

## 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

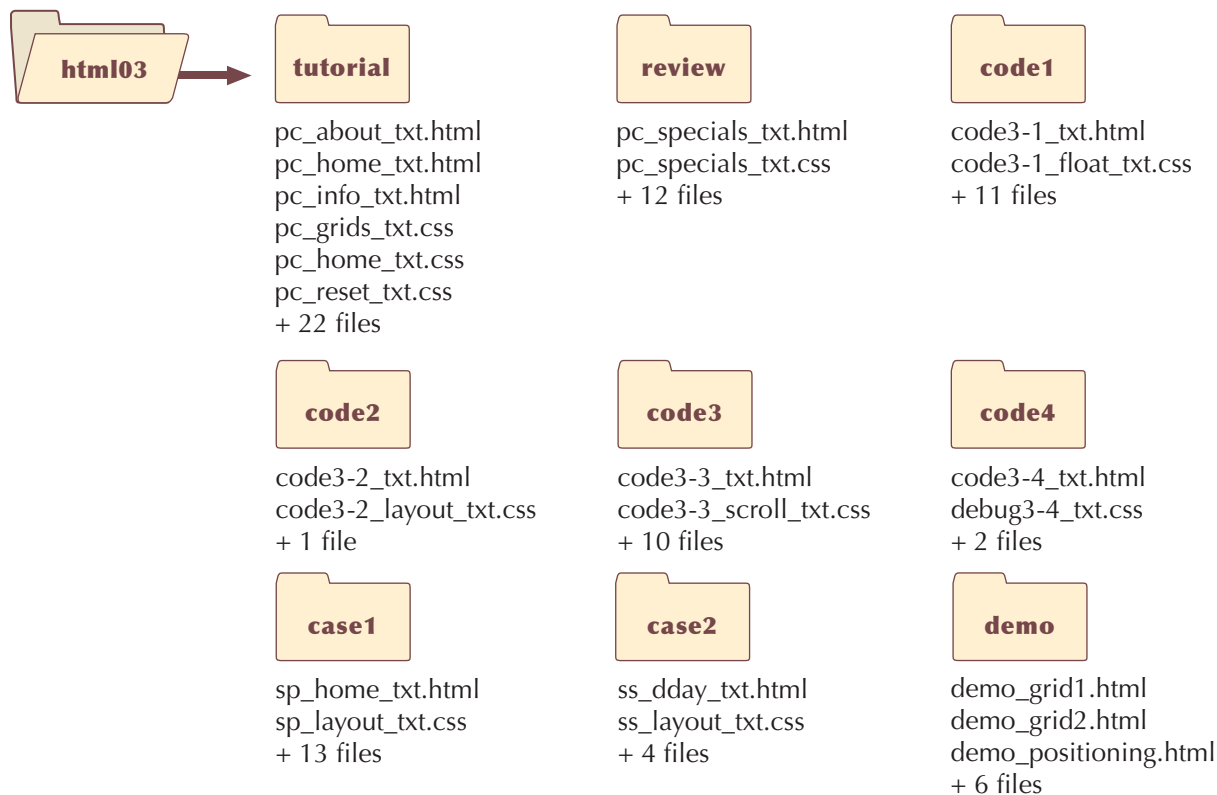
# Designing a Page Layout

## Creating a Website for a Chocolatier

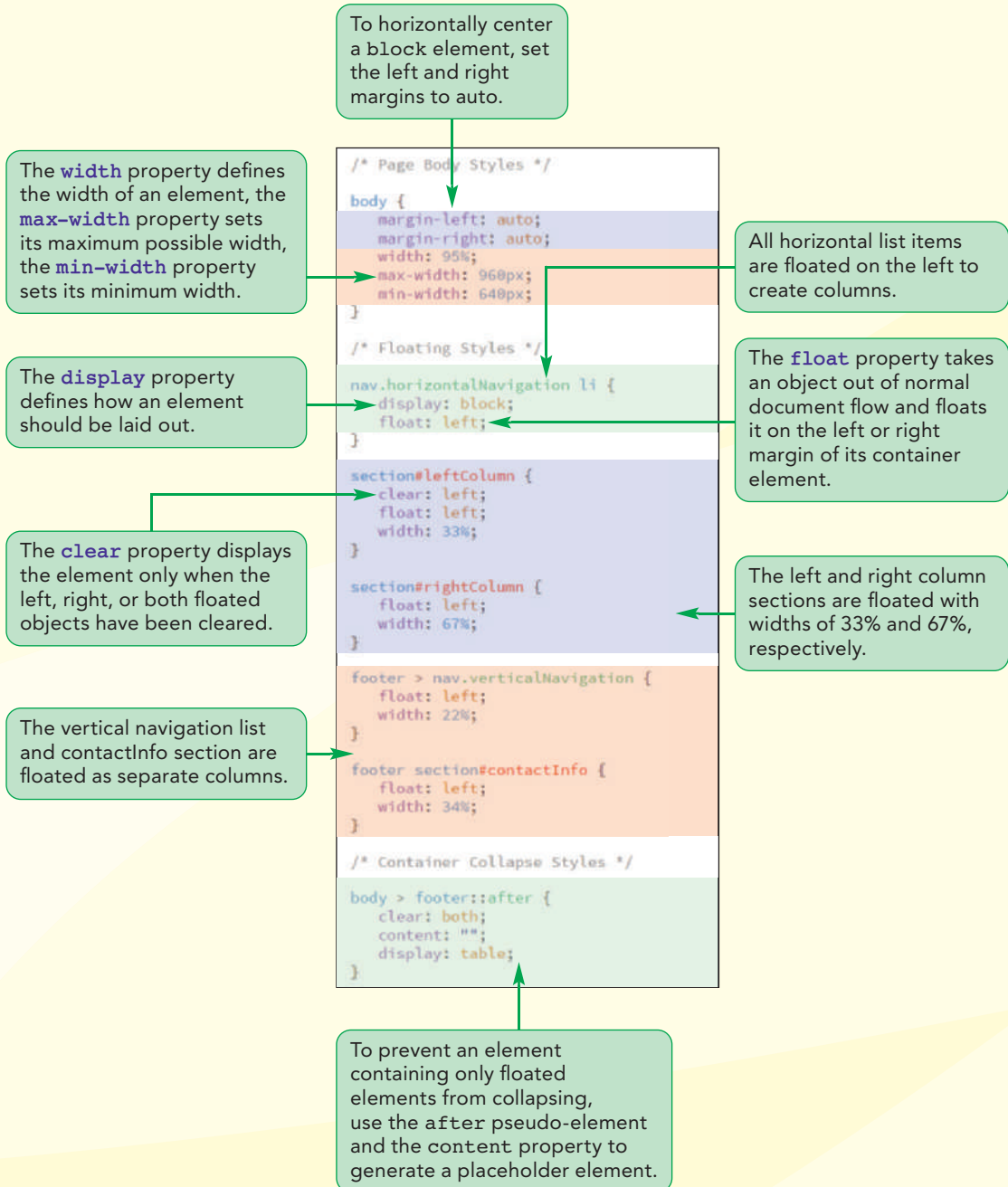
### Case | *Pandaisia Chocolates*

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



# Session 3.1 Visual Overview:



# Page Layout with Floating Elements

Page body is horizontally centered within the browser window.



Horizontal list items are floated into separate columns.

Left and right sections are floated into separate columns.

The contents of the page footer are floated into separate columns.

© Brenda Carson/Shutterstock.com;  
 © Brent Hofacker/Shutterstock.com;  
 © Jim Bowie/Shutterstock.com;  
 © wacomka/Shutterstock.com;  
 © Shebeko/Shutterstock.com;  
 Source: Facebook;  
 Source: Twitter, Inc.

