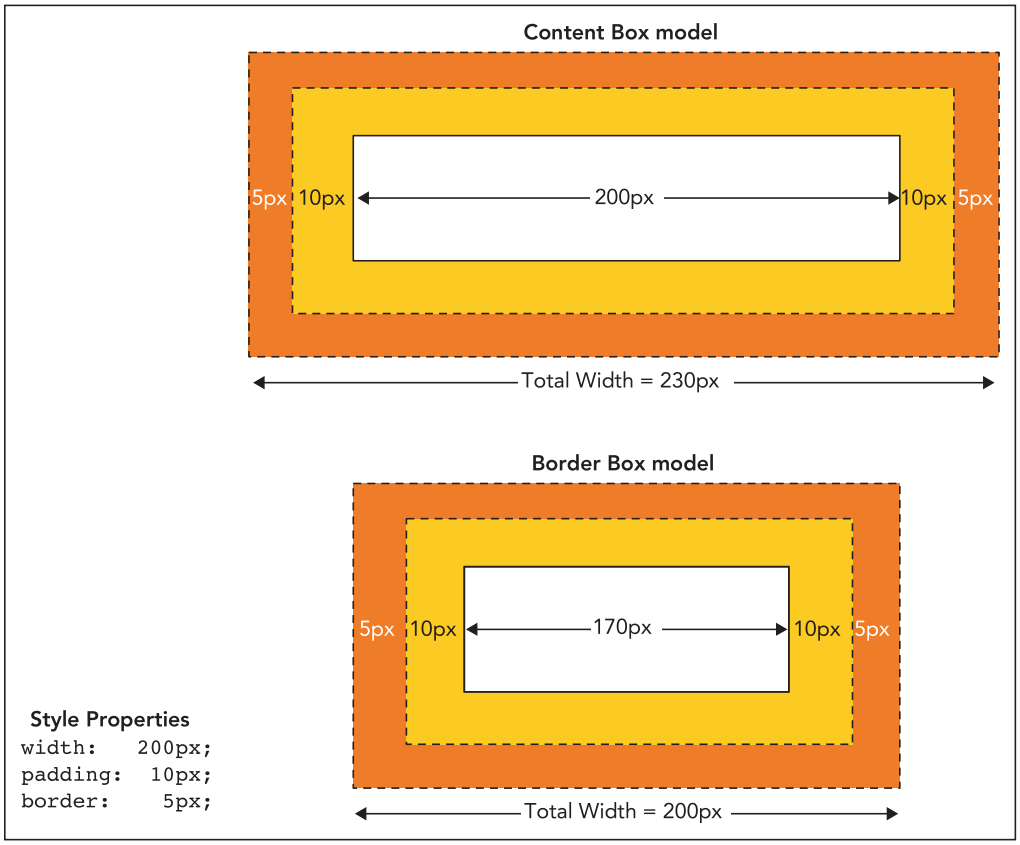
Refining a Floated Layout

When the total width of floated objects exceeds the width of their parent, excess content is automatically wrapped to a new row. The reason the layout for the Pandaisia Chocolates home page crashed is that increasing the padding in the left column, increased the column's width beyond its set value of 33%. Even this small increase caused the total width of the two columns to exceed 100% and, as a result, the right column moved to a new row.

To keep floats within the same row, you have to understand how CSS handles widths. Recall that block elements are laid out according to the box model, as illustrated previously in Figure 2–38, in which the content is surrounded by the padding space, the border space, and finally the margin space. By default, browsers measure widths using the **content box model** in which the `width` property only refers to the width of the element content and any padding or borders constitute added space.

CSS also supports the **border box model**, in which the `width` property is based on the sum of the content, padding, and border spaces and any space taken up by the padding and border is subtracted from space given to the content. Figure 3–21 shows how the two different models interpret the same width, padding, and border values.

Figure 3–21 Comparing the content box and border box models



TIP

Height values are similarly affected by the type of layout model used.

You can choose the layout model using the following `box-sizing` property

```
box-sizing: type;
```

where *type* is `content-box` (the default), `border-box`, or `inherit` (to inherit the property defined for the element's container). Many designers prefer to use the border box model in page layout so that there is no confusion about the total width of each element.

REFERENCE

Defining How Widths Are Interpreted

- To define what the `width` property measures, use the style:

  ```
  box-sizing: type;
  ```

  where *type* is `content-box` (the default), `border-box`, or `inherit` (to inherit the property defined for the element's container).

Add the `box-sizing` property to the reset style sheet and apply it to all block elements.

To set the block layout model:

1. Return to the **pc_reset.css** file in your editor.
2. Add the following style property to the style rule for the list of block elements:

```
box-sizing: border-box;
```

Figure 3–22 highlights the revised style rule.

Figure 3–22 Adding the border-box style to the reset style sheet

applies border-box sizing to all of the listed block elements

```
address, article, aside, blockquote, body, cite,
div, dl, dt, dd, em, figcaption, figure, footer,
h1, h2, h3, h4, h5, h6, header, html, img,
li, main, nav, ol, p, section, span, ul {
   background: transparent;
   font-size: 100%;
   margin: 0;
   padding: 0;
   vertical-align: baseline;
   box-sizing: border-box;
}
```

3. Save your changes to the style sheet and then reload the pc_home.html file in your browser. Verify that the layout of the left and right columns has been restored and additional padding has been added within the left column.

The final part of the Pandaisia Chocolates home page is the footer, which contains three vertical navigation lists and a `section` element with contact information for the store. Once the left margin is clear of previously floated objects, float these four elements on the left margin with the widths of the three navigation lists each set to 22% of the body width and the `section` element occupying the remaining 34%.

To lay out the page footer:

1. Return to the **pc_home.css** file in your editor and scroll down to the Footer Styles section.
2. Insert the following style rules:

```
footer {
   clear: left;
}

footer > nav.verticalNavigation {
   float: left;
   width: 22%;
}

footer > section#contactInfo {
   float: left;
   width: 34%;
}
```

Figure 3–23 highlights the layout style rules for the page footer.

Figure 3–23 Setting the layout of the page footer

displays the footer once the left margin is clear of floated objects

sets the width of the verticalNavigation lists to 22% and floats them on the left

sets the width of the contactInfo section to 34% and floats it on the left

```
/* Footer Styles */

footer {
   clear: left;
}

footer > nav.verticalNavigation {
   float: left;
   width: 22%;
}

footer > section#contactInfo {
   float: left;
   width: 34%;
}
```

3. Save your changes to the style sheet and then reload pc_home.html in your browser. Figure 3–24 shows the new layout of the footer.

Figure 3–24 Page footer layout

each vertical navigation list set at 22% of the footer width and floated on the left

width of the contactInfo section set at 34% of the footer width and floated on the left

Anne asks you to change the background color of the footer to a dark brown to better show the text content.

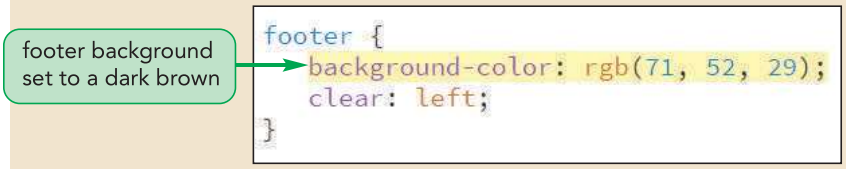
To set the footer background color:

1. Return to the **pc_home.css** file in your editor and go to the Footer Styles section.
2. Insert the following property for the `footer` selector:

```
background-color: rgb(71, 52, 29);
```

Figure 3–25 highlights the footer background color style.

Figure 3–25 Setting the footer background color



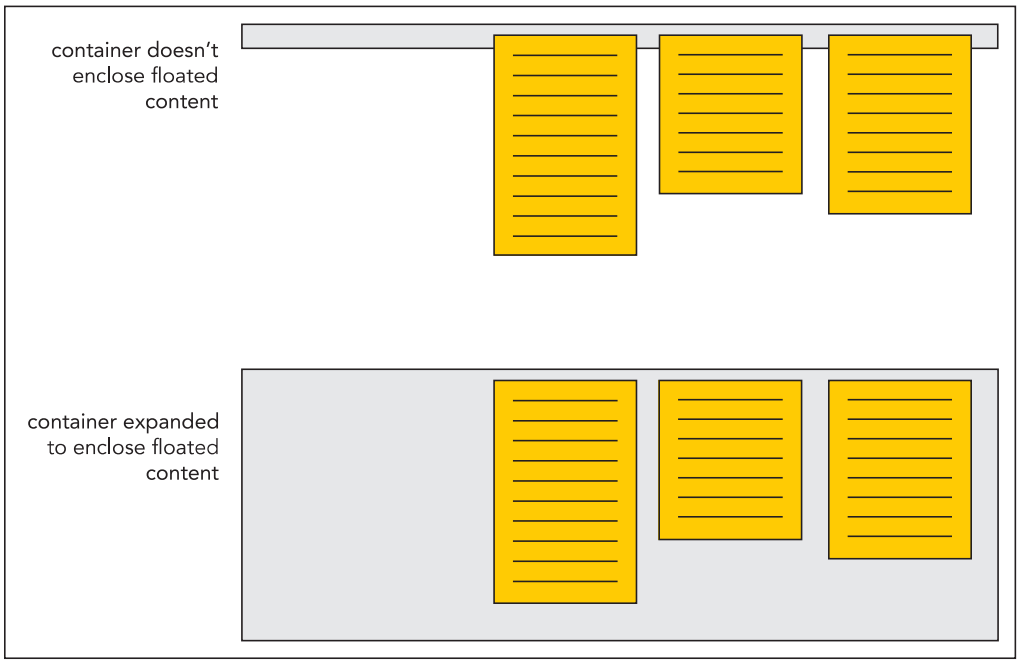
3. Save your changes to the style sheet and then reload pc_home.html in your browser. Note that the background color is *not changed*.

Why didn't the change to the background color take effect? To help you understand why, you'll look once again at the nature of floated elements.

Working with Container Collapse

Recall that a floated element is taken out of the document flow so that it is no longer "part" of the element that contains it. Literally it is floating free of its container. When every element in a container is floated, there is no content left. As far as the browser is concerned, the container is empty and thus has no height and no background to color, a situation known as **container collapse**. Figure 3–26 demonstrates container collapse for a container that has three floating objects that exceed the boundaries of their container.

Figure 3–26 Container collapse



What you usually want in your layout is to have the container expand to surround all of its floating content. One way this can occur is if the container is followed by another element that is displayed only when the margins are clear of floats. In that situation, the container's height will expand up to that trailing element and in the process surround its floating content.

The problem with the footer in the Pandaisia home page is that there is no trailing element—the footer is the last element in the page body. One way to fix that problem is to use the `after` pseudo-element to add a placeholder element after the footer. The general style rule is

```
container::after {
   clear: both;
   content: "";
   display: table;
}
```

TIP

To find other ways to prevent container collapse, search the web using the keywords *CSS clearfix*.

where *container* is the selector for the element containing floating objects. The `clear` property keeps this placeholder element from being inserted until both margins are clear of floats. The element itself is a web table but contains only an empty text string so that no actual content is written to the web page. That's okay because the mere presence of this placeholder element is enough to keep the container from collapsing.

Add a style rule now to create a placeholder element that keeps the footer from collapsing around its floating content.

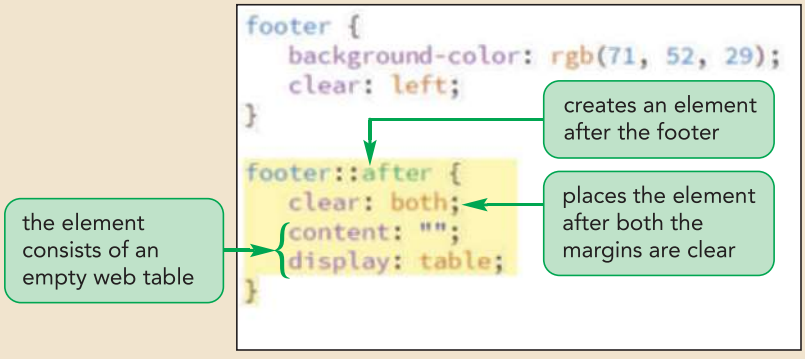
To keep the footer from collapsing:

1. Return to Footer Styles section in the **pc_home.css** file and, after the style rule for the footer element, insert the following rule:

```
footer::after {
   clear: both;
   content: "";
   display: table;
}
```

Figure 3–27 highlights the new rule in the style sheet.

Figure 3–27 Preventing the footer from collapsing



2. Save your changes to the style sheet and then reload pc_home.html in your browser. Figure 3–28 shows the completed layout of the Pandaisia Chocolates home page.

Figure 3–28 Final layout of the Pandaisia Chocolates home page

footer has expanded to contain all floated content

Note that the footer now has a dark brown background because it has expanded in height to contain all of its floated content.

3. Close any of the documents you opened for this session.

REFERENCE

Keeping a Container from Collapsing

- To prevent a container from collapsing around its floating content, add the following style rule to the container

```
container::after {
  clear: both;
  content: "";
  display: table;
}
```

where *container* is the selector for the element containing the floating content.

PROSKILLS

Problem Solving: The Virtue of Being Negative

It's common to think of layout in terms of placing content, but good layout also must be concerned with placing emptiness. In art and page design, this is known as working with positive and negative space. Positive space is the part of the page occupied by text, graphics, borders, icons, and other page elements. Negative space, or white space, is the unoccupied area and provides balance and contrast to elements contained in positive space.

A page that is packed with content leaves the eye with no place to rest; which also means that the eye has no place to focus and maybe even no clear indication about where to start reading. Negative space is used to direct users to resting stops before moving on to the next piece of page content. This can be done by providing a generous margin between page elements and by increasing the padding within an element. Even increasing the spacing between letters within an article heading can alleviate eye strain and make the text easier to read.

White space also has an emotional aspect. In the early days of print advertising, white space was seen as wasted space, and thus, smaller magazines and direct mail advertisements would tend to crowd content together in order to reduce waste. By contrast, upscale magazines and papers could distinguish themselves from those publications with an excess of empty space. This difference carries over to the web, where a page with less content and more white space often feels more classy and polished, while a page crammed with a lot of content feels more commercial. Both can be effective; you should decide which approach to use based on your customer profile.

You've completed your work on the Pandaisia Chocolates home page. In the next session, you'll work on page layout using the technique of grids.

REVIEW

Session 3.1 Quick Check

1. To display an element as a block-level use:
   a. `display: block-level;`
   b. `display: block;`
   c. `display: inline;`
   d. `display: display-block;`
2. What are three types of layouts?
   a. inline, fluid, static
   b. fixed, floating, static
   c. fixed, fluid, elastic
   d. inline, block, scrolling
3. Provide a style rule to set the maximum width of an element to 960 pixels.
   a. `maximum-width: 960px;`
   b. `maxw: 960px;`
   c. `width: 960px;`
   d. `max-width: 960px;`

4. Provide a style rule to horizontally center a block element within its container with a top/bottom margin of 20 pixels.
   a. `margin: 20px center;`
   b. `margin: center 20px;`
   c. `margin: auto 20px;`
   d. `margin: 20px auto;`
5. Provide a style rule to place an object on the right margin of its container.
   a. `margin: right;`
   b. `text-align: right;`
   c. `float: right;`
   d. `padding: right;`
6. Provide a style rule to display an object only when all floating elements have cleared.
   a. `clear: float;`
   b. `clear: floats;`
   c. `clear: both;`
   d. `clear: all;`
7. Your layout has four floated elements in a row but unfortunately the last element has wrapped to a new line. What is the source of the layout mistake?
   a. The widths of the floated elements exceed the available width of their container.
   b. You cannot float more than one object within a row.
   c. You have to clear the first three floating object to make room for the fourth.
   d. You have to clear the fourth floating object to make room for the first three.
8. Provide a style rule to change the width property for the header element so that it measures the total width of the header content, padding, and border spaces.
   a. `box-sizing: border-box;`
   b. `box-sizing: content-box;`
   c. `box-sizing: all;`
   d. `box-sizing: complete;`
9. What causes container collapse?
   a. The width of the child elements exceeds the width of the container.
   b. The width of the container element is fixed at 0 pixels.
   c. The height of the container element is fixed at 0 pixels.
   d. All child elements are floating so that they are free of the container, leaving the container with no content.

Session 3.2 Visual Overview:

The **grid-template-columns** property establishes the size and number of grid columns

To create a grid container set the **display** property to **grid**.

A **fractional unit**, indicated by the unit abbreviation **fr**, expands or contracts to fill available space; these fractional units keep the columns widths in a 2:1 proportion.

The **grid-template-areas** property defines the areas in the grid

The **grid-column-gap** property sets the interior space between grid columns.

Use the **grid-row** and **grid-column** properties to place items at and across specified grid rows and columns.

Use the **grid-area** property to place items within grid areas.

```
body {
   display: grid;
   grid-template-columns: 2fr 1fr;
   grid-template-areas: "header  header"
                        "intro    faq"
                        "articles faq"
                        "footer   footer";
   grid-column-gap: 15px;
}

body > header {
   grid-row: 1;
   grid-column: 1/-1;
}

body > article {grid-area: intro;}
body > aside {grid-area: faq;}
body > section {grid-area: articles;}
body > footer {grid-area: footer;}

/* Grid Styles for Nested Grid */

section {
   display: grid;
   grid-template-columns: repeat(2, 1fr);
}

section > h1 {
   grid-area: 1/1/2/3;
}
```

CSS Grid Layouts

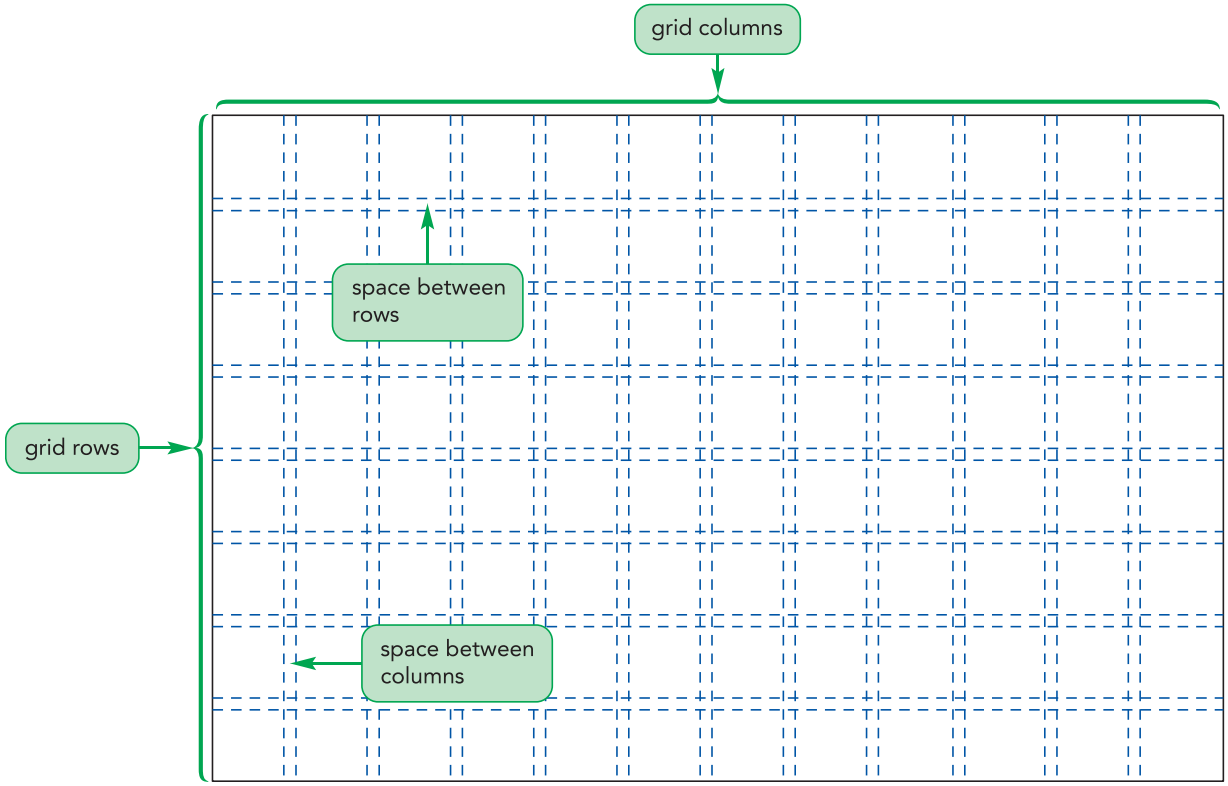
Introducing Grid Layouts

In the previous session, you used the `float` property to lay out a page in sections that floated alongside each other like columns. In this session, you'll explore how to generalize this technique by creating a page layout based on a grid.