## 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:

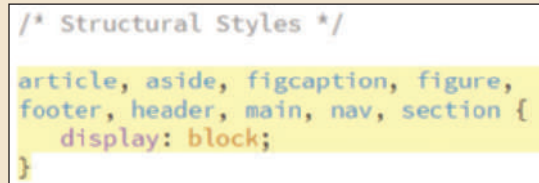
1. Use the text editor or HTML editor of your choice to open the `pc_reset_txt.css` file from the `html03` ► tutorial folder. Enter **your name** and **the date** in the comment section of the file and save the document as `pc_reset.css`.
2. Within the Structural Styles section, insert the following 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;
}
```

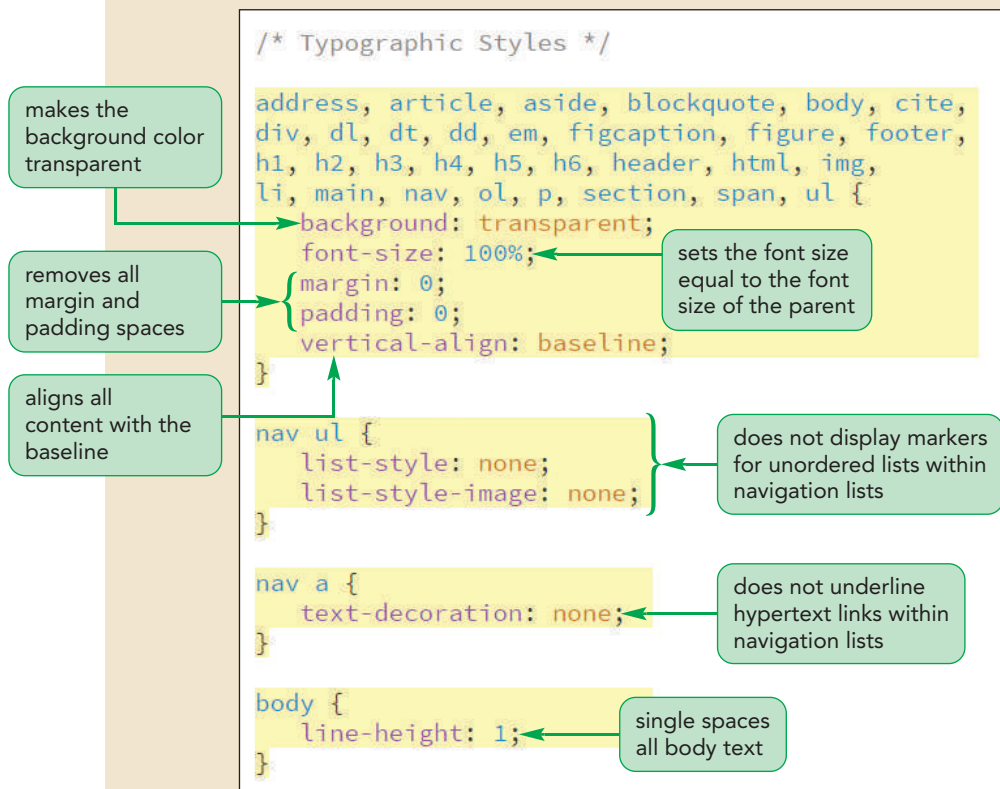
```
nav a {
  text-decoration: none;
}
```

- 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.