Overview of Grid-Based Layouts

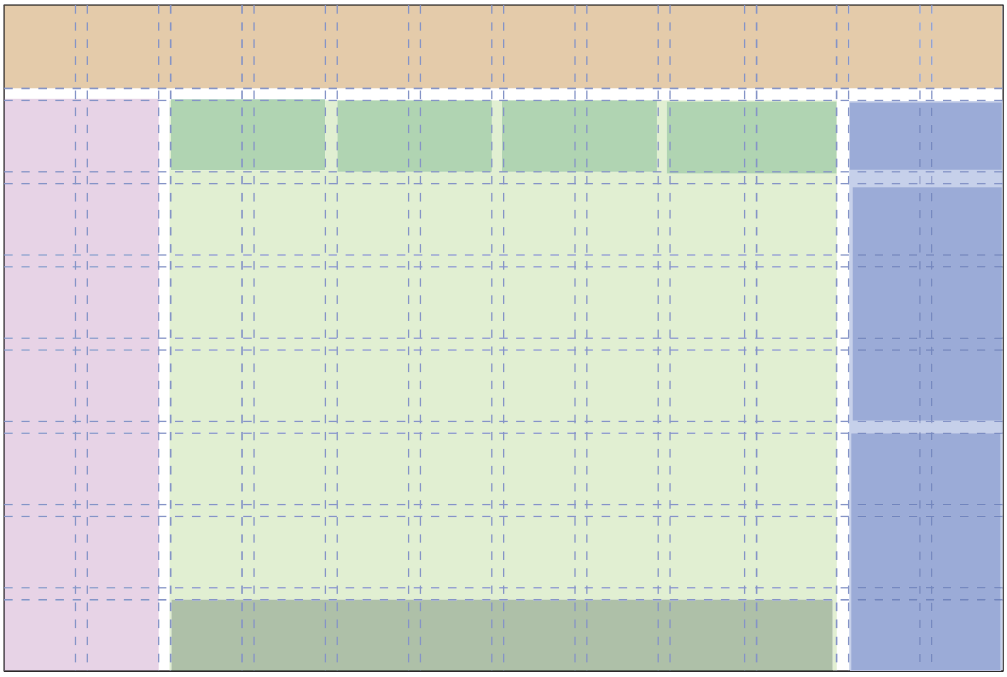
Grids are a classic layout technique that has been used in publishing for hundreds of years and, like many other publishing techniques, can be applied to web design. In a **grid layout**, the page is comprised of a system of intersecting rows and columns that form a grid. The rows are based on the page content. A long page with several articles might span several rows, or it could be a home page with introductory content that fits within a single row. The number of columns is based on the number that provides the most flexibility in laying out the page content. Many grid systems are based on 12 columns because 12 is evenly divisible by 2, 3, 4, and 6, but other sizes are also used. Figure 3–29 shows a 12-column grid layout.

Figure 3–29 **Page grid**



The page designer then arranges the page elements within the chosen grid. Figure 3–30 shows one possible layout comprised of a main header element (the tan area), three major sections (the lavender, light green, and blue areas), as well as a navigation bar and a footer (the dark green areas). Some sections (like the dark green and blue areas) are further divided into small subsections.

Figure 3–30 Layout based on a grid



It should be stressed that the grid is not part of the web page content. Instead, it's a systematic approach to visualizing how to best fit content onto the page. Working from a grid has several aesthetic and practical advantages, including

- Grids add order to the presentation of page content, adding visual rhythm, which is pleasing to the eye.
- A consistent logical design gives readers the confidence to find the information they seek.
- New content can be easily placed within a grid in a way that is consistent with previously entered information.
- A well designed grid is more easily accessible for users with disabilities and special needs.
- Grids speed up the development process by establishing a systematic framework for the page layout.

There are two basic types of grid layouts: fixed grids and fluid grids.

Fixed and Fluid Grids

In a **fixed grid**, the widths of the columns and margins are specified in pixels, where every column has a fixed position. Many fixed grid layouts are based on a page width of 960 pixels because most desktop screen widths are at 1024 pixels (or higher) and a 960-pixel width leaves room for browser scrollbars and other features. The 960-pixel width is also easily divisible into halves, thirds, quarters, and so forth, making it easier to create evenly spaced columns.

The problem of course with a fixed grid layout is that it does not account for other screen sizes and thus, a **fluid grid**, in which column widths are expressed in percentages rather than pixels, is often used to provide more support across different devices. In the examples to follow, you'll base your layouts on a fluid grid system.

Grids are often used with responsive design in which one grid layout is used with mobile devices, another grid layout is used with tablets, and yet another layout is used with desktop computers. A layout for a mobile device is typically based on a 1-column grid, tablet layouts are based on grids of 4 to 12 columns, and desktop layouts are often based on layouts with 12 or more columns.

CSS Frameworks

Designing your own grids can be time-consuming. To simplify the process, you can choose from the many CSS frameworks available on the web. A **framework** is a software package that provides a library of tools to design your website, including style sheets for grid layouts and built-in scripts to provide support for a variety of browsers and devices. Most frameworks include support for responsive design so that you can easily scale your website for devices ranging from mobile phones to desktop computers.

Some popular CSS frameworks include

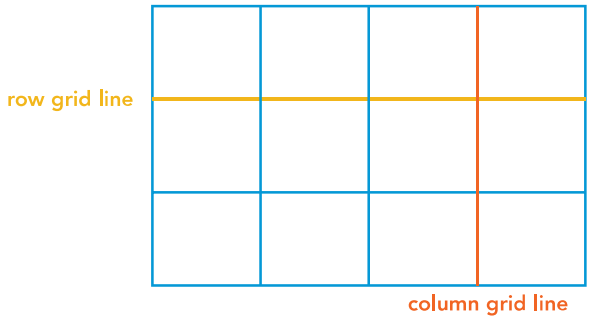
- **Bootstrap** (*getbootstrap.com*)
- **Neat** (*neat.bourbon.io*)
- **Unsemantic** (*unsemantic.com*)
- **Profound Grid** (*www.profoundgrid.com*)
- **HTML 5 Boilerplate** (*html5boilerplate.com*)
- **Skeleton** (*getskeleton.com*)

While a framework does a lot of the work in building the grid, you still need to understand how to interact with the underlying code, including the style sheets used to create a grid layout. In place of third-party frameworks, you can design your own grids using grid styles from CSS. Achieving Candidate Recommendation status by the W3C in December, 2017, the CSS grid styles are now widely supported by all major browsers on almost every device.

Introducing CSS Grids

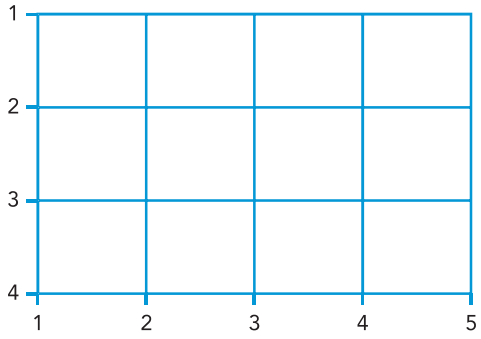
The **CSS grid model** is a set of CSS design styles used to create grid-based layouts. Before discussing the CSS styles, we should first explore the key terms and concepts associated with building a CSS grid. Each CSS grid is laid out in a set of row and column gridlines as shown in Figure 3–31.

Figure 3–31 Row and column gridlines



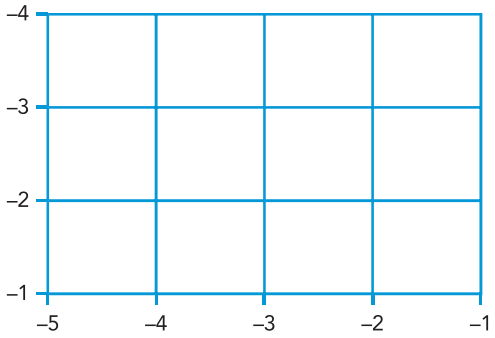
To reference positions within a grid, the CSS grid model numbers the gridlines in the horizontal and vertical directions, starting from the top-left corner of the grid with the row gridlines and then moving left to right with the column gridlines along the bottom. Both gridlines start with a value of "1" and increase in value down and across the grid (see Figure 3–32.)

Figure 3–32 Numbering gridlines



You can reference gridlines in the reverse order starting from the bottom-right corner with the first row and column gridlines in those directions are given a value of "−1" as shown in Figure 3–33.

Figure 3–33 Numbering gridlines from right to left



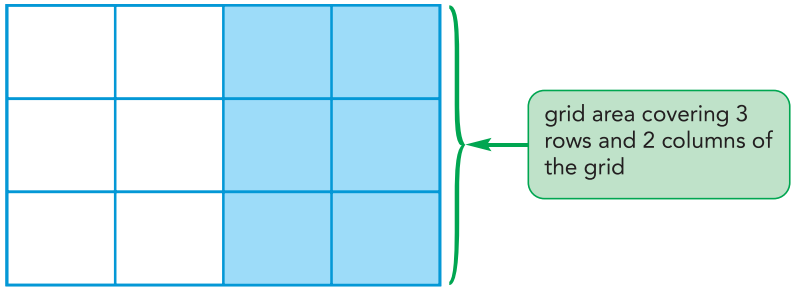
TIP

For countries and regions that read material right-to-left rather than left-to-right, the grid numbering system is reversed to reflect reading order.

The advantage of using both positive and negative gridline numbers is that you can always reference both the first gridline (1) and the last gridline (−1) no matter the size of the grid. This will become important later when placing items at specific locations within the grid or sizing those items to cover multiple rows and columns.

The cells that are created from the intersection of the horizontal and vertical gridlines will contain the elements from the web page. An element can be contained within a single cell or it can span several cells within a **grid area**. Figure 3–34 shows a grid area consisting of three rows and two columns. Note that grid areas must be rectangular; you cannot have an L-shaped grid area.

Figure 3–34 Grid area within a CSS grid



Rows and columns are also called **tracks** or **grid tracks**.

You will create a grid for a web page describing the Pandaisia Chocolates company. Anne has already written and marked up the content of the page. Open Anne's file now.

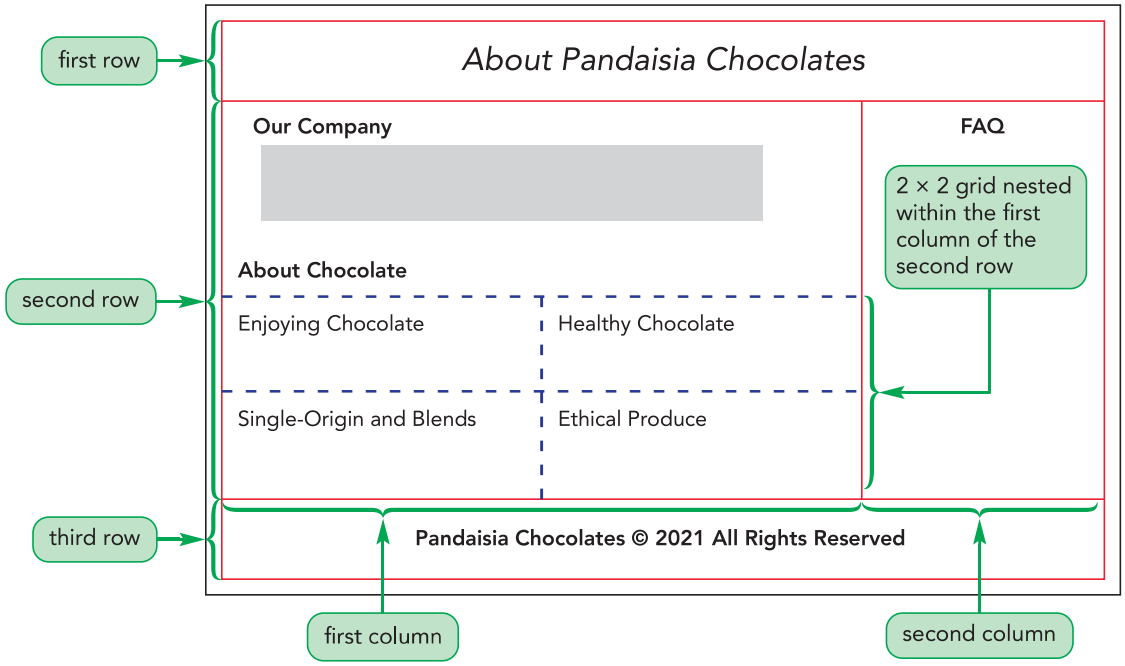
To open the file containing information about the company:

1. Use your editor to open the **pc_about_txt.html** file from the html03 ▸ tutorial folder. Enter ***your name*** and ***the date*** in the comment section and save the file as **pc_about.html**.
2. Take some time to examine the contents of the page.
3. Open the pc_about.html file in your browser. See Figure 3–35.

Figure 3–35 Initial About Pandaisia Chocolates page

There is currently no layout for the page contents and all the structural elements appear stacked within a single column. Anne sketches her idea for a different page layout, shown in Figure 3–36.

Figure 3–36 **Proposed grid layout for the About Pandaisia Chocolates page**



Anne's layout consists two grids: one nested within the other. The outer grid consists of three rows and two columns. The first and third rows contain the page header and page footer, with the header and footer both spanning an entire row. The second row displays information about the company in the first column and a list of frequently asked questions is displayed in the second column. Within the second row is a nested grid of two rows and two columns containing four articles about Pandaisia Chocolates, its operations, and products. You will use the CSS grid model to create this grid layout.

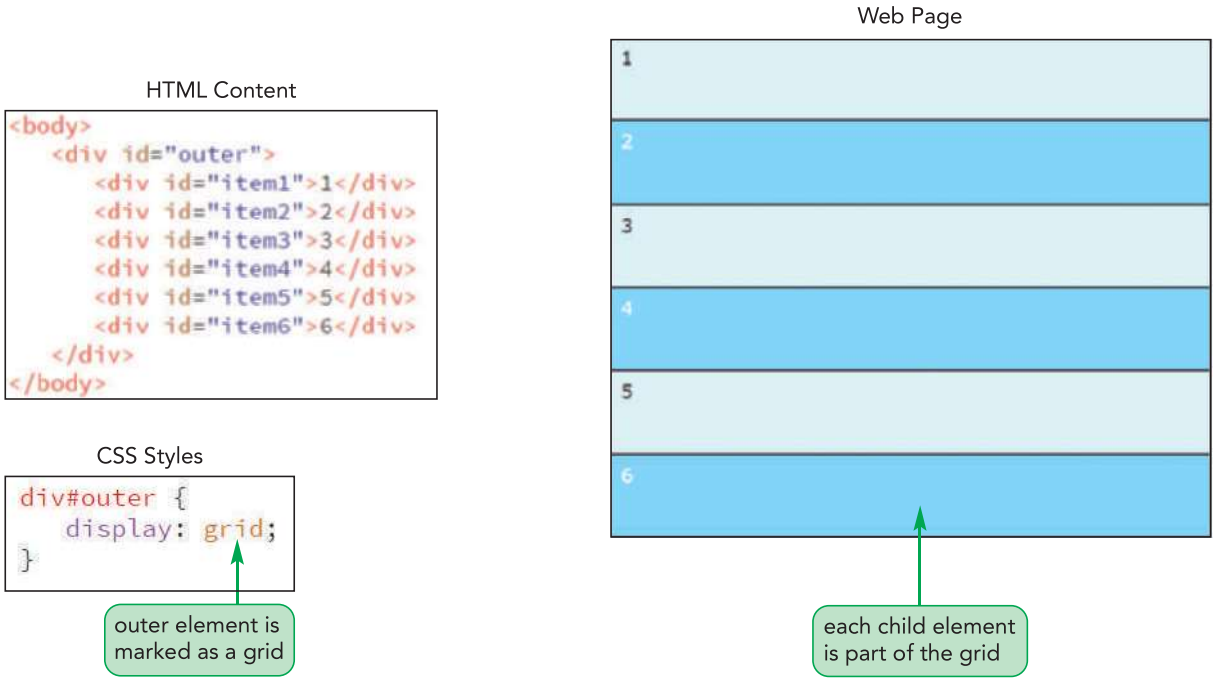
Creating a CSS Grid

To create a CSS grid, you must first identify a page element as the grid container using the following `display` property:

```
display: grid;
```

Figure 3–37 shows a simple web page containing a `div` element with the id "outer" that contains six nested `div` elements. The outer element is displayed as a grid and each of the six child elements become items within that grid. The grid is limited to those six `div` elements. Any elements nested within those `div` elements are not part of the grid structure.

Figure 3–37 Using the display style



The six child `div` elements are now grid items, so they are no longer considered block-level elements because they are fixed within the grid structure. You couldn't, for example, float any of those elements because floating them would remove them from the grid and the CSS grid model doesn't allow that. The entire grid itself is considered a block-level element and thus could be floated or resized within the web page just like any other block-level element.

Grids can also be created as inline elements using the style:

```
display: inline-grid;
```

which creates the grid inline with other elements in the web page. In this session we will only examine grids as block-level elements, but be aware than any of the techniques introduced for setting up a grid can be applied to both the block-level and inline versions.

Use the `display` property now to define outer and inner grids described in Figure 3–36. You will place all your grid styles within a separate CSS file.

To create the grid style sheet:

1. Use your editor to open the **pc_grids_txt.css** file from the html03 ▸ tutorial folder. Enter ***your name*** and ***the date*** in the comment section and save the file as **pc_grids.css**.
2. Within the Grid Styles for Page Body section, insert the following style rule to define a grid for the entire page body:

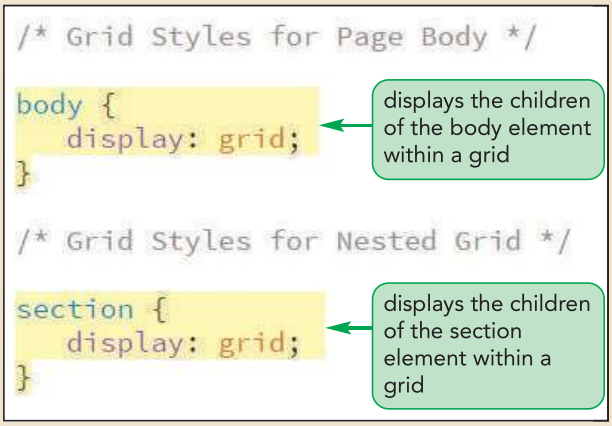
```
body {
   display: grid;
}
```

3. Within the Grid Styles for Nested Grid section, insert the following style rule to turn the `section` element into a grid:

```
section {
   display: grid;
}
```

Figure 3–38 highlights the newly added code.

Figure 3–38 Creating two grids for the web page



4. Save your changes to the file and then return to the **pc_about.html** file in your editor.
5. Within the head section, add the following `link` element to link the web page to the pc_grids.css style sheet.

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

6. Save your changes to the file.

Having set up the grids you will next use CSS to define the rows and columns of the grid structure.

Working with Grid Rows and Columns

To define the number and size of grid columns, use the following `grid-template-columns` style:

```
grid-template-columns: width1 width2 …;
```

where *width1*, *width2*, etc. is a space-separated list that defines the width of the columns or tracks within the grid. For example, the following style rule creates two grid columns: the first 250 pixels in width and the second with a width of 100 pixels.

```
grid-template-columns: 250px 100px;
```

TIP

The number of columns in the grid is determined by the number of entries in the *grid-template-columns* property.

Column widths can be expressed using any CSS unit measures such as pixels, em units, and percentages. You can also use the keyword `auto` to allow the column width to be automatically set by the browser. The following style creates a three-column grid with the width of the first column fixed at 100 pixels, the third column at 50 pixels, and the center column occupying whatever space remains.

```
grid-template-columns: 100px auto 50px;
```

You might use such a layout in a page in which the first and third columns contain navigation lists that are fixed in size while the middle column contains an article that should expand to fill the remaining space.