**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.

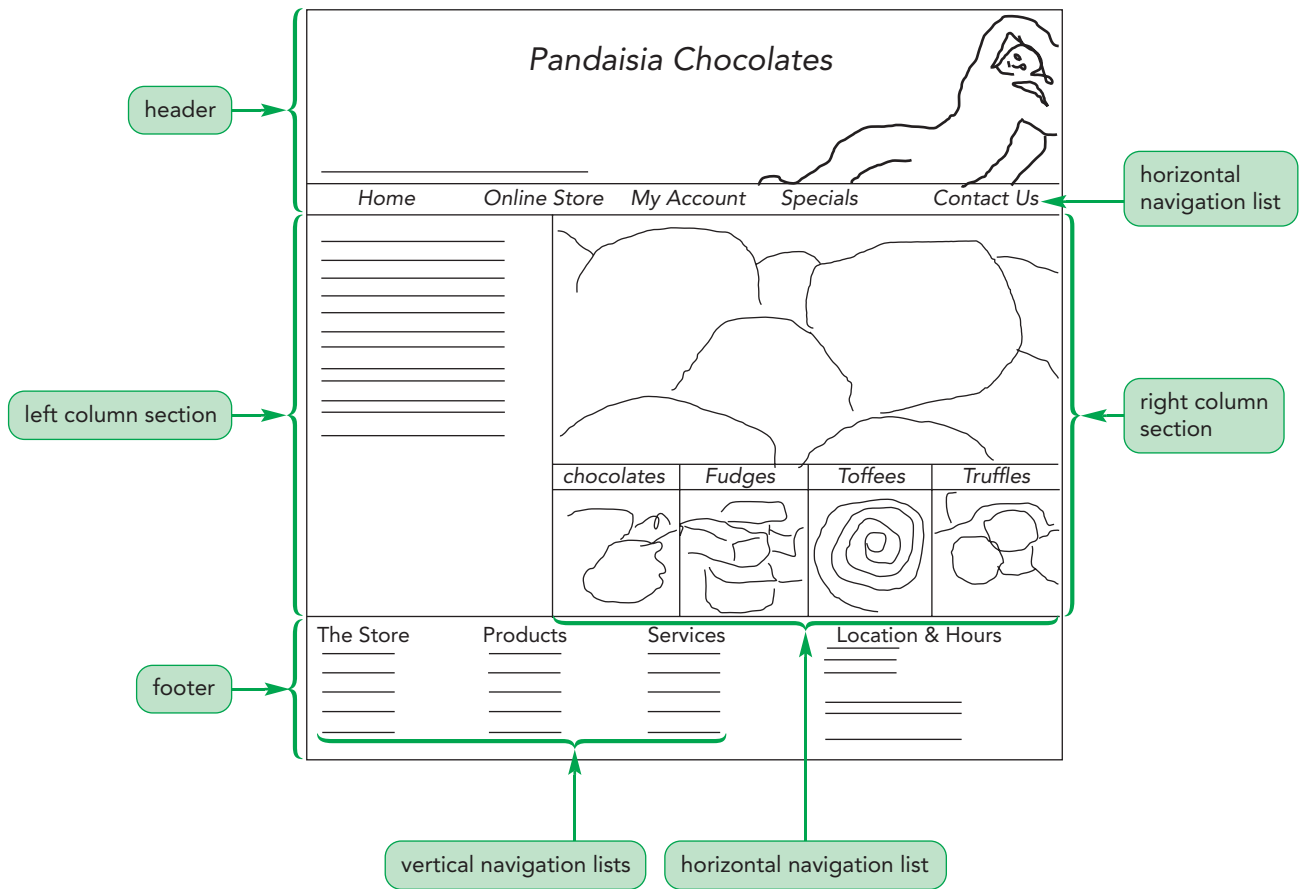
**To get started on the Pandaisia Chocolates home page:**

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.

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.

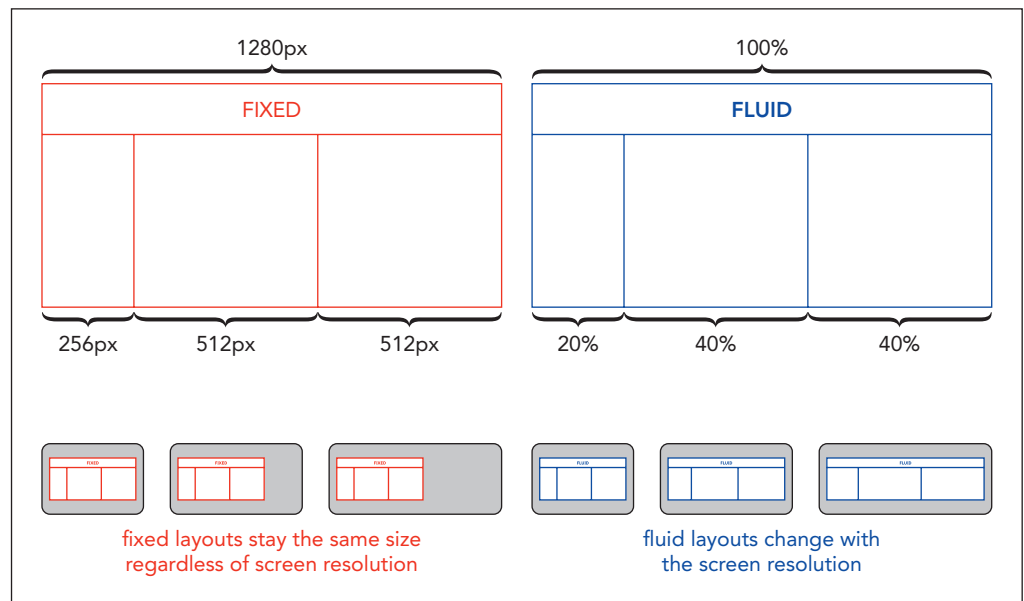
## Fixed, Fluid, and Elastic Layouts

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.



### 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

web page width is 95% of the browser window ranging from 640 pixels to 960 pixels

displays the logo image as a block element

sets the width of the logo to 100% of the page body

```

/* Body Styles */
body {
  max-width: 960px;
  min-width: 640px;
  width: 95%;
}

/* Body Header Styles */
body > header > img {
  display: block;
  width: 100%;
}

```

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

page body width is 95% of the browser window

browser window background

logo is 100% of the body width

body background

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

```
body {  
    margin-left: auto;  
    margin-right: auto;  
    max-width: 960px;  
    min-width: 640px;  
    width: 95%;  
}
```

setting the left and right margins to `auto` forces block elements to be horizontally centered within their parent

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.

### 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.

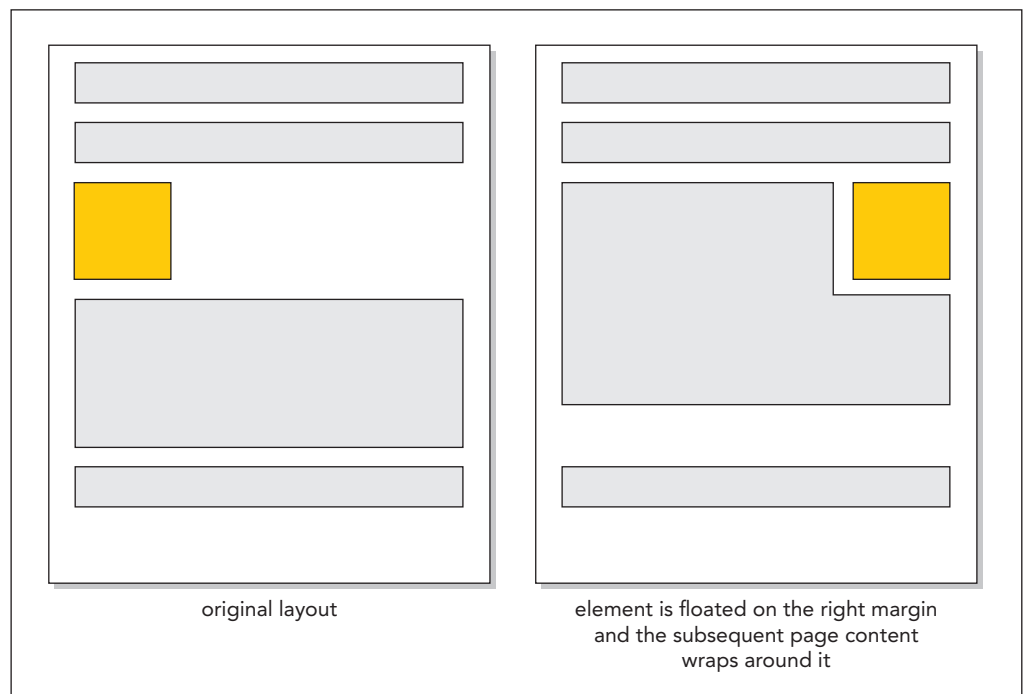
Next, you will lay out the links in the navigation list. Anne wants the links displayed horizontally rather than vertically. You can accomplish this using CSS floats.

## 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

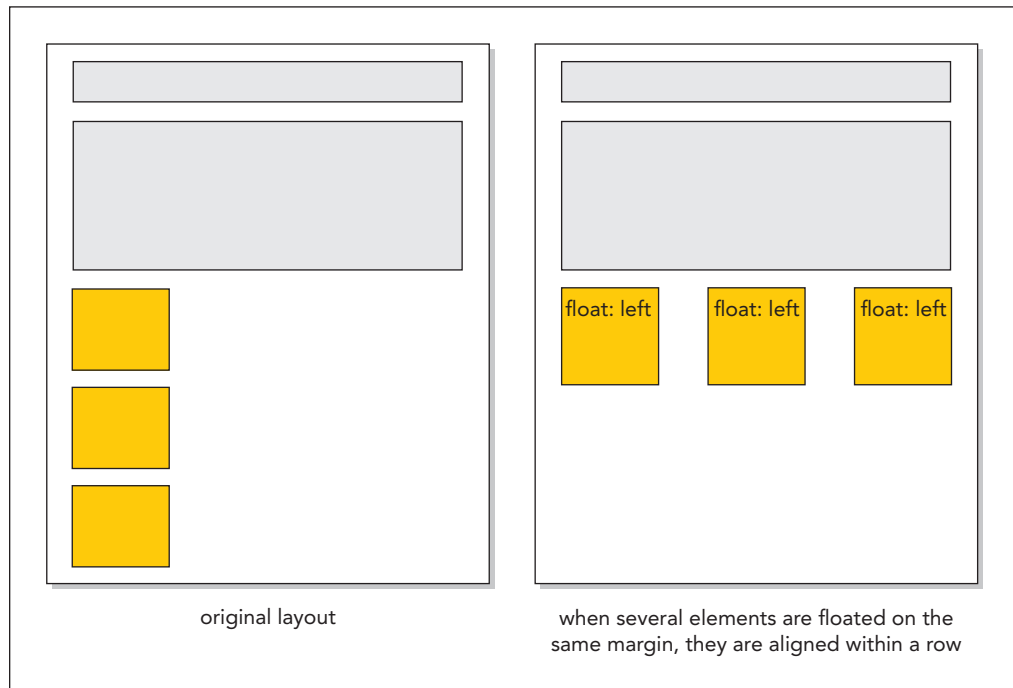


To float an element, apply the following `float` property

```