Figure 3–39 shows a two-column grid with fixed widths of 250 pixels and 100 pixels. Notice that the number of rows is not defined, so that the browser automatically adds rows as needed to contain all page elements within the grid container. Because there are six grid items, this grid is arranged in a layout of three rows and two columns. If there were eight child elements the grid would be four rows by two columns, and so forth.

Figure 3–39 Setting the grid columns

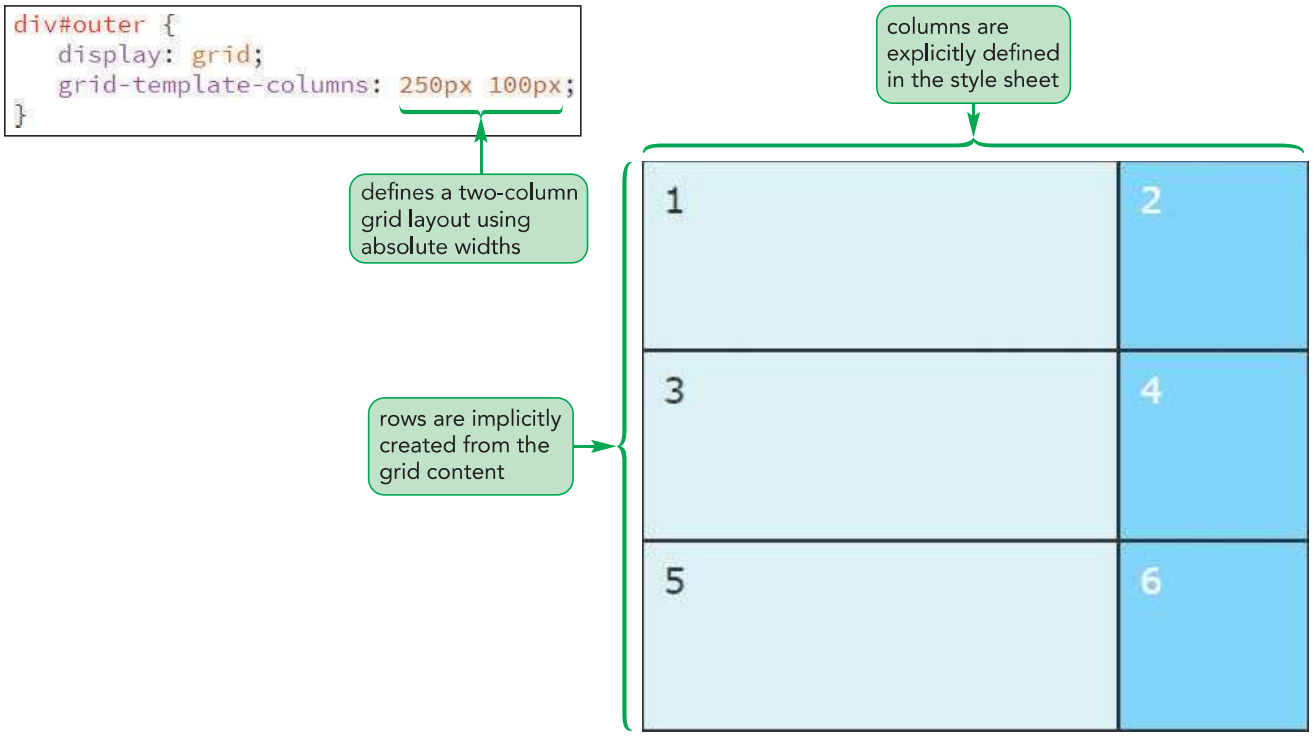


Figure 3–39 highlights an important contrast between explicit grids and implicit grids. An **explicit grid** completely defines the number and size of the grid rows and columns. An **implicit grid** contains rows and/or columns that are generated by the browser as it populates the grid with items from the grid container. In most grid layouts you will explicitly define the columns and let the browser fill out the grid rows drawn from the web page content.

To explicitly define the number of rows and their height, use the following `grid-template-rows` property:

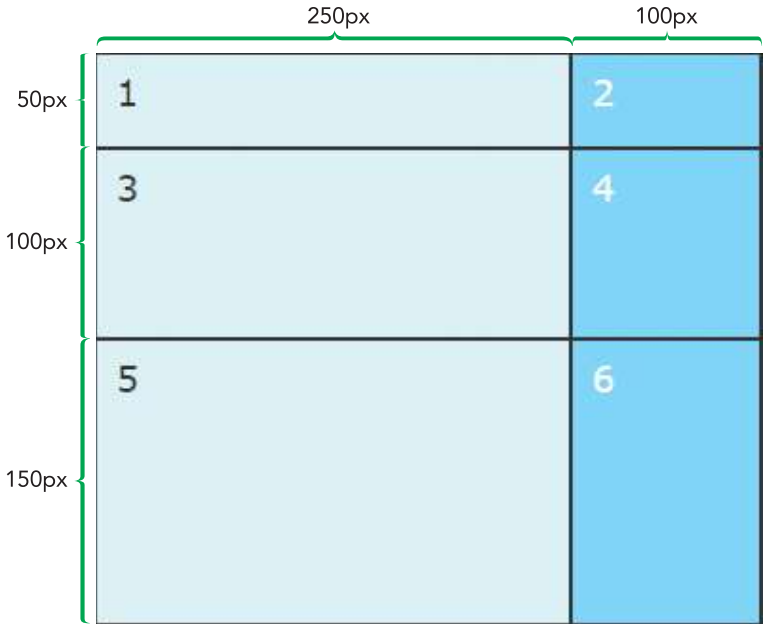
```
grid-template-rows: height1 height2 …;
```

where *height1*, *height2*, etc. define the heights of the grid rows. Figured 3-40 shows an example of an explicit grid in which both the columns and rows are defined in the CSS style sheet.

Figure 3–40 Explicitly defining grid columns and rows

```
div#outer {
   display: grid;
   grid-template-columns: 250px 100px;
   grid-template-rows: 50px 100px 150px;
}
```

explicitly defines both the rows and columns of the grid



In an implicit grid, the row heights are determined by the page element. You can also set the row heights in an implicit grid using the following `grid-auto-rows` property:

```
grid-auto-rows: height1 height2 …;
```

where *height1*, *height2*, etc. are the heights of the rows with the sequence repeating for each new set of rows. For example, the style:

```
grid-auto-rows: 100px;
```

sets the height of each row to 100 pixels, while the style

```
grid-auto-rows: 100px 200px;
```

sets the height of the first row in the implicit grid to 100 pixels, the height of the second row to 200 pixels, and then repeats those heights for each subsequent set of two rows until the grid is filled.

TIP

If the content of a grid item is greater than can be displayed within the allotted height, the browser will automatically increase the row height to match.

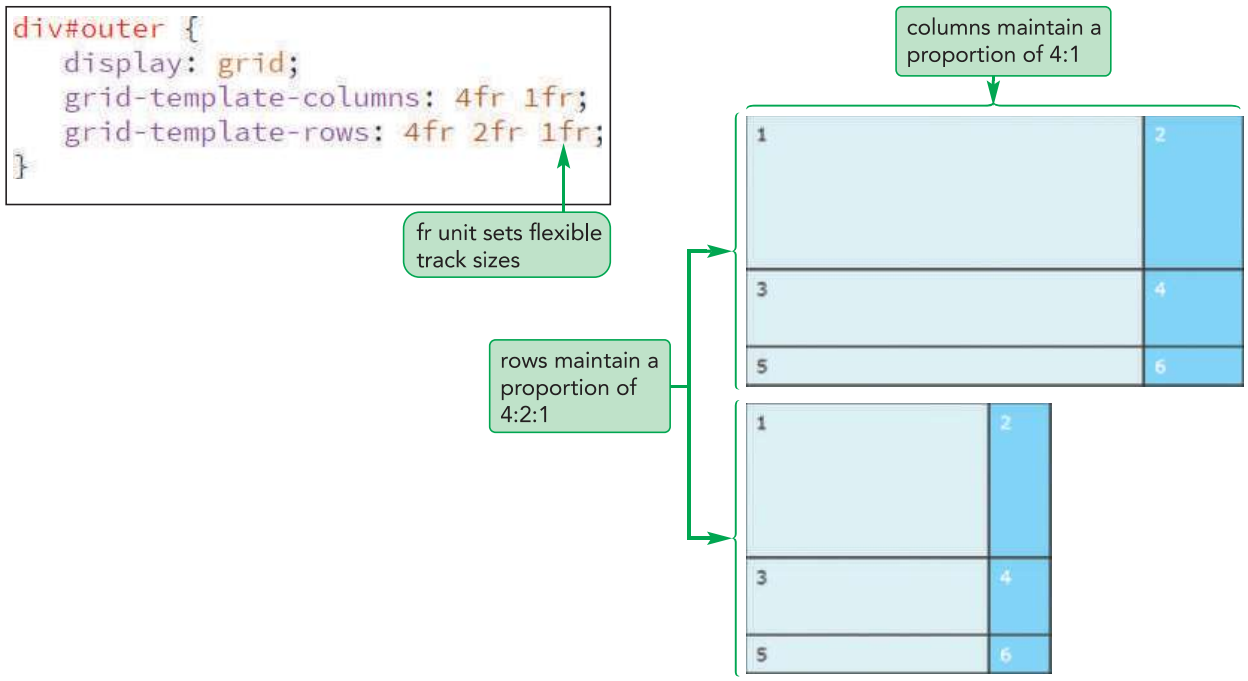
Track Sizes with Fractional Units

A grid layout will often need to adapt to devices of various screen widths and sizes. One way of accomplishing this is with flexible units. A **fr (fractional) unit**, indicated by the unit abbreviation `fr`, creates grid tracks that expand or contract in size to fill available space while retaining their relative proportions to one another. The following is an example of a grid in which the track sizes of the columns and rows is set using fractional units:

```
grid-template-columns: 4fr 1fr;
grid-template-rows: 4fr 2fr 1fr;
```

As the size of the display window changes, the column widths maintain their proportions so that the first column is always four times wider than the second column and the row heights maintain their proportion of 4:2:1. See Figure 3–41.

Figure 3–41 Using flexible units in a grid



Fractional units are often combined with absolute units to create grid layouts that are both fixed and flexible. The following style rule generates a grid in which the width of the first column is set to 250 pixels with the remaining space allotted to the other two columns in a proportion of 2 to 1.

```
grid-template-columns: 250px 2fr 1fr
```

Such a layout might be used in a web page in which the first column contains a navigation list whose width is fixed, the second column contains an article of primary importance, and the third column contains a sidebar of lesser importance. As the size of the display window changes, the width of the second and third columns automatically change, filling the screen while maintaining their 2:1 ratio.

Repeating Columns and Rows

Some grid layouts involve 12 or 16 columns or more. With so many columns it's difficult to specify the size of each column. You can simplify the layout style by using the following `repeat()` function

```
repeat(repeat, tracks)
```

where *repeat* is the number of repetitions of the tracks specified in *tracks*. For example, the following expression creates a layout consisting of one fixed column 250 pixels in width followed by four sets of two columns in which the first column in each set is twice the width of the second column for a total of nine grid columns.

```
grid-template-columns: 250px repeat(4, 2fr 1fr);
```

In place of a *repeat* value, you can use the keyword `auto-fill` to fill up the grid with as many columns (or rows) that will fit within the grid container. The following style uses the `auto-fill` keyword to fill the grid with as many 100 pixel-wide columns that will fit within the container:

```
grid-template-columns: 250px repeat(auto-fill, 100px);
```

Any grid item that cannot fit within the grid container will be automatically wrapped to a new row. This type of layout could be used with an image gallery in which each

row contains as many 100 pixel-wide images that can fit within the display window, arranged in columns.

Finally, you can switch between fixed and flexible track sizes using the following `minmax()` function

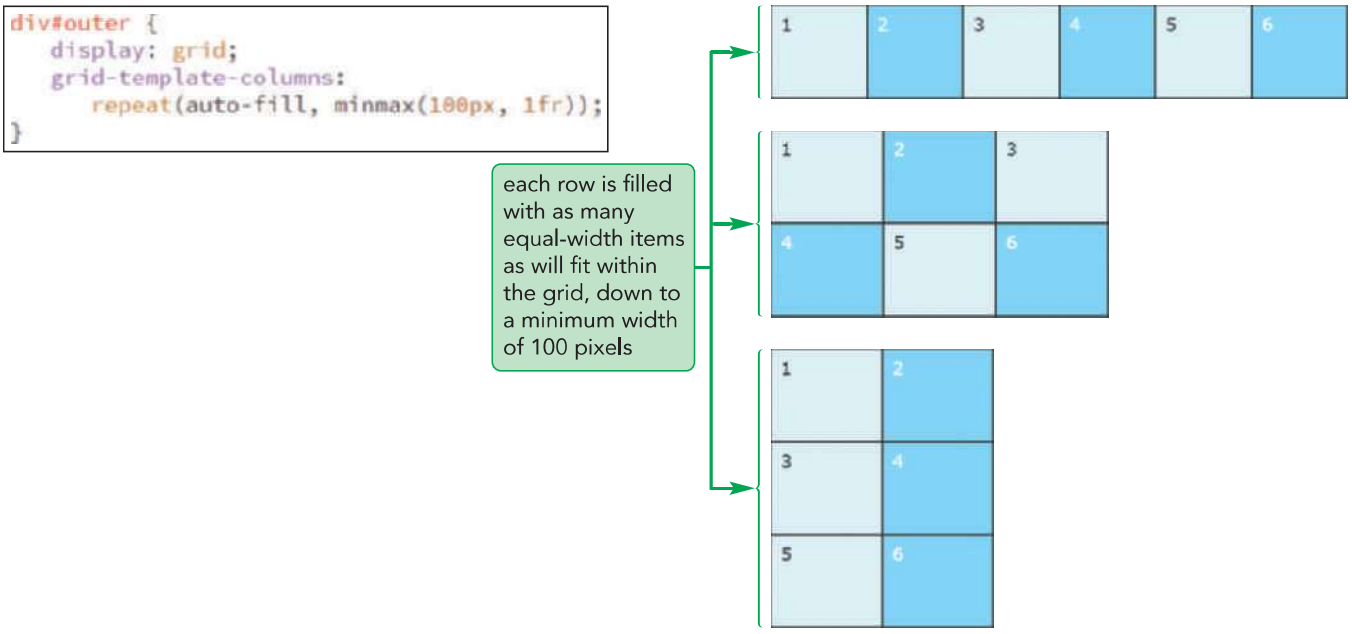
```
minmax(min, max)
```

where *min* is the minimum track size for a row and column and *max* is the maximum. Used in the following style rule, the grid will contain as many columns of equal width that can fit within a grid container down to a minimum width of 100 pixels:

```
grid-template-columns: repeat(auto-fill, minmax(100px, 1fr));
```

Figure 3–42 shows how such a layout would be applied to grid containers of different widths.

Figure 3–42 Using the minmax function in a grid



As the grid container decreases in size, grid items are automatically wrapped to a new row. Under each layout, the columns are given equal widths down to a minimum width value of 100 pixels.

Applying a Grid Layout

Now that you've seen how to set the size of rows and columns within a grid, you will apply your knowledge to the About Pandaisia web page. From Anne's proposed layout shown earlier in Figure 3–36, you've learned that the two columns in the outer grid should be displayed in a proportion of 2:1, while the four articles about chocolate in the nested grid should be displayed with columns of equal width. You will define the column widths for both the outer and nested grids using fractional units. You won't, however, explicitly define the number and height of the grid rows, leaving the browser to implicitly lay out that content.

To define the grid columns:

1. Return to the **pc_grids.css** file in your editor.
2. Within the style rule for the `body` element, add the style:

```
grid-template-columns: 2fr 1fr;
```

3. Within the style rule for the `section` element, add:

```
grid-template-columns: repeat(2, 1fr);
```

Figure 3–43 highlights the newly added code.

Figure 3–43 Creating two grids for the web page

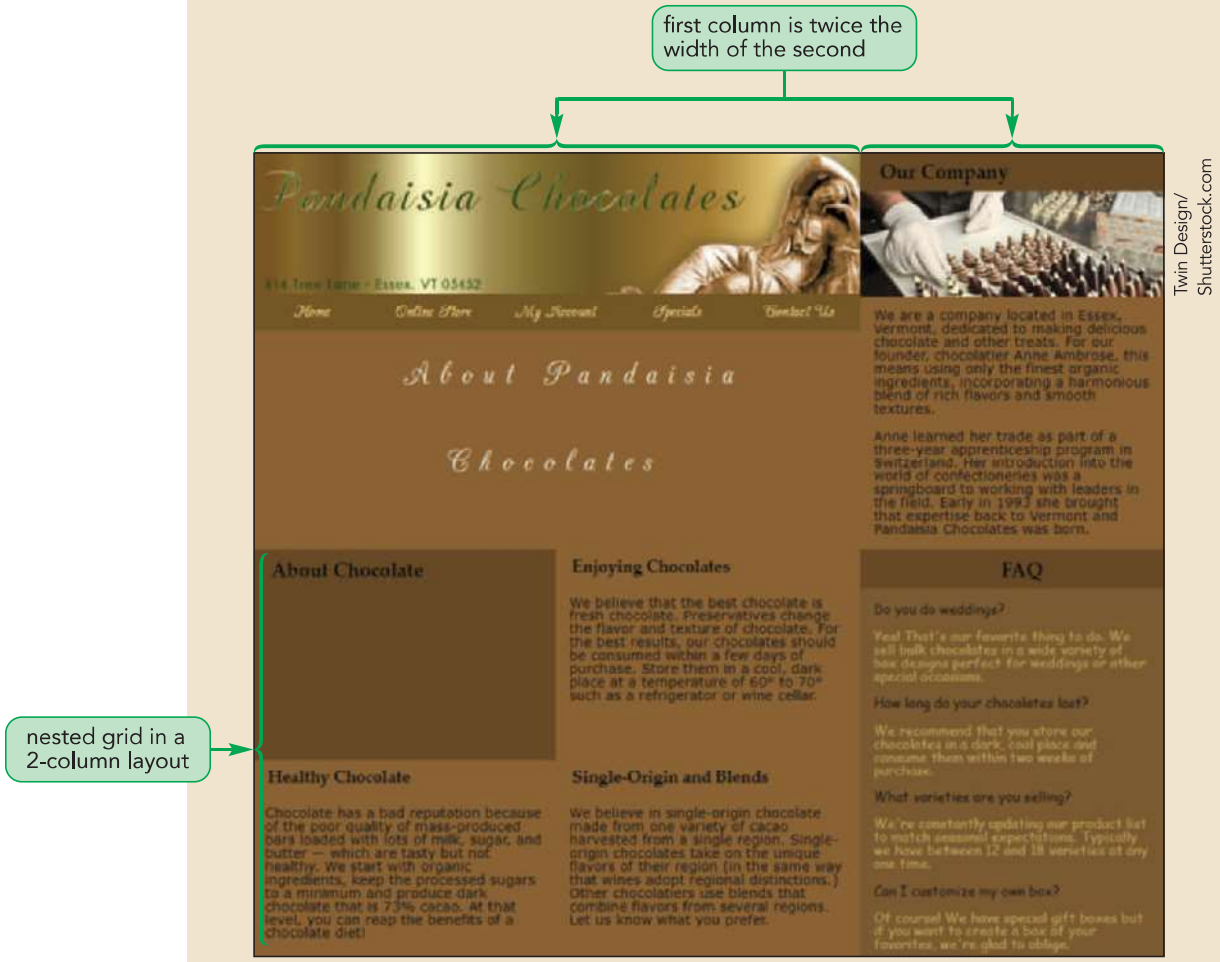
```
/* Grid Styles for Page Body */

body {
   display: grid;
   grid-template-columns: 2fr 1fr;
}

/* Grid Styles for Nested Grid */

section {
   display: grid;
   grid-template-columns: repeat(2, 1fr);
}
```

creates two grid columns with the first column twice the width of the second

creates two grid columns of equal width

4. Save your changes to the file and then reload pc_about.html in your browser. Figure 3–44 shows part of the layout of the page under the current grid structure.