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 to 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 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%;
}
```

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



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

sets the width of the list item to 20% of the width of 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;
}
```

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:

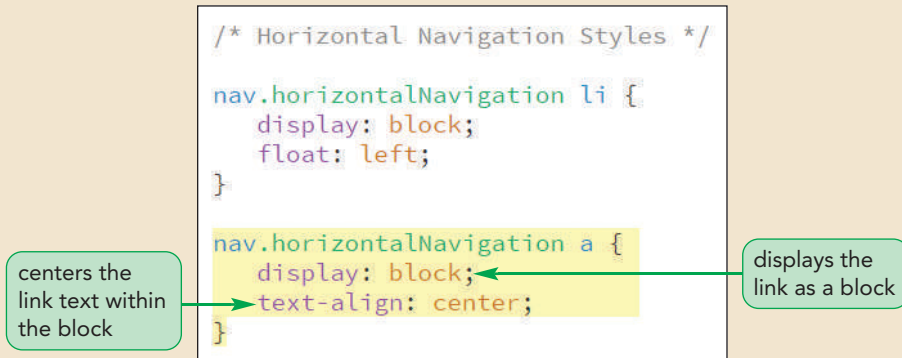
1. Return to the `pc_home.css` file in your editor.
2. Within the Horizontal Navigation Styles section, insert the following style rule to format the appearance of the hypertext links within the horizontal navigation lists:

```
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



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.

### 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:

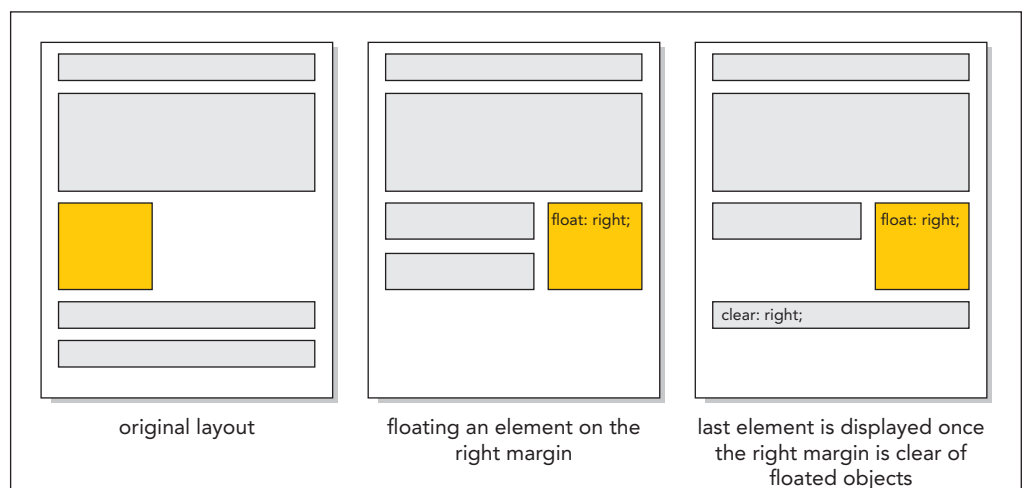
```
clear: position;
```

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



### 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:

- 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%;
}
```

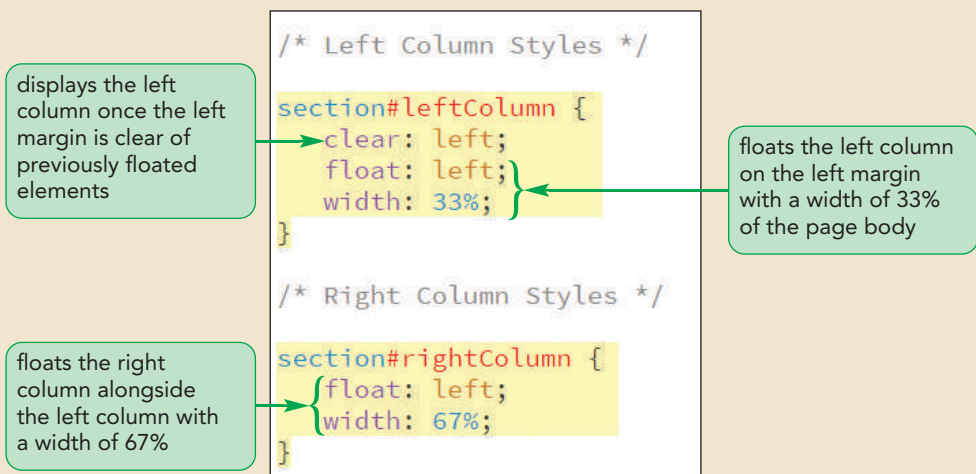
- Within the Right Column Styles section, insert:

```
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

```
/* Right Column Styles */

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

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

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

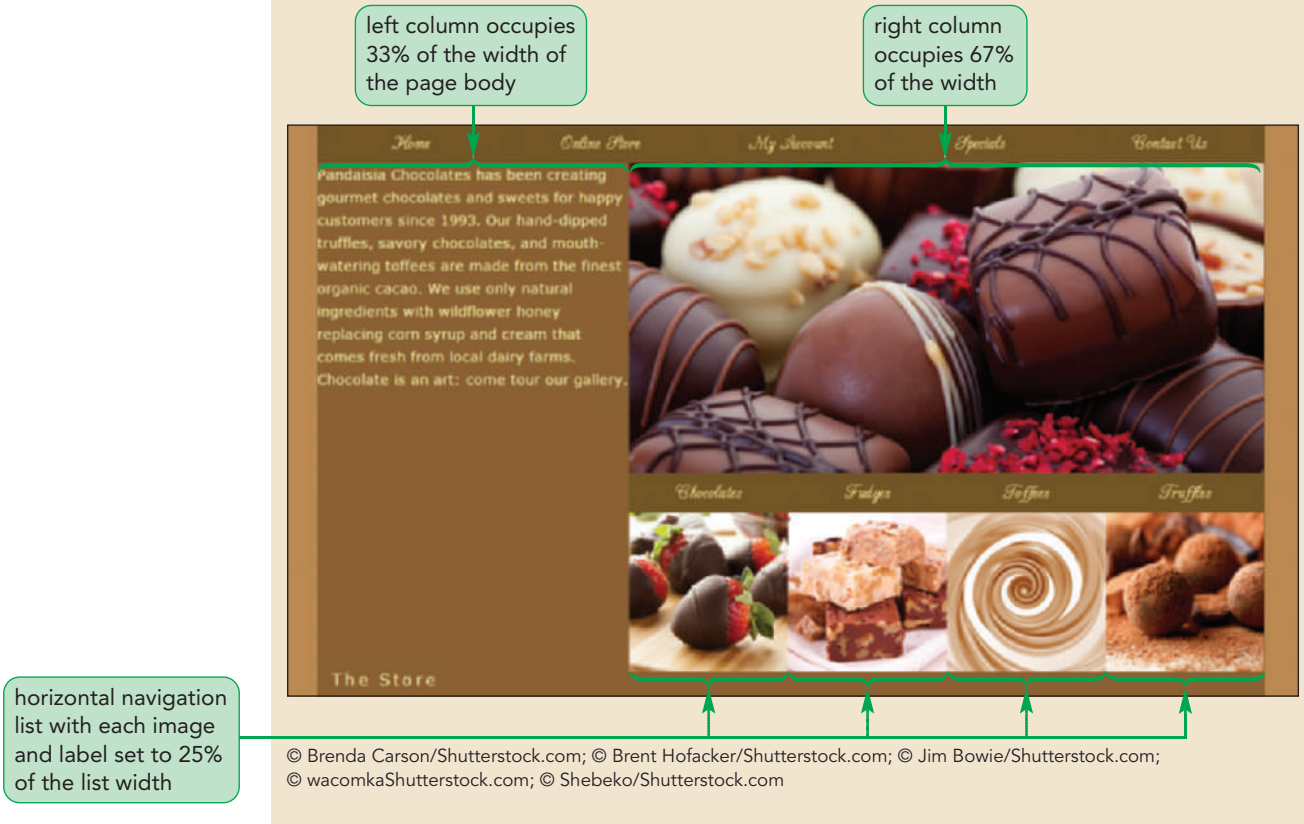
sets the width of each list item to 25% of the width of the navigation list