Figure 3–44 Web page with the column layout



The page layout does not resemble the plan that Anne outlined in Figure 3–36 because we have not specified where each item should be placed within the grid. That will be the final step in designing the grid layout. Before doing that however, it would be helpful to view the page with gridlines superimposed on the web page. We can do that with outline styles.

Outlining a Grid

Outlines are simply lines drawn around an element, enclosing the element content, padding, and border spaces. Unlike borders, which you'll study in the next tutorial, an outline doesn't add anything to the width or height of the object, it only indicates the extent of the element on the rendered page.

The width of the line used in the outline is defined by the following `outline-width` property

```
outline-width: value;
```

where *value* is expressed in one of the CSS units of length, or with the keywords `thin`, `medium`, or `thick`. The line color is set using the `outline-color` property

```
outline-color: color;
```

where *color* is a CSS color name or value. Finally, the design of the line can be set using the following `outline-style` property

```
outline-style: style;
```

where *style* is `none` (to display no outline), `solid` (for a single line), `double`, `dotted`, `dashed`, `groove`, `inset`, `ridge`, or `outset`. All the outline properties can be combined into the following **`outline`** shorthand property

```
outline: width style color;
```

where *width*, *style*, and *color* are the values for the line's width, design, and color. For example, the following style rule uses the wildcard selector along with the outline shorthand property to draw a 1 pixel dotted green line around every element on the web page:

```
* {
   outline: 1px dotted green;
}
```

TIP

Outlines can also be applied to inline elements such as inline images, citations, quotations, and italicized text.

Note that there are no separate outline styles for the left, right, top, or bottom edge of the object. The outline always surrounds an entire element.

REFERENCE

Adding an Outline to an Element

- To add an outline around an element, use the property

  ```
  outline: width style color;
  ```

  where *width*, *style*, and *color* are the outline width, outline design, and outline color respectively. These attributes can be listed in any order.

Use the `outline` property now to add an outline to each item in both the outer and inner grids.

To define the grid columns:

1. Return to the **pc_grids.css** file in your editor.
2. At the bottom of the style sheet, add the following style rule to place a red dashed outline around every child of the `body` element:

```
body > * {
   outline: 2px dashed red;
}
```

3. Add the following style rule to place a blue dashed outline around every child of the `section` element:

```
section > * {
   outline: 2px dashed blue;
}
```

Figure 3–45 highlights the code for the style rules.

TIP

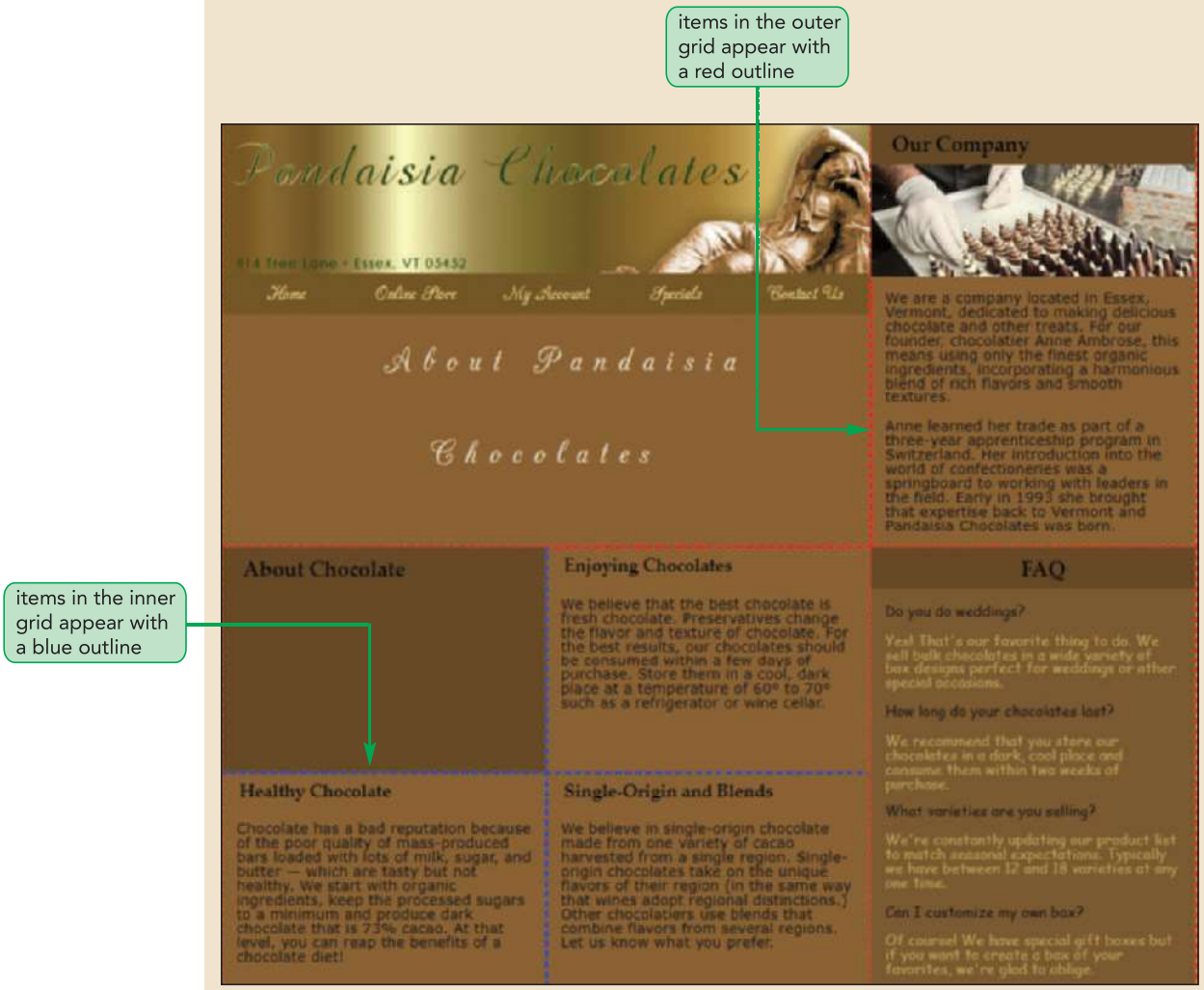
Most browsers include developer tools for viewing the gridlines from a CSS grid. See your browser documentation for specific instructions.

Figure 3–45 Adding outlines to grid items

```
section {
   display: grid;
   grid-template-columns: repeat(2, 1fr);
}

body > * {
   outline: 2px dashed red;
}

section > * {
   outline: 2px dashed blue;
}
```

4. Save your changes to the file and then reload pc_about.html in your browser. Figure 3–46 shows the outlines around both the outer and inner grids.

Figure 3–46 Web page with grid items outlined



Twin Design/ Shutterstock.com

Adding an outline makes it clear how each item is placed within the grid. Next you will change the location and sizes of the grid items to match Anne's proposed layout.

Placing Items within a Grid

By default, grid items are laid out in document order going from left to right and up to down, with each item placed within a single cell. Thus, the page header in Figure 3–46, being the first item in the grid, appears in the first row and column. The next item, an article about the company, occupies the cell in the second column of the first row. Subsequent items are placed in cells in the next rows filling the grid until the last item is reached, which in this case is the page footer. In many layouts however, you might want to move items around or a have a single item occupy multiple rows and columns.

REFERENCE

Defining Grids with CSS

- To assign a CSS grid to an element, use the property

```
display: grid;
```

- To define the number of rows and columns within the grid, use the properties

```
grid-template-rows: height1 height2 …;
grid-template-columns: width1 width2 …;
```

where *height1*, *height2*, *width1*, *width2*, etc. define the row heights or column widths.

- To place an element within a specific intersection of grid rows and columns, use the properties

```
grid-row-start: integer;
grid-row-end: integer;
grid-column-start: integer;
grid-column-end: integer;
```

where *integer* defines the starting and ending row or column that contains the content.

- To more compactly set the location of the element within the grid, use the properties

```
grid-row: start/end;
grid-column: start/end;
```

where *start* and *end* are the starting and ending coordinates of the row and columns containing the element.

Placing Items by Row and Column

To move a grid item to a specific location within the grid, use the following `grid-row` and `grid-column` properties:

```
grid-row: row;
grid-column: column;
```

where *row* is the row number and *column* is the column number. Thus, to place the `article` element in a **grid cell** located in the first row and second column of the grid, apply the following style rule:

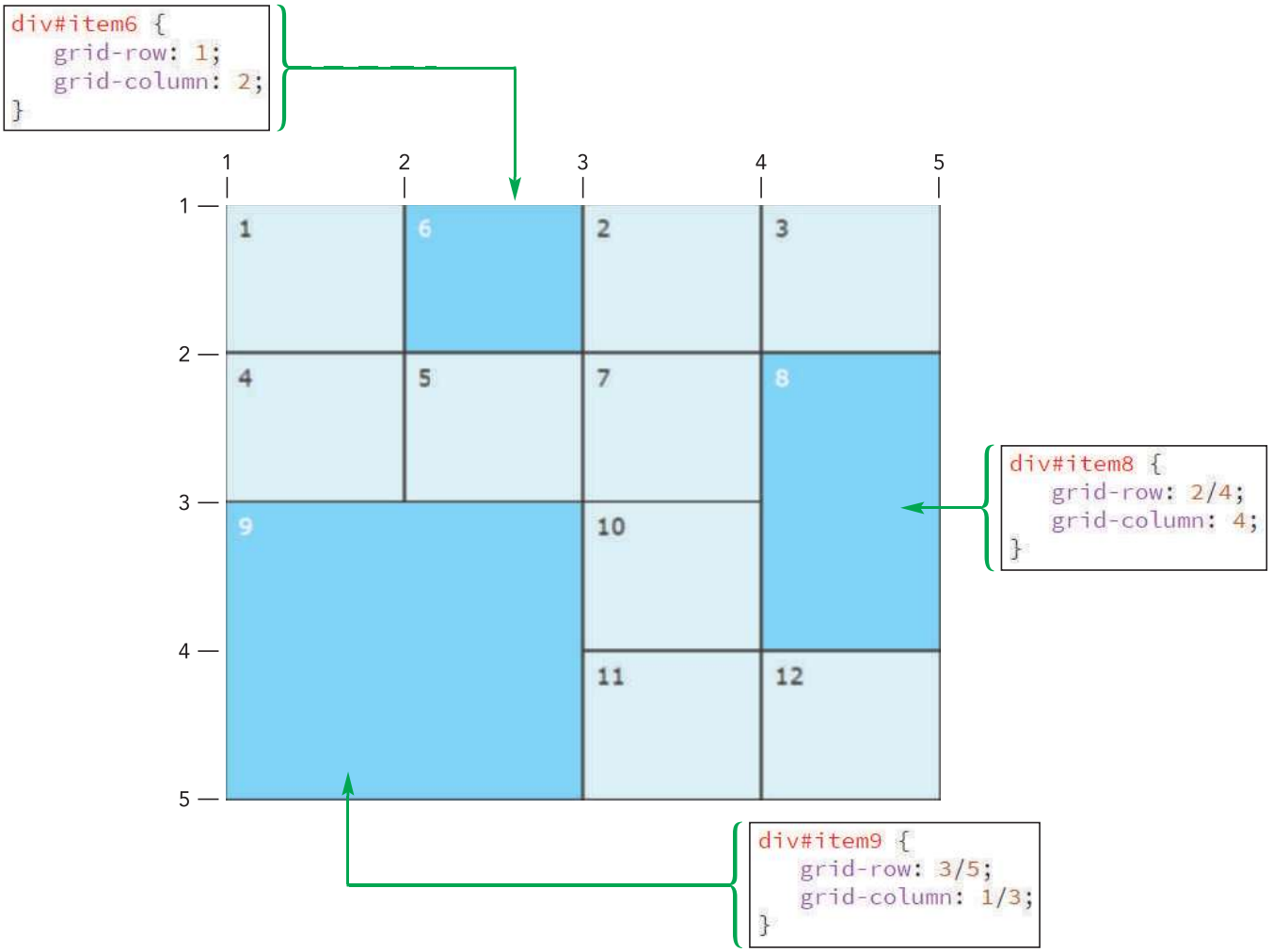
```
article {
   grid-row: 1;
   grid-column: 2;
}
```

To extend a grid item so that it covers multiple rows or multiple columns, include the starting and ending gridline in the style property as follows:

```
grid-row: start/end;
grid-column: start/end;
```

where *start* is the starting gridline and *end* is the ending gridline. Figure 3–47 shows a page layout in which grid items 6, 8, and 9 have been moved and resized using the `grid-row` and `grid-column` properties. For example, item 6 is moved to the first row and second column of the grid while items 8 and 9 have been resized to cover multiple rows and/or columns. The other items in the grid are placed in their default locations and sized to fit within a single grid cell.

Figure 3–47 Placing items within a grid layout



Starting and ending gridlines can also be expressed in the following four properties:

```
grid-column-start: integer;
grid-column-end: integer;
grid-row-start: integer;
grid-row-end: integer;
```

so that the style rule grid-column: 2/5 is equivalent to:

```
grid-column-start: 2;
grid-column-end: 5;
```

Which approach you use is a matter of personal preference.

You can also use negative gridline numbers (shown earlier in Figure 3–33) to extend an item from the first gridline to the last. Recall that since the last column or row gridline has a value of -1, the expression

```
grid-column: 1/-1;
```

would extend the grid item across the entire row from the first gridline to the last. Similarly, the expression

```
grid-row: 1/-1;
```

would create a grid item that extends across an entire column. Note that this technique only works with explicit grids because in an implicit grid there is no defined last column or row.

INSIGHT

Naming Gridlines

Gridline numbers can be difficult and cumbersome to work with, so the CSS grid model also supports **gridline names**, which are descriptive names for row and column gridlines. Gridline names are created by adding a name enclosed within square brackets into the `grid-template-columns` or `grid-template-rows` style. For example, the following style creates a grid with three columns and four column gridlines named row-start, main-start, main-end, and row-end.

```
grid-template-columns: [row-start] 50px [main-start] 250px
[main-end] 100px [row-end];
```

To extend a grid item across the entire row, you could apply the style:

```
grid-column: 1/4;
```

or

```
grid-column: row-start/row-end;
```

An article could be placed within the center grid column with the style:

```
grid-column: main-start/main-end;
```

Gridline names can make your CSS code easier to interpret and manage and can be more easily updated if you insert additional rows or columns within your grid layout.

Using the span Keyword

Another way of setting the size of a grid cell is with the `span` keyword. The general syntax is:

```
grid-row: span value;
grid-column: span value;
```

where *value* is the number of rows or columns covered by the item. The following style rule extends the `article` element across 2 rows and 3 columns of the grid.

```
article {
   grid-row: span 2;
   grid-column: span 3;
}
```

To specify both the location and the size of the item, include the starting gridline in the style rule. The following style rule places the `article` element in the first row and fourth column of the grid while spanning two rows and three columns from that location.

```
article {
   grid-row: 1/span 2;
   grid-column: 4/span 3;
}
```

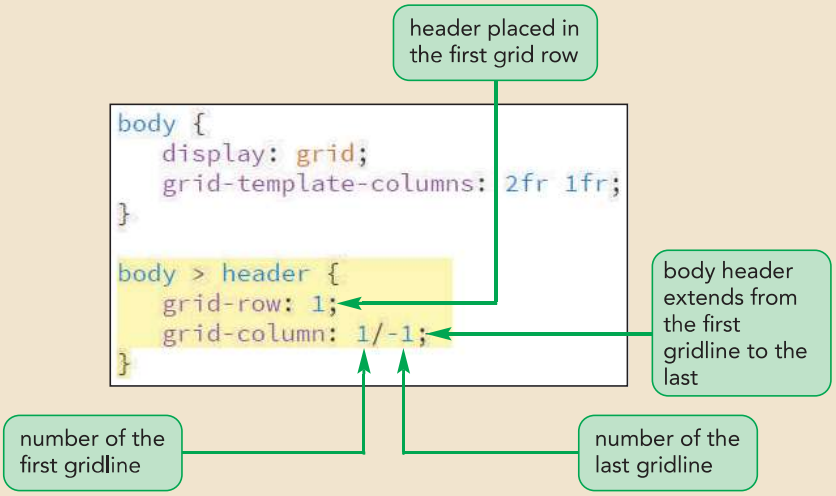
In Anne's proposed layout, the page header should occupy a single row, extending from the first column to the last. Use the `grid-column` style rule now to display the body header across the first row of the grid.

To place the body header across the first row:

1. Return to the **pc_grids.css** file in your editor.
2. Directly below the style rule for the `body` element, insert the following style rule as shown in Figure 3–48.

```
body > header {
   grid-row: 1;
   grid-column: 1/-1;
}
```

Figure 3–48 Spanning the body header across the grid row



3. Save your changes to the file and reload pc_about.html in your browser. The body header extends across the first row of the grid (see Figure 3–49.)

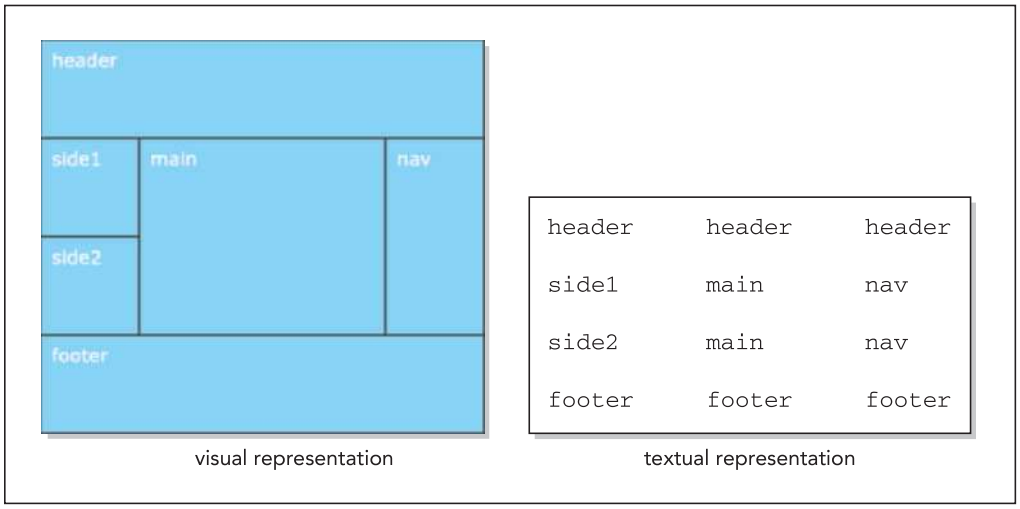
Figure 3–49 Body header in the first row

Gridlines are a quick and effective method of placing and sizing grid items, but they can be confusing when applied to a grid of several rows and columns. An easier and more intuitive approach is to use grid areas.

Placing Grid Items by Area

In the grid areas approach to layout you identify sections of the grid with item names, creating a textual representation of the layout. Figure 3–50 shows a grid of four rows and three columns in which several items span multiple rows and columns, represented both visually as it would appear on the web page and textually.

Figure 3–50 Mapping out grid areas



To create a textual representation in a style sheet, use the following `grid-template-areas` property:

```
grid-template-areas: "row1"
                     "row2"
                     …;
```

where *row1*, *row2*, etc. are text strings containing the names of the areas for each row. Thus, to create the grid layout shown in Figure 3–50, you would enter the style:

```
grid-template-areas: "header header header"
                     "side1 main nav"
                     "side2 main nav"
                     "footer footer footer";
```

You will add a `grid-template-areas` property to the style sheet, representing the layout Anne proposed in Figure 3–36.

To place the body header across the first row:

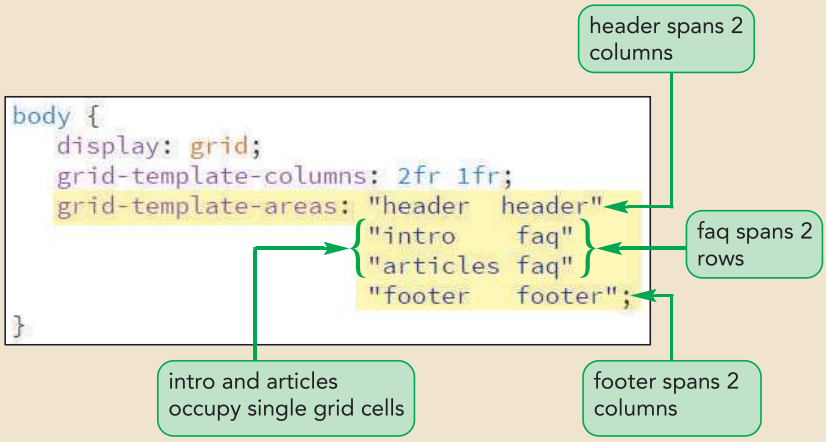
1. Return to the **pc_grids.css** file in your editor.
2. Within the style rule for the body element, insert the following style:

```
grid-template-areas: "header   header"
                     "intro    faq"
                     "articles faq"
                     "footer   footer";
```

See Figure 3–51.