displays every image in the right column as a block with a width equal to the width of its parent element

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



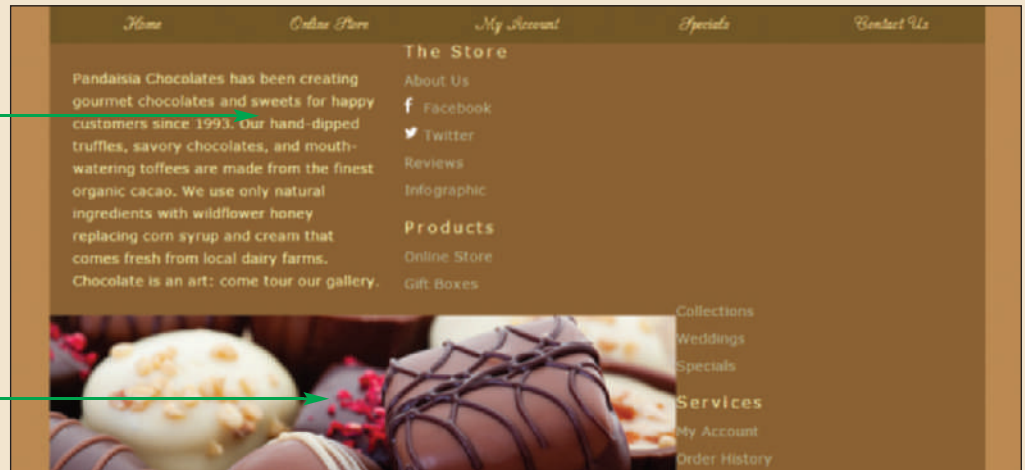
3. Save your changes to the style sheet and then reload the `pc_home.html` file in your browser. Figure 3–20 shows the result of your change.

Figure 3–20

## Page layout crashes with increased padding

increased padding increases the width of the left column, making it bigger than 33% of the page body width

the right column is forced to wrap to a new row, ruining the page layout



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

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

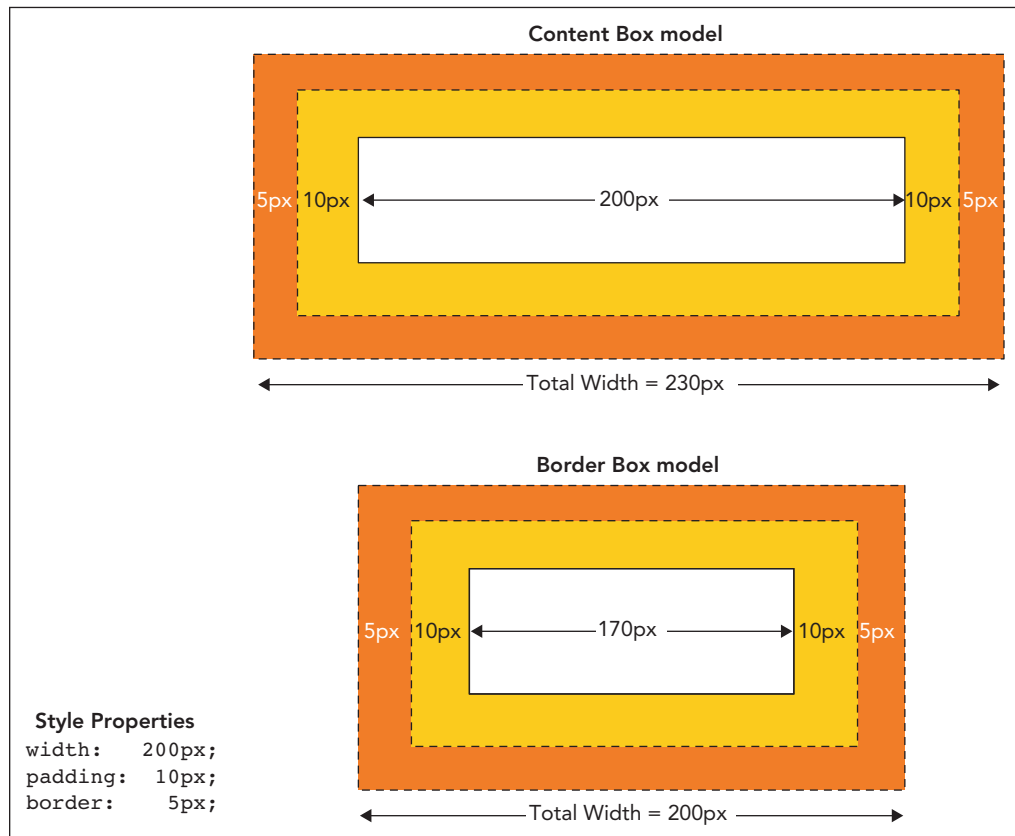
## Refining a Floated Layout

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**

```
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;
}
```