Make sure you enclose each row of the grid layout within a set of quotation marks.

Figure 3–51 Defining areas for the outer grid



To assign elements to grid areas, use the following `grid-area` property:

```
grid-area: area;
```

where *area* is the name of an area defined in the `grid-template-areas` property. Use the `grid-area` property now to assign elements from the web page to areas within the grid.

To assign the page elements to grid areas:

1. Below the `body > header` style rule, add the following style to assign the `article` element to the intro grid area.

   ```
   body > article {grid-area: intro;}
   ```

2. Place the `aside` element in the faq grid area with the style:

   ```
   body > aside {grid-area: faq;}
   ```

3. Place the `section` element in the articles grid area with the style:

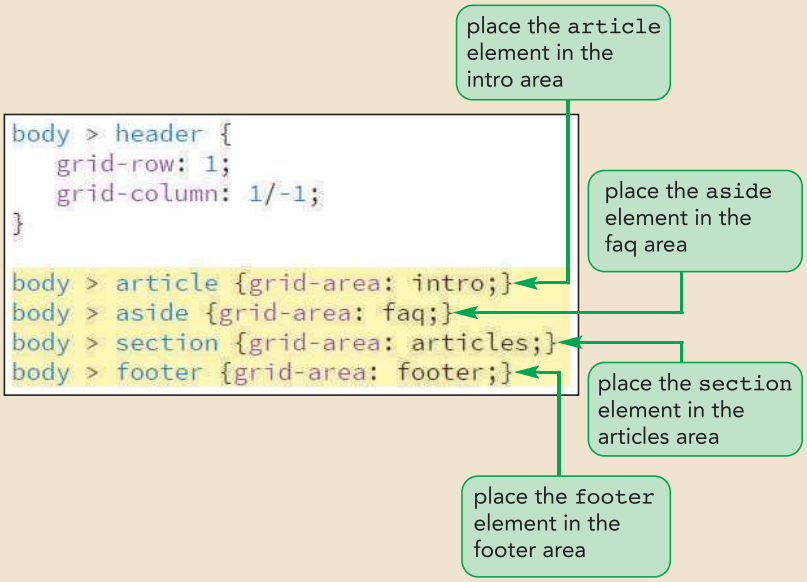
   ```
   body > section {grid-area: articles;}
   ```

4. Place the body `footer` element in the footer grid area using the style:

   ```
   body > footer {grid-area: footer;}
   ```

   Figure 3–52 highlights the newly added code.

Figure 5–52 Assigning elements to grid areas



The `grid-area` property can also be used as a shorthand to place and size grid items using gridline numbers. The general syntax is:

```
grid-area: row-start/col-start/row-end/col-end;
```

where *row-start*, *col-start*, *row-end*, and *col-end* are the starting and ending gridline numbers from the grid's rows and columns. For example, the following expression places the grid item to extend from the first row gridline through the fourth and from the third column gridline through the fifth.

```
grid-area: 1/3/4/5;
```

The only remaining part of the About Pandaisia web page that needs to be placed within the layout is the "About Chocolate" h1 heading that appears in the nested grid. Anne wants this heading to extend across two columns in the first row of that grid. Add a style rule to place the heading now.

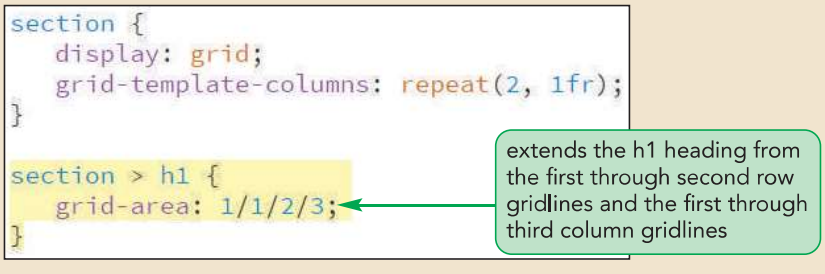
To place the h1 heading:

1. Below the style rule for the `section` element, add the following rule to place the h1 heading:

```
section > h1 {
   grid-area: 1/1/2/3;
}
```

See Figure 3–53.

Figure 3–53 Placing the h1 heading



2. Save your changes to the pc_grids.css file and then reload the pc_about.html file in your browser. Figure 3–54 shows the complete layout of the page.

Figure 3–54 About Pandaisia web page

Twin Design/Shutterstock.com

3. Return to the **pc_grids.css** file in your text editor.
4. Remove the two style rules that create the red and blue dashed outlines and then save your changes to the file.
5. Reload the pc_about.html file in your browser and confirm that the grid outlines are removed from the rendered page.

Compare the appearance of the page content in Figure 3–54 with the schematic diagram shown earlier in Figure 3–36 to see how using a grid provided a unified layout for the page. As you become more experienced with setting up and applying grids, you can move to more intricate and dynamic page layouts.

INSIGHT

Generating Content with Lorem Ipsum

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer nec odio. Praesent libero. Sed cursus ante dapibus diam. Sed nisi. Nulla quis sem at nibh elementum imperdiet. Duis sagittis ipsum. Vestibulum lacinia arcu eget nulla. Sed dignissim lacinia nunc.

That previous paragraph is an example of **lorem ipsum**, which is nonsensical, improper Latin commonly used in page design as filler text. Rather than creating large portions of sample text before you can view your layout, lorem ipsum is used to quickly generate sentences, lines, and paragraphs that resemble the structure and appearance of real text. Lorem ipsum is a particularly useful tool for web designers because they can begin working on page design without waiting for their clients to supply all the page content.

Many popular web editors include tools to generate lorem ipsum text strings in a wide variety of formats and styles. There are also lorem ipsum generators freely available on the web that supplement the lorem ipsum text with HTML markup tags.

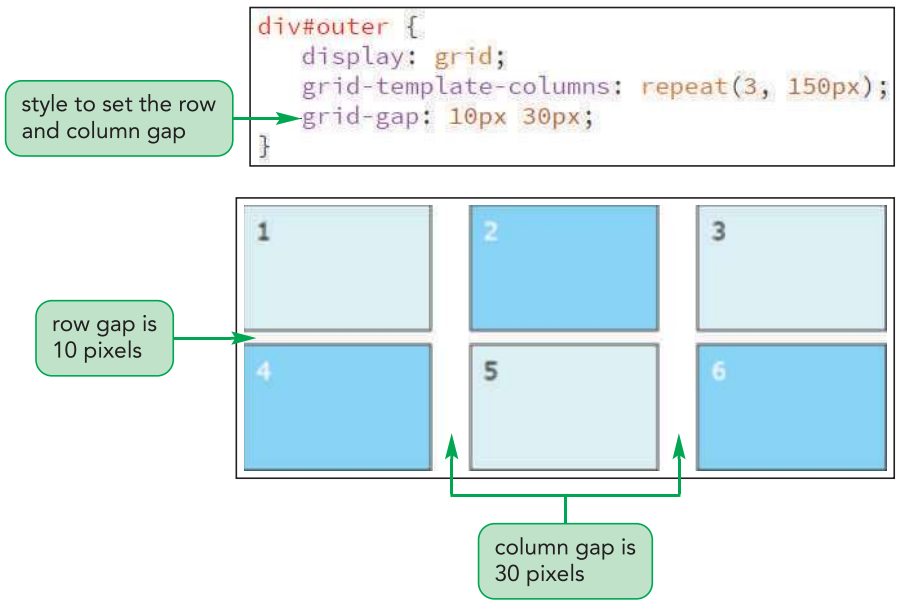
Defining the Grid Gap

Another part of grid layout is defining the space between items in a grid. So far, all our layouts have assumed no spacing, but many layouts include interior spaces to allow each item "room to breathe." The gap size is defined using the following `grid-gap` property:

```
grid-gap: row column;
```

where *row* is the internal space between grid rows and *column* is the internal space between grid columns. Figure 3–55 shows a grid layout in which the rows are separated by a 10-pixel space and the columns by a space of 30 pixels.

Figure 3–55 Setting the grid gap



You can also set the grid gaps for rows and columns using the following properties:

```
grid-column-gap: value;
grid-row-gap: value;
```

where *value* is the size of the gap in one of the CSS units of measure. Anne wants you to add a 15-pixel column gap to the About Pandaisia web page but leave the row gap at its default value of 0 pixels.

To set the size of the column gap:

1. Return to the **pc_grids.css** file in your editor.
2. Within the style rule for the body element, add the following property to set the column gap size as shown in Figure 3–56.

```
grid-column-gap: 15px;
```

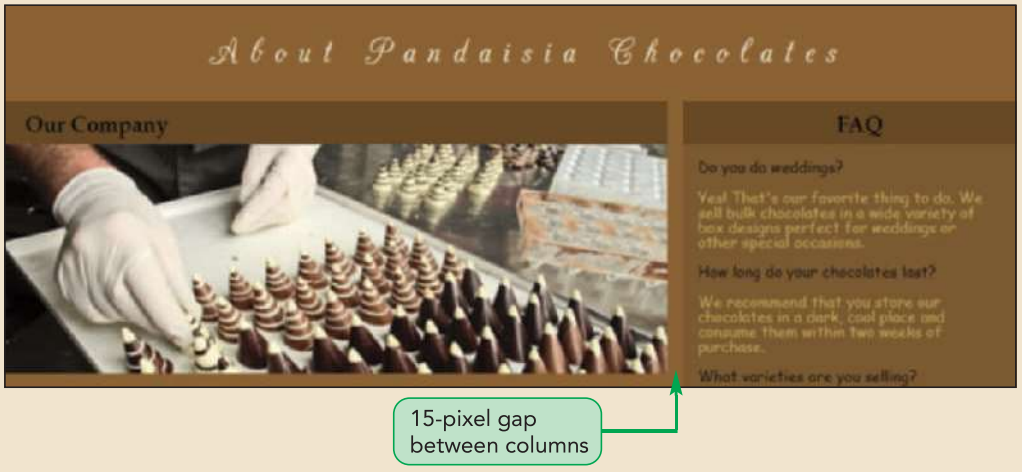
Figure 3–56 Setting the size of the column gap

```
body {
   display: grid;
   grid-template-columns: 2fr 1fr;
   grid-template-areas: "header  header"
                        "intro   faq"
                        "articles faq"
                        "footer  footer";
   grid-column-gap: 15px;
}
```

interior space between columns set to 15 pixels

3. Save your changes to the file then reload the pc_about.html file in your browser. As shown in Figure 3–57, the gap between the first and second columns is set to 15 pixels.

Figure 5–57 Gap between the first and second columns



Twin Design/Shutterstock.com

4. You've completed your work on the web page. Close the pc_about.html and pc_grids.css files.

Note that, unlike margins, the gap size setting is applied only to the interior space between the grid items and not to the exterior space between the grid items and the edge of the grid container.

Managing Space within a Grid

The `grid-gap` property allows you to define the space between grid items. CSS includes other properties to manage the space around the content of each grid cell. By default, there is no space between the grid cell borders and the grid cell content. However, you can position the content within the grid cell using the `align-items` and `justify-items` properties. The `align-items` property sets the vertical placement of the content, while the `justify-items` property sets the horizontal placement. The syntax of both properties is:

```
align-items: placement;
justify-items: placement;
```

where *placement* is:

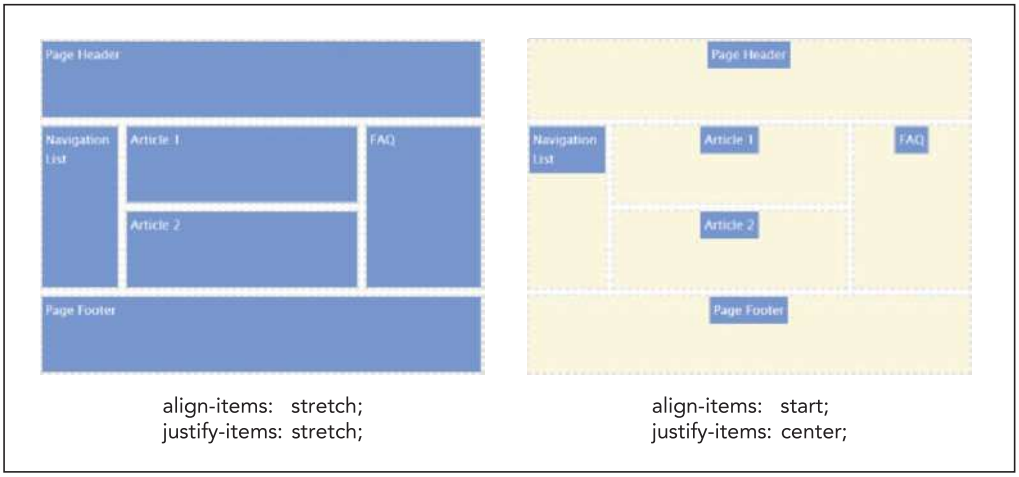
- `stretch` to expand the content between the top/bottom or left/right edges, removing any spacing between the content and the cell border (the default)
- `start` to position the content with the top or left edge of the cell
- `end` to position the content with the bottom or right edge of the cell
- `center` to center the content vertically or horizontally within the cell

TRY IT

You can explore the `align-items` and `justify-items` properties using the demo_grid1.html file from the html03 ▸ demo folder.

Figure 3–58 shows the impact of the `align-items` and `justify-items` properties on the placement of the content within each grid cell. By default, there is no spacing between the content and the cell border as the content "stretches" to fill the cell (shown in the grid on the left in the figure). In the grid on the right, the content is placed at the start (top) and horizontal center of the cell and spacing is added between the cell content and the cell borders.

Figure 3–58 Applying the align-items and justify-items properties



align-items: stretch;
justify-items: stretch;

align-items: start;
justify-items: center;

Alignment for a Single Grid Cell

The `align-items` and `justify-items` properties apply to every cell in the grid. To align and justify only one cell, apply the `align-self` and `justify-self` properties to the content within the grid cell. The placement values are the same as for the `align-items` and `justify-items` properties. For example, the following style rule displays the content of the `article` element in both the horizontal and vertical center of the grid cell.

```
article {
   align-self: center;
   justify-self: center;
}
```

Using the `align-self` and `justify-self` properties is a quick and easy way of centering your content within the web page. Before the introduction of the CSS grid model, this was a difficult task involving a lot of CSS hacks, but now it can be accomplished by simply placing the item within a grid and centering the content both horizontally and vertically.

Aligning the Grid

In some layouts involving fixed units, the space taken up by the items within the grid will be less than the total space allotted to the grid container itself, creating unused space in the container. By default, the grid will be positioned at the top-left corner of the container, but you can modify that position using the `align-content` and `justify-content` properties:

```
align-content: placement;
justify-content: placement;
```

where *placement* is:

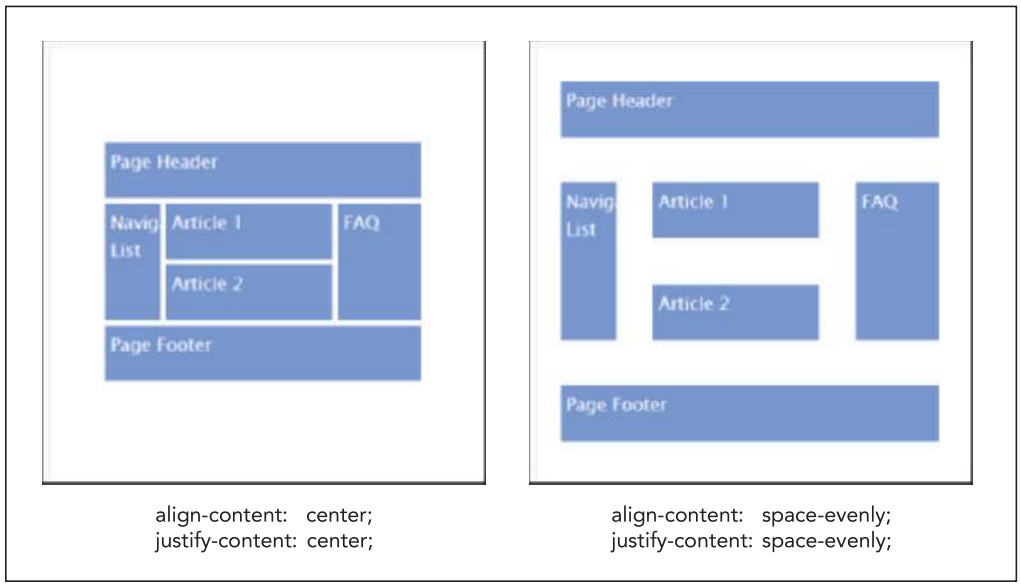
- `start` to position the grid with the top or left edge of the container (the default)
- `end` to position the grid with the bottom or right edge of the container
- `center` to center the grid vertically or horizontally within the container
- `space-around` to insert an even amount of space between each grid item, with half-sized spaces on the far ends
- `space-between` to insert an even amount of space between each grid item, with no space at the far ends
- `space-evenly` to insert an even amount of space between each grid item, including the far ends

TRY IT

You can explore the `align-content` and `justify-content` properties using the demo_grid2.html file from the html03 ▸ demo folder.

As with the other grid properties, the `align-content` property positions the grid vertically within the container and the `justify-content` property moves the grid horizontally. Figure 3–59 shows how the interior space within the grid container can be managed using the `align-content` and `justify-content` properties. In the left grid, the layout is centered both horizontally and vertically within the container. In the right grid, the grid items themselves are positioned evenly within the container.

Figure 3–59 Applying the align-content and justify-content properties



The `align-content` and `justify-content` properties are useful when you want to center your entire layout within the web page in both the horizontal and vertical directions.

PROSKILLS

Written Communication: Getting to the Point with Layout

Page layout is one of the most important aspects of web design. A well-constructed layout naturally guides a reader's eyes to the most important information in the page. You should use the following principles to help your readers quickly get to the point:

- *Guide the eye.* Usability studies have shown that a reader's eye first lands in the top center of the page, then scans to the left, and then to the right and down. Arrange your page content so that the most important items are the first items a user sees.
- *Avoid clutter.* If a graphic or an icon is not conveying information or making the content easier to read, remove it.
- *Avoid overcrowding.* Focus on a few key items that will be easy for readers to locate while scanning the page, and separate these key areas from one another with ample white space. Don't be afraid to move a topic to a different page if it makes the current page easier to understand.
- *Make it manageable.* It's easier for the brain to process information when it's presented in smaller chunks. Break up long extended paragraphs into smaller paragraphs or bulleted lists.
- *Use a grid.* Users find it easier to digest content when it's aligned vertically and horizontally. Using a grid helps you line up your elements in a clear and consistent way.
- *Cut down on distractions.* If you're thinking about using blinking text or a cute animated icon, *don't*. The novelty of such features wears off very quickly and distracts users from the valuable content.

Always remember that your goal is to convey information to readers, and that an important tool in achieving that is to make it as easy as possible for readers to find that information. A thoughtfully constructed layout is a great aid to effective communication.

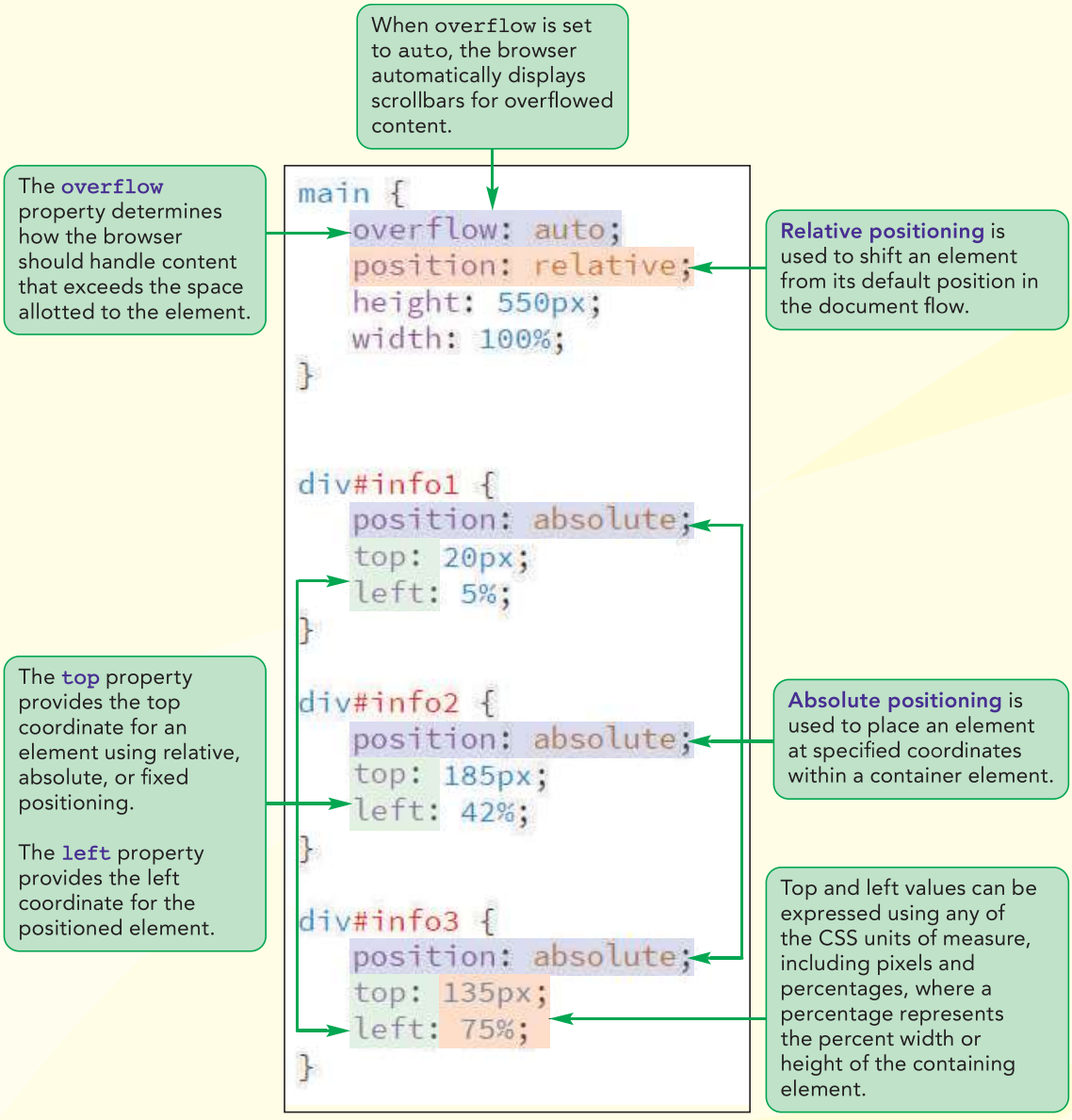
In the next session, you'll explore CSS positioning styles that allow you to place page elements anywhere within the rendered page.

REVIEW

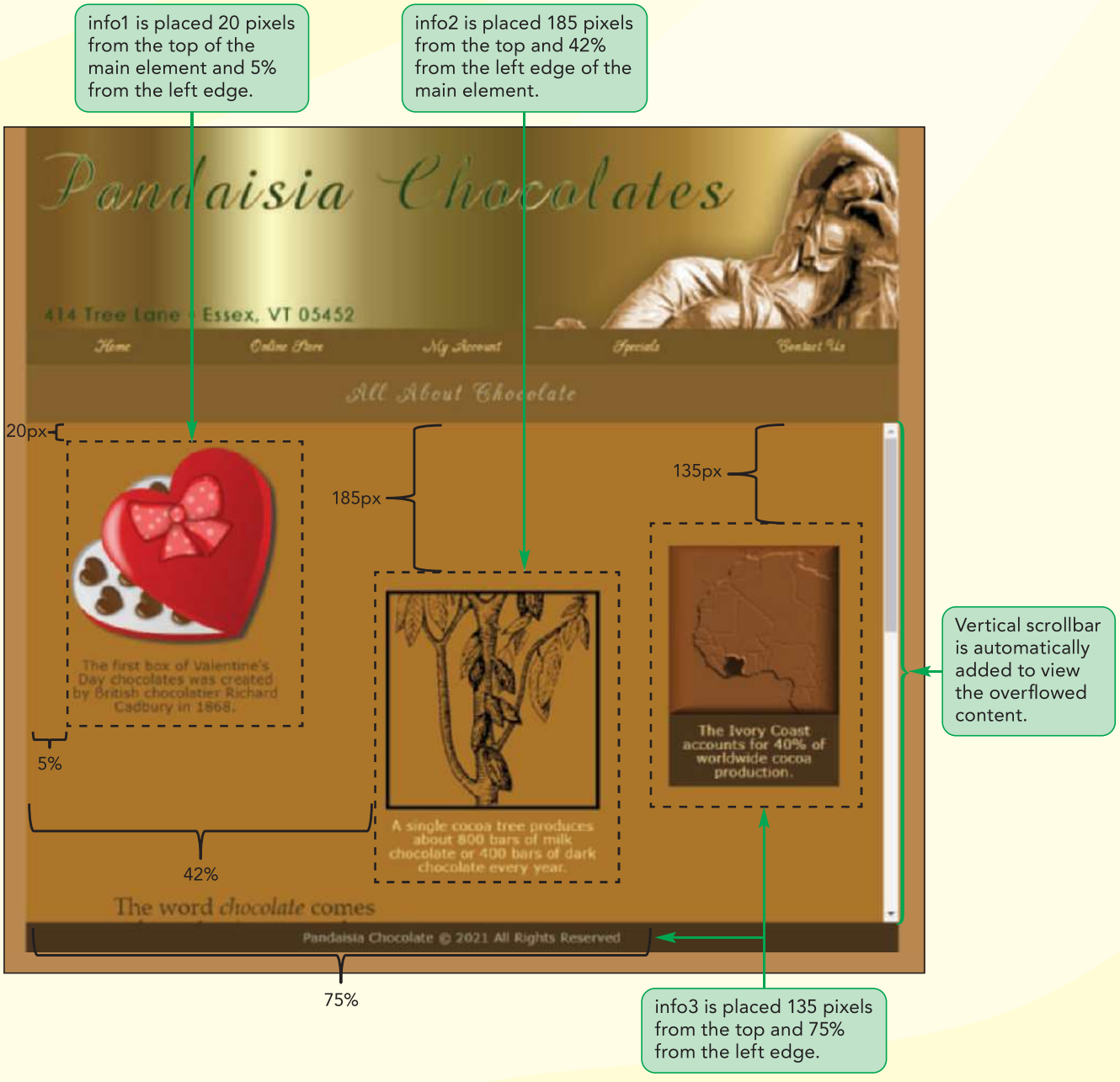
Session 3.2 Quick Check

1. To change an element into a grid container, apply the style:
   a. `display: grid-container;`
   b. `grid-template-columns: auto;`
   c. `display: grid;`
   d. All of the above
2. A fractional unit (`fr`) is:
   a. part of a pixel
   b. a fraction of an em unit
   c. a fraction of any fixed unit
   d. used to create elements that expand or contract to fill available space
3. To explicitly define the columns within a grid, use the CSS property:
   a. `grid-template-columns`
   b. `grid-columns`
   c. `columns`
   d. `grid-auto-columns`
4. To implicitly define the height of grid rows, use:
   a. `grid-template-rows`
   b. `grid-rows`
   c. `rows`
   d. `grid-auto-rows`
5. To position a grid item in the second row and cover the second and third column, apply the style(s):
   a.
   ```
   grid-row: 2;
   grid-column: 2/3;
   ```
   b.
   ```
   grid-row: 2;
   grid-column: 2/4;
   ```
   c.
   ```
   row: 2;
   column: 2/3;
   ```
   d.
   ```
   grid-row: 2;
   column-span: 2/2;
   ```
6. To define a grid layout with areas, use the property:
   a. `grid-areas`
   b. `grid-area`
   c. `grid-template-areas`
   d. `grid-areas-template`
7. To place an element in the grid area named "intro", apply the style:
   a. `grid-area: intro;`
   b. `grid-template-areas: "intro";`
   c. `area: intro;`
   d. All of the above
8. To set the space between grid columns to 15 pixels and the space between grid rows to 10 pixels, apply the style:
   a. `gap: 10px 15px;`
   b. `grid-gap: 10px 15px;`
   c. `grid-gap: 15px/10px;`
   d. `gap: 15px/10px;`
9. To horizontally center the content of a grid cell, apply the style:
   a. `align-content: center;`
   b. `align-self: center;`
   c. `justify-self: center;`
   d. `justify-content: center;`

Session 3.3 Visual Overview:



Layout with Positioning Styles



Positioning Objects

In the last session, you developed a layout in which page objects were strictly aligned according to the rows and columns of a grid. While a grid layout gives a page a feeling of uniformity and structure, it does limit your freedom to place objects at different locations within the page. In this session, you'll explore how to "break out" of the grid using the CSS positioning styles.

The CSS Positioning Styles

CSS supports several properties to place objects at specific coordinates within the page or within their container. To place an element at a specific position within its container, you use the following style properties

```
position: type;
top: value;
right: value;
bottom: value;
left: value;
```

where *type* indicates the kind of positioning applied to the element, and the `top`, `right`, `bottom`, and `left` properties indicate the coordinates of the top, right, bottom, and left edges of the element, respectively. The coordinates can be expressed in any of the CSS measuring units or as a percentage of the container's width or height.

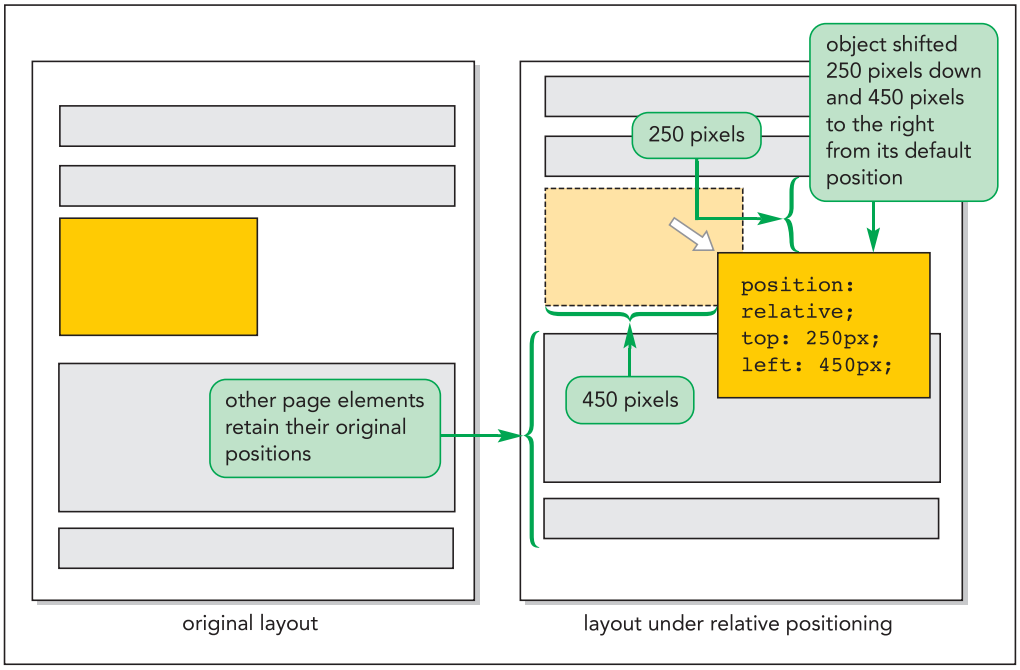
CSS supports five kinds of positioning: `static` (the default), `relative`, `absolute`, `fixed`, and `inherit`. In **static positioning**, the element is placed where it would have fallen naturally within the flow of the document. This is essentially the same as not using any CSS positioning at all. Browsers ignore any values specified for the `top`, `left`, `bottom`, or `right` properties under static positioning.

Relative Positioning

Relative positioning is used to nudge an element out of its normal position in the document flow. Under relative positioning, the `top`, `right`, `bottom`, and `left` properties indicate the extra space that is placed alongside the element as it is shifted into a new position. For example, the following style rule adds 250 pixels of space to the top of the element and 450 pixels to the left of the element, resulting in the element being shifted down and to the right (see Figure 3–60):

```
div {
   position: relative;
   top: 250px;
   left: 450px;
}
```

Figure 3–60 Moving an object using relative positioning



Note that the layout of the other page elements are not affected by relative positioning; they will still occupy their original positions on the rendered page, just as if the object had never been moved at all.

Relative positioning is sometimes used when the designer wants to "tweak" the page layout by slightly moving an object from its default location to a new location that fits the overall page design better. If no top, right, bottom, or left values are specified with relative positioning, their assumed values are 0 and the element will not be shifted at all.

Absolute Positioning

Absolute positioning places an element at specific coordinates within a container where the `top` property indicates the position of the element's top edge, the `right` property sets the position of the right edge, the `bottom` property sets the bottom edge position, and the `left` property sets the position of the left edge.

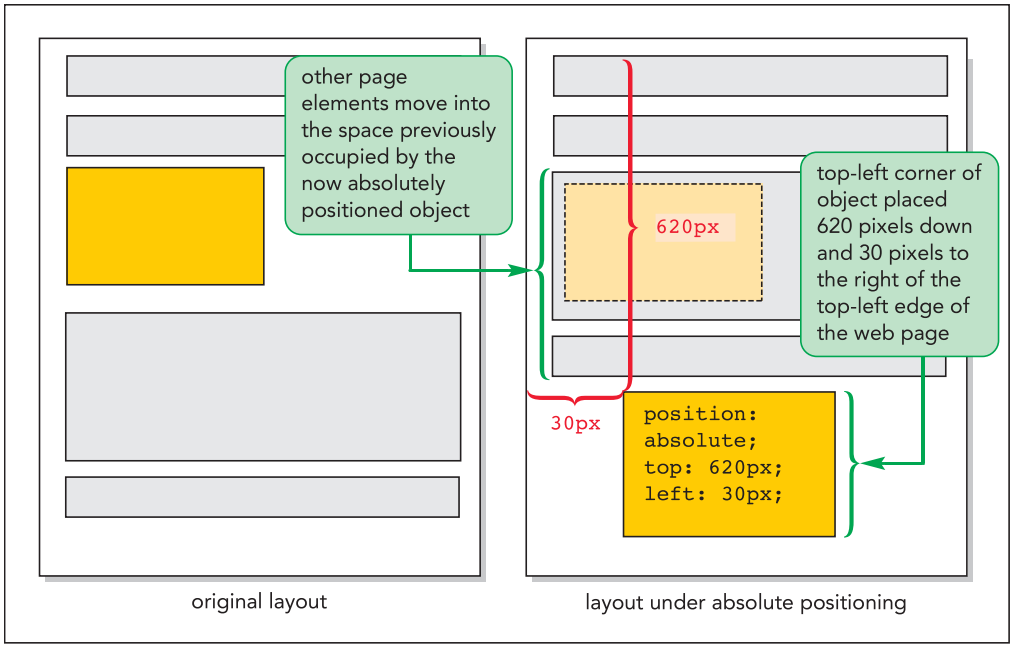
TIP

To place an element at the bottom right corner of its container, use absolute positioning with the right and bottom values set to 0 pixels.

For example, the following style rule places the `header` element 620 pixels from the top edge of its container and 30 pixels from the left edge (see Figure 3–61).

```
header {
   position: absolute;
   top: 620px;
   left: 30px;
}
```

Figure 3–61 Moving an object using absolute positioning



To place an object with absolute positioning, you use either the top/left coordinates or the bottom/right coordinates, but you don't use all four coordinates at the same time because that would confuse the browser. For example an object cannot be positioned along both the left and right edge of its container simultaneously.

As with floating an element, absolute positioning takes an element out of normal document flow with subsequent elements moving into the space previously occupied by the element. This can result in an absolutely positioned object overlapping other page elements.

TRY IT

You can explore positioning styles using the file demo_positioning.html from the html04 ▸ demo folder.

The interpretation of the coordinates of an absolutely positioned object are all based on the edges of the element's container. Thus the browser needs to "know" where the object's container is before it can absolutely position objects within it. If the container has been placed using a `position` property set to `relative` or `absolute`, the container's location is known and the coordinate values are based on the edges of the container. For example the following style rules place the `article` element at a coordinate that is 50 pixels from the top edge of the section element and 20 pixels from the left edge.

```
section {
   position: relative;
}
section > article {
   position: absolute;
   top: 50px;
   left: 20px;
}
```

Note that you don't have to define coordinates for the `section` element as long as you've set its position to relative.

The difficulty starts when the container has not been set using relative or absolute positioning. In that case, the browser has no context for placing an object within the container using absolute positioning. As a result, the browser must go up a level in the hierarchy of page elements, that is, to the container's container. If that container has been placed with absolute or relative positioning, then any object nested within it

can be placed with absolute positioning. For example, in the following style rule, the position of the `article` element is measured from the edges of the `body` element, not the `section` element:

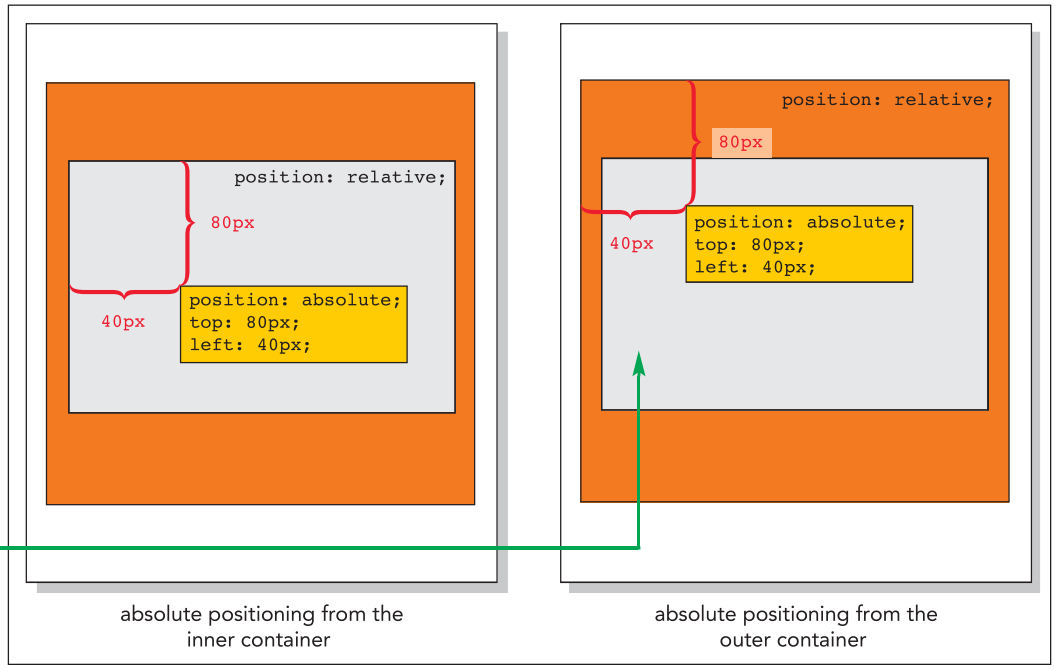
```
body {position: absolute;}

body > section {position: static;}

body > section > article {
   position: absolute;
   top: 50px;
   left: 20px;
}
```

TIP

If all of the objects within a container are placed using absolute positioning, the container will have no content and will collapse.

Proceeding in this fashion the browser will continue to go up the hierarchy of elements until it finds a container that has been placed with absolute or relative positioning or it reaches the root `html` element. If it reaches the `html` element, the coordinates of any absolutely positioned object are measured from the edges of the browser window itself. Figure 3–62 shows how the placement of the same object can differ based on which container supplies the context for the top and left values.

Figure 3–62 Context of the top and left coordinates



because the inner container has not been placed using relative or absolute positioning, the context shifts up the hierarchy to the outer container

Coordinates can be expressed in percentages as well as pixels. Percentages are used for flexible layouts in which the object should be positioned in relation to the width or height of its container. Thus, the following style rule places the `article` element halfway down and 30% to the right of the top-left corner of its container.

```
article {
   position: absolute;
   top: 50%;
   left: 30%;
}
```

As the container of the article changes in width or height, the article's position will automatically change to match.

Fixed and Inherited Positioning

When you scroll through a document in the browser window, the page content scrolls along. If you want to fix an object within the browser window so that it doesn't scroll, you can set its `position` property to `fixed`. For example, the following style rule keeps the `footer` element at a fixed location, 10 pixels up from the bottom of the browser window:

```
footer {
   position: fixed;
   bottom: 10px;
}
```

Note that a fixed object might cover up other page content, so you should use it with care in your page design.

Finally, you can set the `position` property to `inherit` so that an element inherits the position value of its parent element.

REFERENCE

Positioning Objects with CSS

- To shift an object from its default position, use the properties

```
position: relative;
top: value;
left: value;
bottom: value;
right: value;
```

where *value* is the distance in one of the CSS units of measure that the object should be shifted from the corresponding edge of its container.

- To place an object at a specified coordinate within its container, use the properties

```
position: absolute;
top: value;
left: value;
bottom: value;
right: value;
```

where *value* is a distance in one of the CSS units of measure or a percentage of the container's width or height.

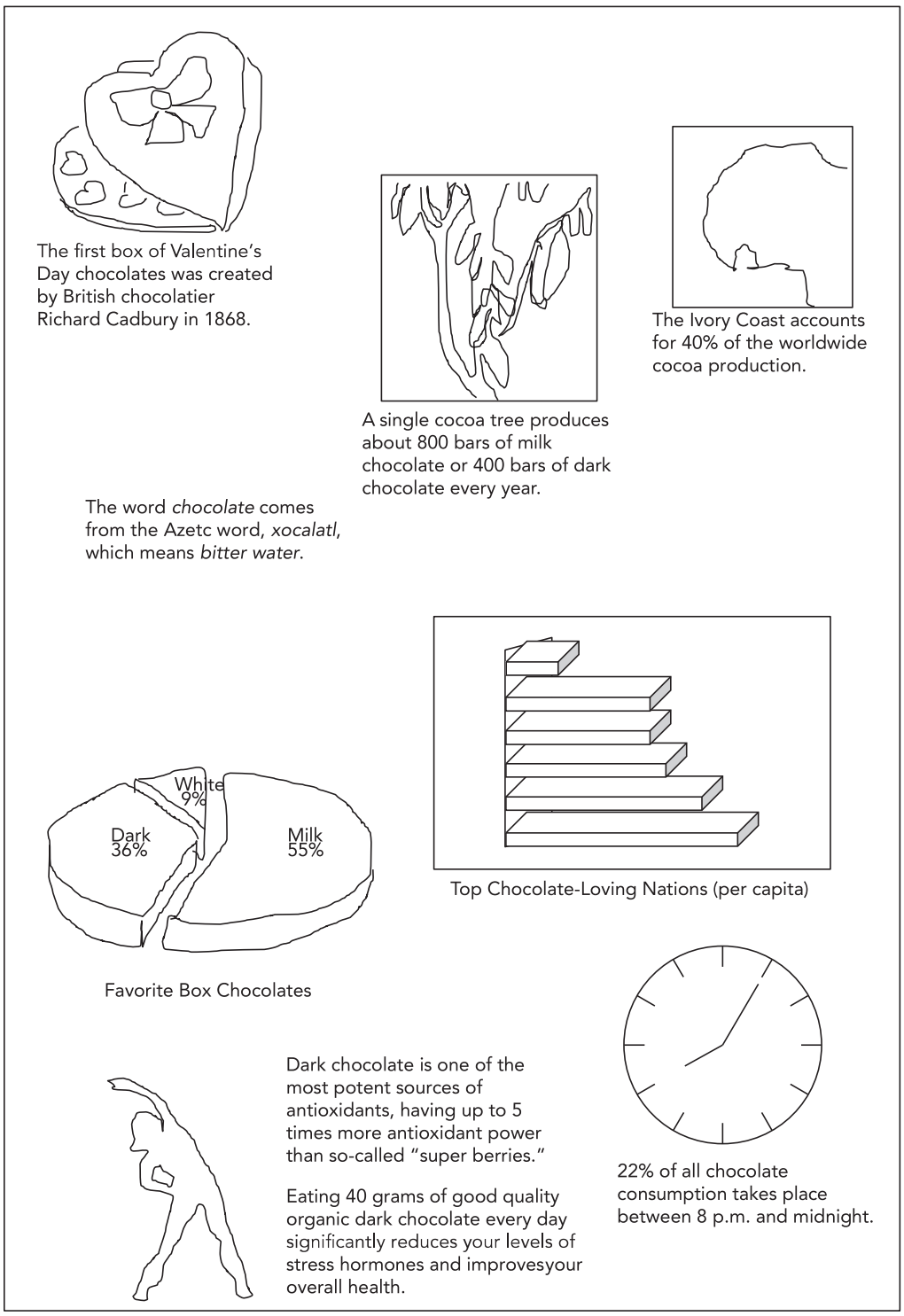
- To fix an object within the browser window so that it does not scroll with the rest of the document content, use the property

```
position: fixed;
```

Using the Positioning Styles

Anne wants you to work on the layout for a page that contains an infographic on chocolate. She sketched the layout of the infographic page, as shown in Figure 3–63.

Figure 3–63 Proposed layout of the chocolate infographic



Because the placement of the text and figures do not line up nicely within a grid, you'll position each graphic and text box using the CSS positioning styles. Anne has already created the content for this page and written the style sheets to format the appearance of the infographic. You will write the style sheet to layout the infographic contents using the CSS positioning styles.

To open the infographic file:

1. Use your editor to open the **pc_info_txt.html** file from the html03 ▸ tutorial folder. Enter ***your name*** and ***the date*** in the comment section of the file and save the document as **pc_info.html**.
2. Directly after the `title` element, insert the following `link` elements to attach the file to the pc_reset.css, pc_styles3.css, and pc_info.css style sheets.

```
<link href="pc_reset.css" rel="stylesheet" />
<link href="pc_styles3.css" rel="stylesheet" />
<link href="pc_info.css" rel="stylesheet" />
```

3. Take some time to study the structure and content of the pc_info.html document. Note that Anne has placed eight information graphics, each within a separate `div` element with a class name of infobox and an id name ranging from info1 to info8.
4. Close the file, saving your changes.

Next, you'll start working on the pc_info.css file, which will contain the positioning and other design styles for the objects in the infographic. You will begin by formatting the `main` element, which contains the infographics. Because you'll want the position of each infographic to be measured from the top-left corner of this container, you will place the `main` element with relative positioning and extend the height of the container to 1400 pixels so that it can contain all eight of the graphic elements.

To format the main element:

1. Use your editor to open the **pc_info_txt.css** file from the html03 ▸ tutorial folder. Enter ***your name*** and ***the date*** in the comment section of the file and save the document as **pc_info.css**.

When you want to position objects in an exact or absolute position within a container, set the `position` property of the container to `relative`.

2. Go to the Main Styles section and insert the following style rule to format the appearance of the `main` element:

```
main {
   position: relative;
   height: 1400px;
   width: 100%;
}
```

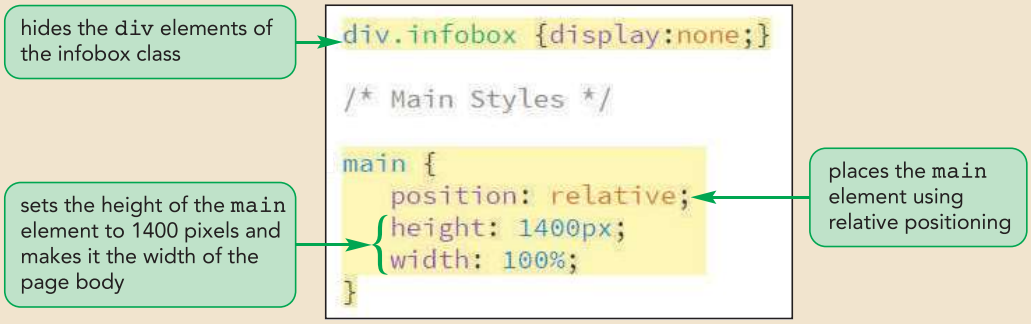
It will be easier to see the effect of placing the different `div` elements if they are not displayed until you are ready to position them. Add a rule to hide the `div` elements, then as you position each element, you can add a style rule to redisplay it.

3. Directly before the Main Styles section, insert the following style rule to hide all of the infoboxes:

```
div.infobox {display:none;}
```

Figure 3–64 highlights the newly added code in the style sheet.

Figure 3–64 Setting the display styles of the main element



4. Save your changes to the file and then open the pc_info.html file in your browser. Verify that the browser shows an empty box, about 1400 pixels high, where the infographic will be placed.

Next, you will add a style rule for all of the information boxes so that they are placed within the `main` element using absolute positioning.

To position the information boxes:

1. Return to the **pc_info.css** file in your editor and scroll down to the Infographic Styles section.
2. Add the following style rule to set the position type of all of the information boxes.

```
div.infobox {
   position: absolute;
}
```

3. Within the First Infographic section, add the following style rule to position the first information box 20 pixels from the top edge of its container and 5% from the left edge.

```
div#info1 {
   display: block;
   top: 20px;
   left: 5%;
}
```

Note that we set the `display` property to `block` so that the first information box is no longer hidden on the page. Figure 3–65 highlights the style rules for all of the information boxes and the placement of the first information box.

Figure 3–65 Placing the first information box

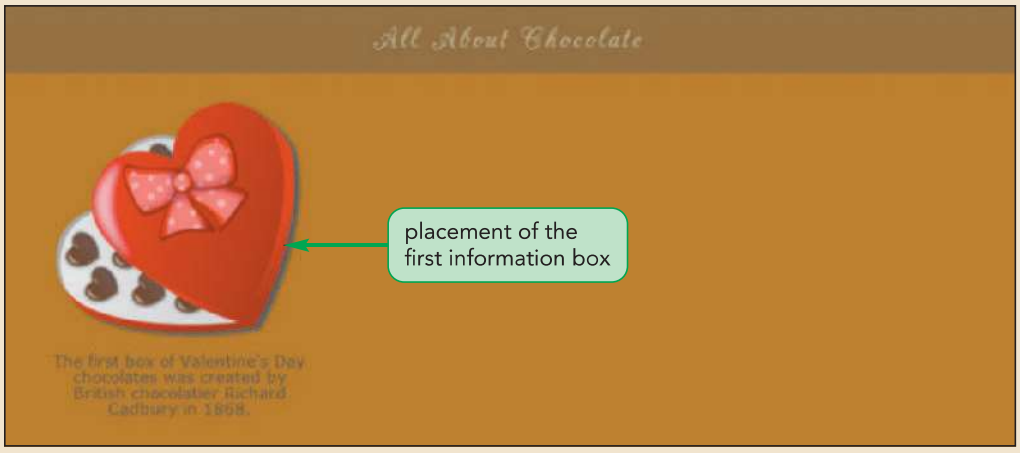
```
/* Infographic Styles */

div.infobox {
   position: absolute;
}

/* First Infographic */

div#info1 {
   display: block;
   top: 20px;
   left: 5%;
}
```

places every information box using absolute positioning

places the first box 20 pixels from the top edge of the main element and 5% from the left

4. Save your changes to the file and then reload the pc_info.html file in your browser. Figure 3–66 shows the placement of the first information box.

Figure 3–66 Appearance of the first information box



Now place the second and third information boxes.

To place the next two boxes:

1. Return to the **pc_info.css** file in your editor and go to the Second Infographic section.
2. Add the following style rule to place the second box 185 pixels down from the top of the container and 42% from the left edge.

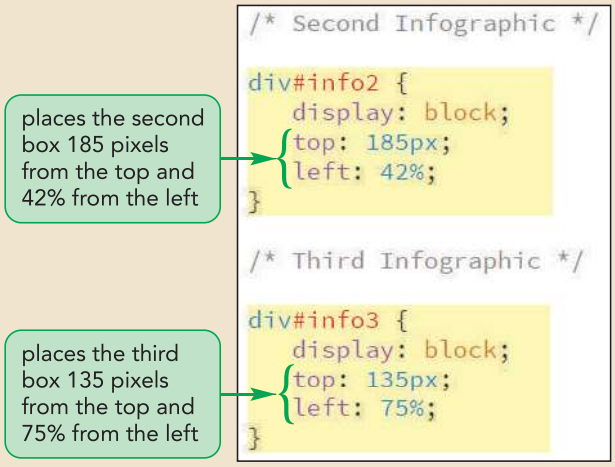
```
div#info2 {
   display: block;
   top: 185px;
   left: 42%;
}
```

3. Within the Third Infographic section insert the following style rule to place the third box 135 pixels from the top edge and 75% of the width of its container from the left edge.

```
div#info3 {
   display: block;
   top: 135px;
   left: 75%;
}
```

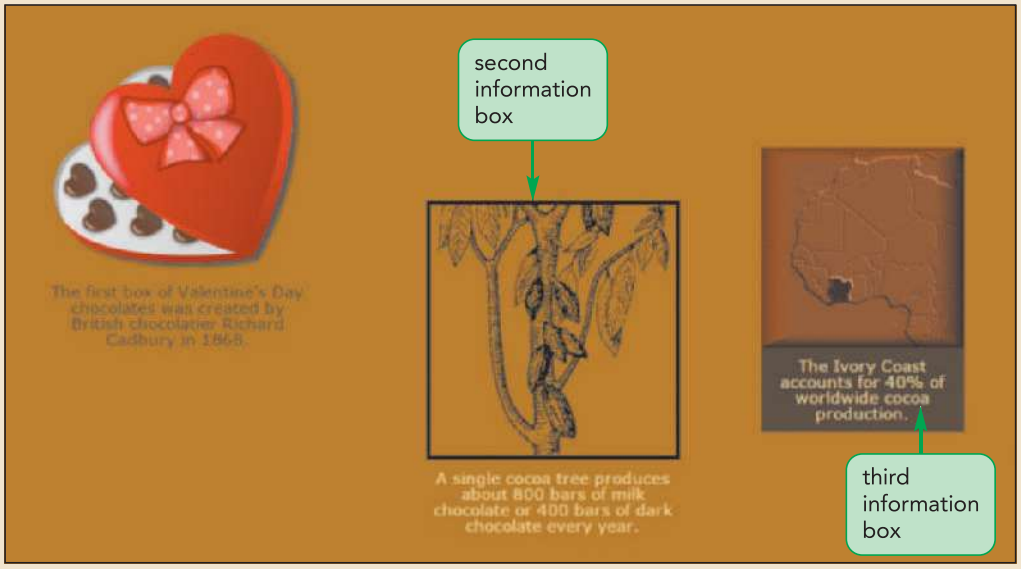
Figure 3–67 highlights the style rules to position the second and third information boxes.

Figure 3–67 Positions of the second and third boxes



4. Save your changes to the file and reload pc_info.html in your browser. Figure 3–68 shows the placement of the first three information boxes.

Figure 3–68 Placement of the first three boxes



Place the next three information boxes.

To place the next three boxes:

1. Return to the **pc_info.css** file in your editor, go to the Fourth Infographic section and place the fourth box 510 pixels from the top edge and 8% from the left edge.

```
div#info4 {
   display: block;
   top: 510px;
   left: 8%;
}
```

2. Add the following style rule to the Fifth Infographic section to position the fifth box:

```
div#info5 {
   display: block;
   top: 800px;
   left: 3%;
}
```

3. Add the following style rule to the Sixth Infographic section to position the sixth box:

```
div#info6 {
   display: block;
   top: 600px;
   left: 48%;
}
```

Figure 3–69 highlights the positioning styles for the fourth, fifth, and sixth information boxes.

Figure 3–69 Positions of the fourth, fifth, and sixth boxes

places the fourth box 510 pixels from the top and 8% from the left

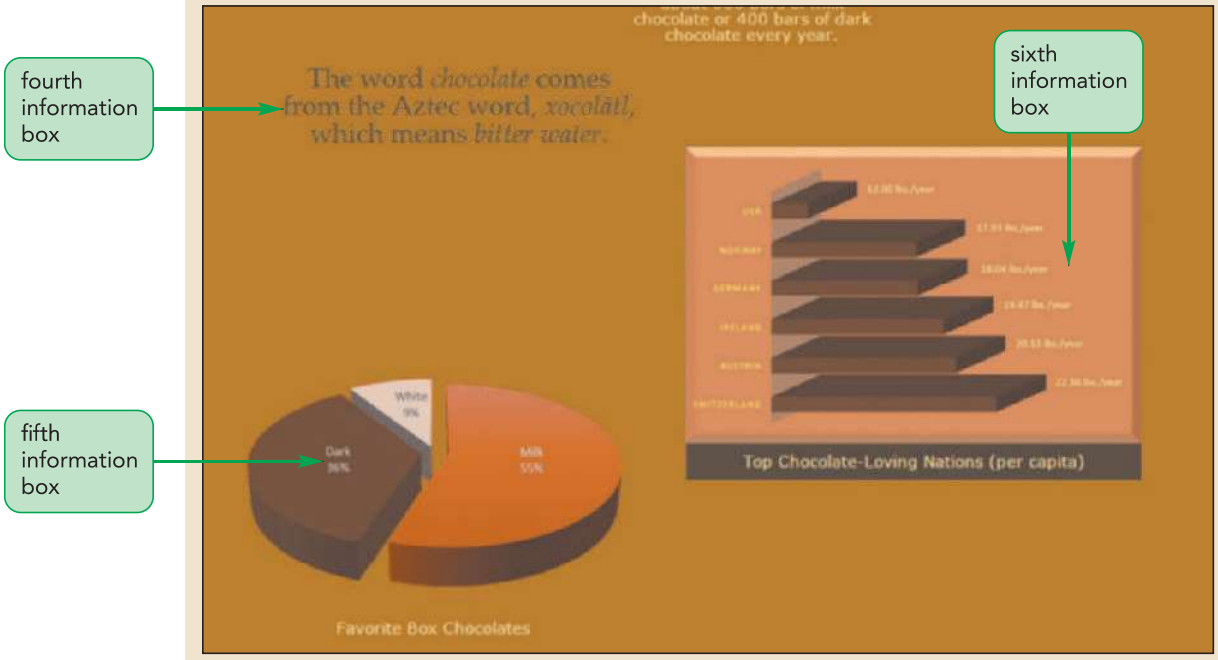
places the fifth box 800 pixels from the top and 3% from the left

places the sixth box 600 pixels from the top and 48% from the left

```
/* Fourth Infographic */

div#info4 {
   display: block;
   top: 510px;
   left: 8%;
}

/* Fifth Infographic */

div#info5 {
   display: block;
   top: 800px;
   left: 3%;
}

/* Sixth Infographic */

div#info6 {
   display: block;
   top: 600px;
   left: 48%;
}
```

4. Save your changes to the file and reload pc_info.html in your browser. Figure 3–70 shows the revised layout of the infographic.

Figure 3–70 Placement of the next three boxes



Complete the layout of the infographic by placing the final two boxes on the page.

To place the last two boxes:

1. Return to the **pc_info.css** file in your editor, go to the Seventh Infographic section and insert the following style rules:

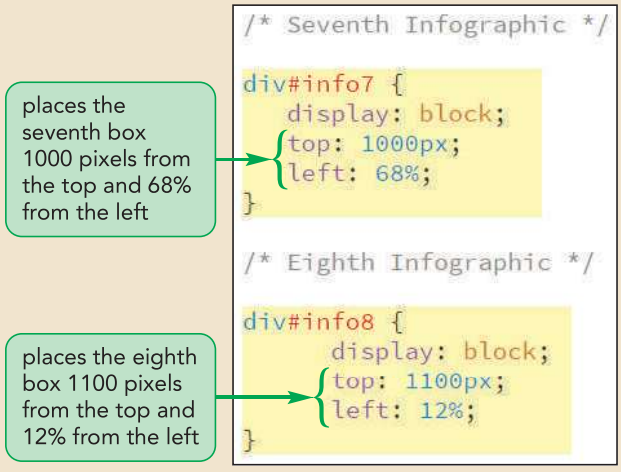
```
div#info7 {
   display: block;
   top: 1000px;
   left: 68%;
}
```

2. Add the following style rules to the Eighth Infographic section:

```
div#info8 {
   display: block;
   top: 1100px;
   left: 12%;
}
```

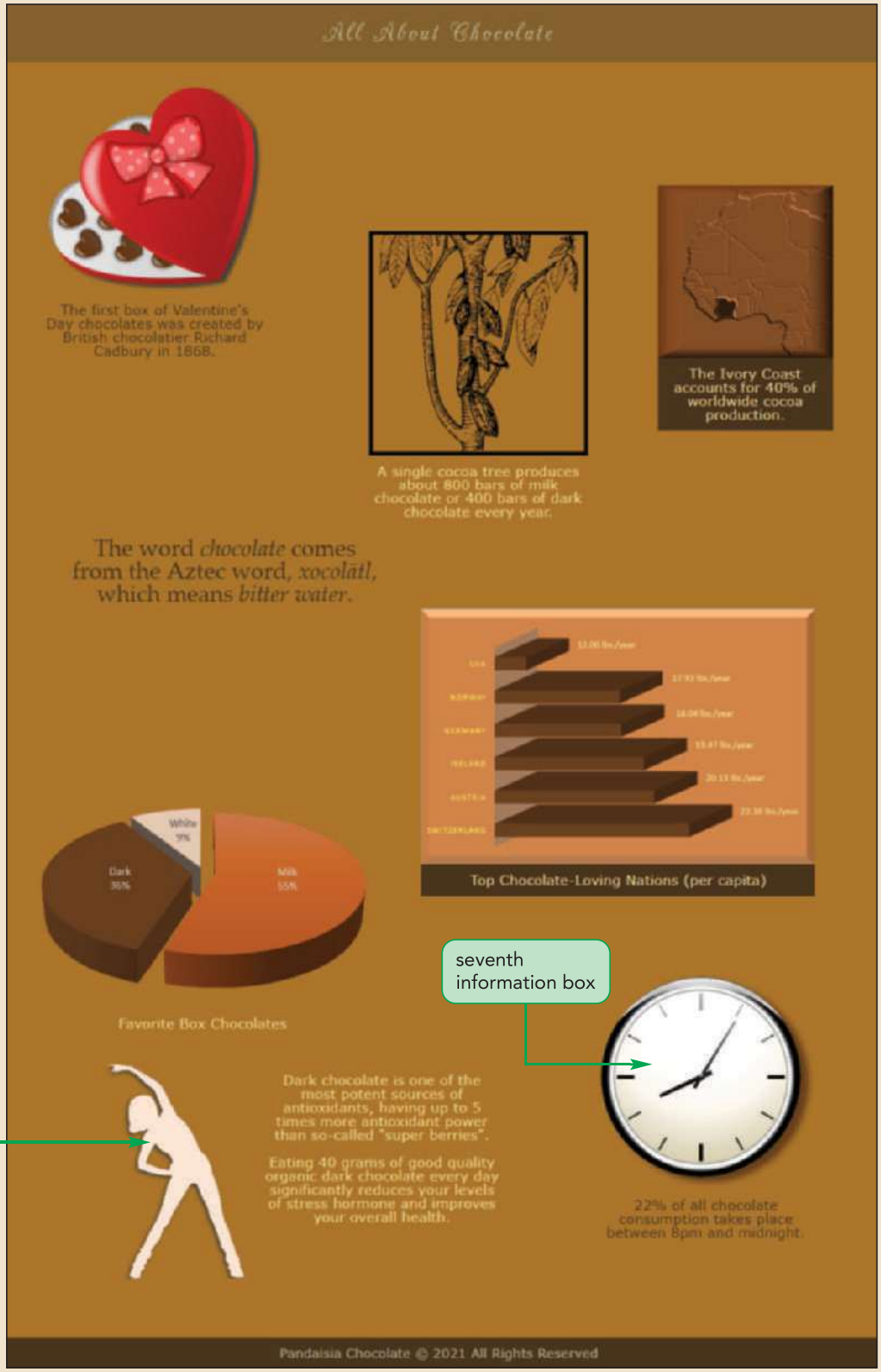
Figure 3–71 highlights the style rules for the seventh and eighth information boxes.

Figure 3–71 Positioning the seventh and eighth boxes



3. Scroll up to before the Main Styles section and delete the style rule `div.infobox {display: none;}` because you no longer need to hide any information boxes.
4. Save your changes to the file and reload pc_info.html in your browser. Figure 3–72 show the complete layout of the eight boxes in the infographic.

Figure 3–72 Final layout of the infographic



Anne likes the appearance of the infographic, but she is concerned about its length. She would like you to reduce the height of the infographic so that it appears within the boundaries of the browser window. This change will create overflow because the content is longer than the new height. You will read more about overflow and how to handle it now.

INSIGHT

Creating an Irregular Line Wrap

Many desktop publishing and word-processing programs allow designers to create irregular line wraps in which the text appears to flow tightly around an image. This is not easily done in a web page layout because all images appear as rectangles rather than as irregularly shaped objects. However, with the aid of a graphics package, you can simulate an irregularly shaped image.

The trick is to use your graphics package to slice the image horizontally into several pieces and then crop the individual slices to match the edge of the image you want to display. Once you've edited all of the slices, you can use CSS to stack the separate slices by floating them on the left or right margin, displaying each slice only after the previous slice has been cleared. For example, the following style rule stacks all inline images that belong to the "slice" class on the right margin:

```
img.slice {
   clear: right;
   float: right;
   margin-top: 0px;
   margin-bottom: 0px;
}
```

Now any text surrounding the stack of images will tightly match the image's boundary, creating the illusion of an irregular line wrap. Note that you should always set the top and bottom margins to 0 pixels so that the slices join together seamlessly.

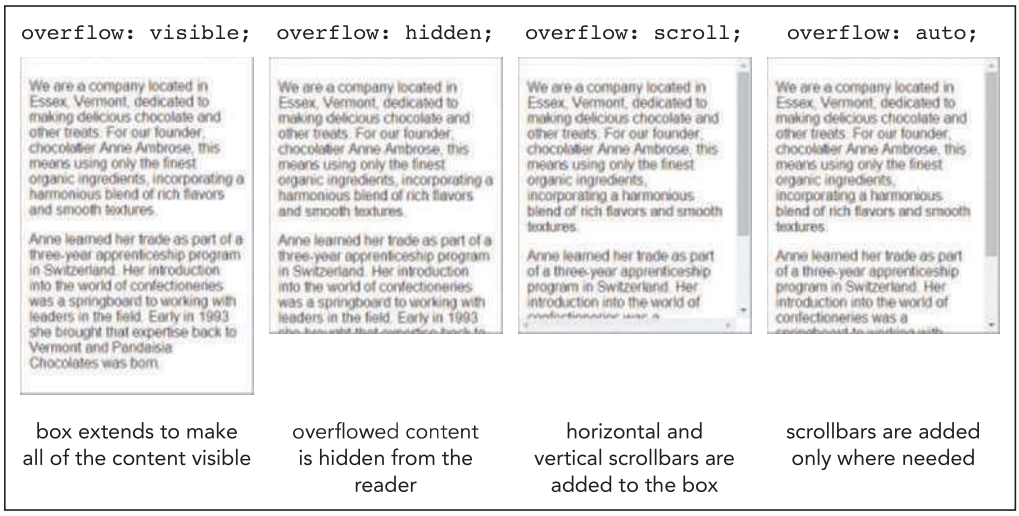
Handling Overflow

The infographic is long because it displays several information boxes. If you reduce the height of the infographic you run the risk of cutting off several of the boxes that will no longer fit within the reduced infographic. However you can control how your browser handles this excess content using the following `overflow` property

```
overflow: type;
```

where *type* is `visible` (the default), `hidden`, `scroll`, or `auto`. A value of `visible` instructs browsers to increase the height of an element to fit the overflow content. The `hidden` value keeps the element at the specified height and width, but cuts off excess content. The `scroll` value keeps the element at the specified dimensions, but adds horizontal and vertical scroll bars to allow users to scroll through the overflowed content. Finally, the `auto` value keeps the element at the specified size, adding scroll bars only as they are needed. Figure 3–73 shows examples of the effects of each overflow value on content that is too large for its space.

Figure 3–73 Values of the overflow property



CSS also provides the `overflow-x` and `overflow-y` properties to handle overflow specifically in the horizontal and vertical directions.

REFERENCE

Working with Overflow

- To specify how the browser should handle content that overflows the element's boundaries, use the property

  `overflow: type;`

  where *type* is `visible` (the default), `hidden`, `scroll`, or `auto`.

You decide to limit the height of the infographic to 450 pixels and to set the `overflow` property to `auto` so that browsers displays scroll bars as needed for the excess content.

To apply the `overflow` property:

1. Return to the **pc_info.css** file in your editor and go to the Main Styles section.
2. Within the style rule for the `main` selector, insert the property **`overflow: auto;`**.
3. Reduce the height of the element from 1400px to **450px**.

   Figure 3–74 highlights the revised code in the style rule.

Figure 3–74 Setting the overflow property

displays scrollbars if the content overflows the allotted height

```
/* Main Styles */

main {
   overflow: auto;
   position: relative;
   height: 450px;
   width: 100%;
}
```

sets the height of the infographic to 450 pixels

4. Close the file, saving your changes.
5. Reload the pc_info.html file in your browser. As shown in Figure 3–75, the height of the infographic has been reduced to 450 pixels and scrollbars have been added that you can use to view the entire infographic.

Figure 3–75 Final layout of the infographic page

height of the infographic set at 450 pixels

scrollbar automatically added to view the infographic content

6. Close any open files now.

INSIGHT

Managing White Space with CSS

Scroll bars for overflow content are usually placed vertically so that you scroll down to view the extra content. In some page layouts, however, you may want to view content in a horizontal rather than a vertical direction. You can accomplish this by adding the following style properties to the element:

```
overflow: auto;
white-space: nowrap;
```

The `white-space` property defines how browsers should handle white space in the rendered document. The default is to collapse consecutive occurrences of white space into a single blank space and to automatically wrap text to a new line if it extends beyond the width of the container. However, you can set the `white-space` property of the element to `nowrap` to keep inline content on a single line, preventing line wrapping. With the content thus confined to a single line, browsers will display only horizontal scroll bars for the overflow content. Other values of the `white-space` property include `normal` (for default handling of white space), `pre` (to preserve all white space from the HTML file), and `pre-wrap` (to preserve white space but to wrap excess content to a new line).

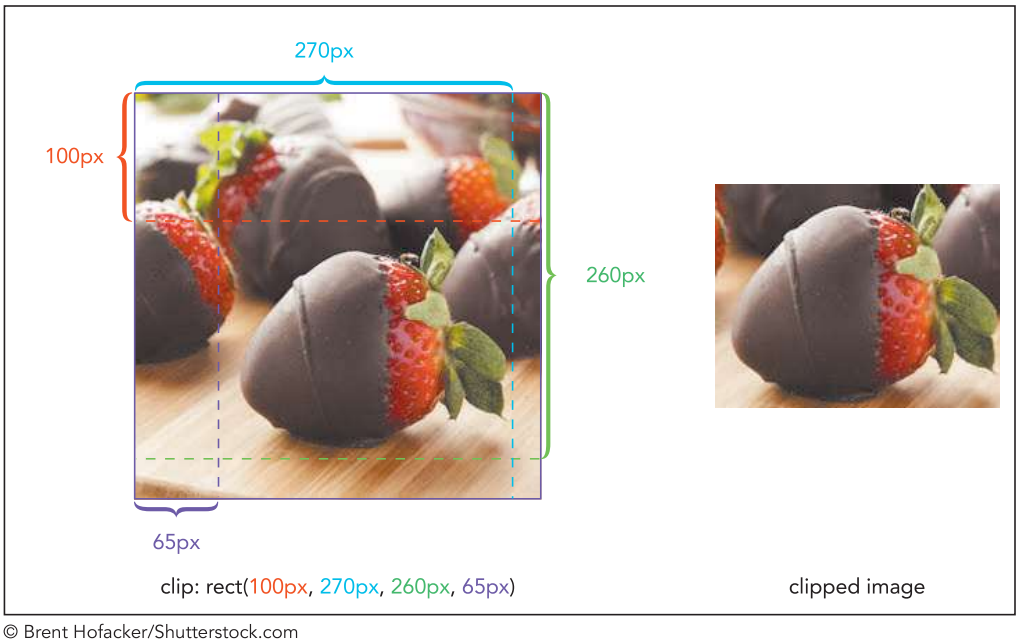
Clipping an Element

Closely related to the `overflow` property is the `clip` property, which defines a rectangular region through which an element's content can be viewed. Anything that lies outside the boundary of the rectangle is hidden. The syntax of the `clip` property is

```
clip: rect(top, right, bottom, left);
```

where *top*, *right*, *bottom*, and *left* define the coordinates of the clipping rectangle. For example, a clip value of rect(100px, 270px, 260px, 65px) defines a clip region whose top and bottom boundaries are 100 and 260 pixels from the top edge of the element, and whose right and left boundaries are 270 and 65 pixels from the element's left edge. See Figure 3–76.

Figure 3–76 **Clipping an image**



The top, right, bottom, and left values also can be set to `auto`, which matches the specified edge of the clipping region to the edge of the parent element. A clip value of rect(10, auto, 125, 75) creates a clipping rectangle whose right edge matches the right edge of the parent element. To remove clipping completely, apply the style `clip: auto`. Clipping can only be applied when the object is placed using absolute positioning.

REFERENCE

Clipping Content

- To clip an element's content, use the property

```
clip: rect(top, right, bottom, left);
```

where *top*, *right*, *bottom*, and *left* define the coordinates of the clipping rectangle.

- To remove clipping for a clipped object, use

```
clip: auto;
```

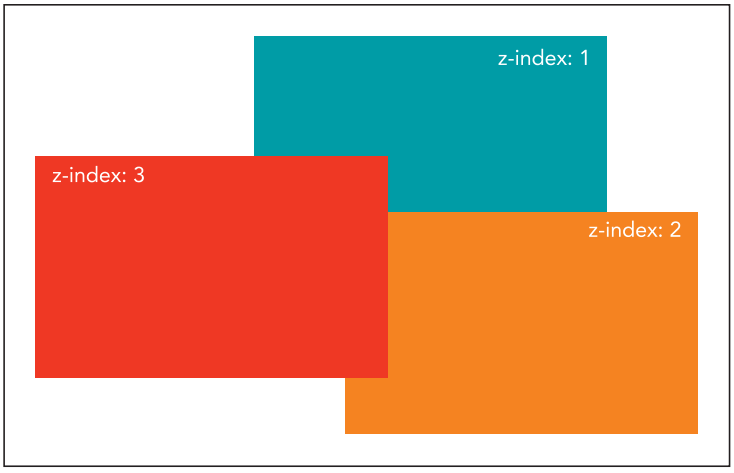
Stacking Elements

Positioning elements can sometimes lead to objects that overlap each other. By default, elements that are loaded later by the browser are displayed on top of elements that are loaded earlier. In addition, elements placed using CSS positioning are stacked on top of elements that are not. To specify a different stacking order, use the following `z-index` property:

```
z-index: value;
```

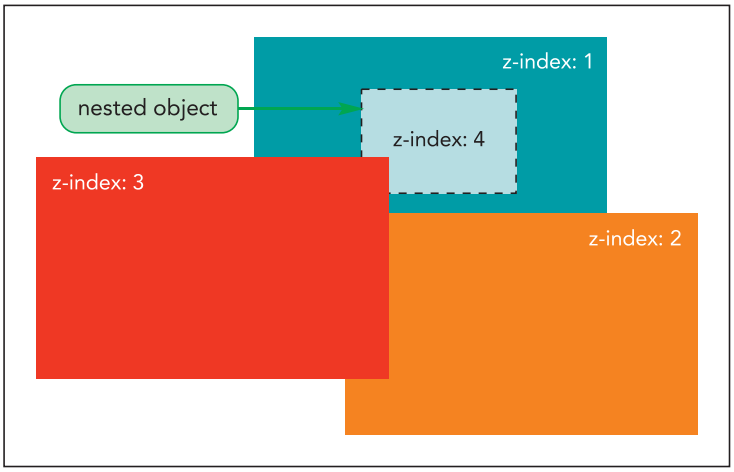
where *value* is a positive or negative integer, or the keyword `auto`. As shown in Figure 3–77, objects with the highest z-index values are placed on top of other page objects. A value of `auto` stacks the objects using the default rules.

Figure 3–77 Using the z-index property to stack elements



The `z-index` property works only for elements that are placed with absolute positioning. Also, an element's z-index value determines its position relative only to other elements that share a common parent; the style has no impact when applied to elements with different parents. Figure 3–78 shows a layout in which the object with a high z-index value of 4 is still covered because it is nested within another object that has a low z-index value of 1.

Figure 3–78 Stacking nested objects



You do not need to include the `z-index` property in your style sheet because none of the elements in the infographic page are stacked upon another.

PROSKILLS

Problem Solving: Principles of Design

Good web page design is based on the same common principles found in other areas of art, which include balance, unity, contrast, rhythm, and emphasis. A pleasing layout involves the application of most, if not all, of these principles, which are detailed below:

- **Balance** involves the distribution of elements. It's common to think of balance in terms of **symmetrical balance**, in which similar objects offset each other like items on a balance scale; but you often can achieve more interesting layouts through asymmetrical balance, in which one large page object is balanced against two or more smaller objects.
- **Unity** is the ability to combine different design elements into a cohesive whole. This is accomplished by having different elements share common colors, font styles, and sizes. One way to achieve unity in a layout is to place different objects close to each other, forcing your viewers' eyes to see these items as belonging to a single unified object.
- **Contrast** consists of the differences among all of the page elements. To create an effective design, you need to vary the placement, size, color, and general appearance of the objects in the page so that your viewers' eyes aren't bored by the constant repetition of a single theme.
- **Rhythm** is the repetition or alteration of a design element in order to provide a sense of movement, flow, and progress. You can create rhythm by tiling the same image horizontally or vertically across the page, by repeating a series of elements that progressively increase or decrease in size or spacing, or by using elements with background colors of the same hue but that gradually vary in saturation or lightness.
- **Emphasis** involves working with the focal point of a design. Your readers need a few key areas to focus on. It's a common design mistake to assign equal emphasis to all page elements. Without a focal point, there is nothing for your viewers' eyes to latch onto. You can give a page element emphasis by increasing its size, by giving it a contrasting color, or by assigning it a prominent position in the page.

Designers usually have an intuitive sense of what works and what doesn't in page design, though often they can't say why. These design principles are important because they provide a context in which to discuss and compare designs. If your page design doesn't feel like it's working, evaluate it in light of these principles to identify where it might be lacking.

Anne is pleased with the final design of the infographic page and all of the other pages you've worked on. She'll continue to develop the website and test her page layouts under different browsers and screen resolutions. She'll get back to you with future projects as she continues the redesign of the Pandaisia Chocolates website.

REVIEW

Session 3.3 Quick Check

1. To shift an object from its default placement in the document flow but keep it within the document flow, use:
   a. absolute positioning
   b. relative positioning
   c. fixed positioning
   d. static positioning
2. Provide a style to shift rule to shift an article element 15 pixels to the left of its default position in the document flow.
   a.
```
article {
   position: absolute;
   left: 15px;
}
```
   b.
```
article {
   position: relative;
   left: 15px;
}
```
   c.
```
article {
   position: absolute;
   left: -15px;
}
```
   d.
```
article {
   position: relative;
   left: -15px;
}
```
3. Provide a style to place an article element 15 pixels up from the top edge of its container element.
   a.
```
article {
   position: absolute;
   top: 15px;
}
```
   b.
```
article {
   position: relative;
   top: 15px;
}
```
   c.
```
article {
   position: absolute;
   top: -15px;
}
```
   d.
```
article {
   position: relative;
   top: -15px;
}
```
4. To place an object using absolute positioning within its container, the container:
   a. must also have absolute positioning
   b. must have absolute or relative positioning
   c. must have statistic positioning
   d. must not have any position property value