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

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

```
/* Footer Styles */
footer {
  clear: left;
}
```

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

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

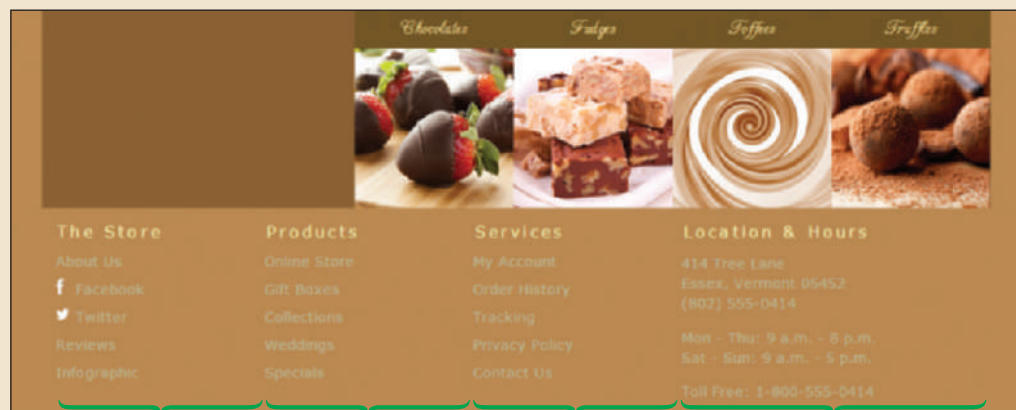
sets the width of the contactInfo section to 34% and floats it on the left

```
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

© Brent Hofacker/Shutterstock.com; © Jim Bowie/Shutterstock.com;  
© wacomkaShutterstock.com; © Shebeko/Shutterstock.com;  
Source: Facebook; Source: Twitter, Inc.

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

footer background set to a dark brown

```
footer {  
  background-color: rgb(71, 52, 29);  
  clear: left;  
}
```

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*.

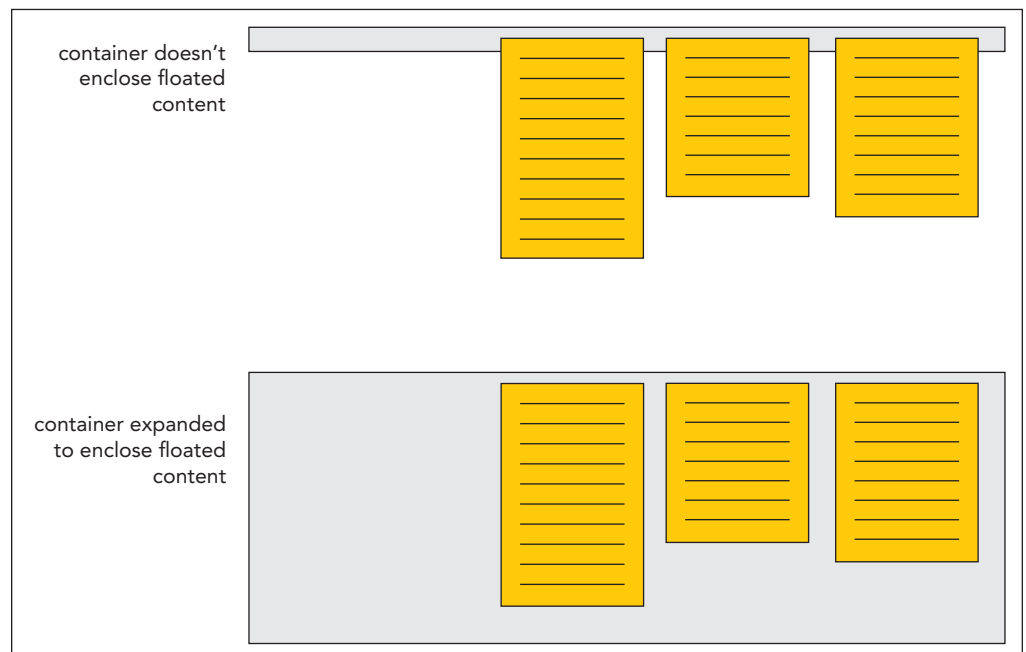
Why didn't the change to the background color take effect? To help you understand why, you'll look once again at the nature of floated elements.

## 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.

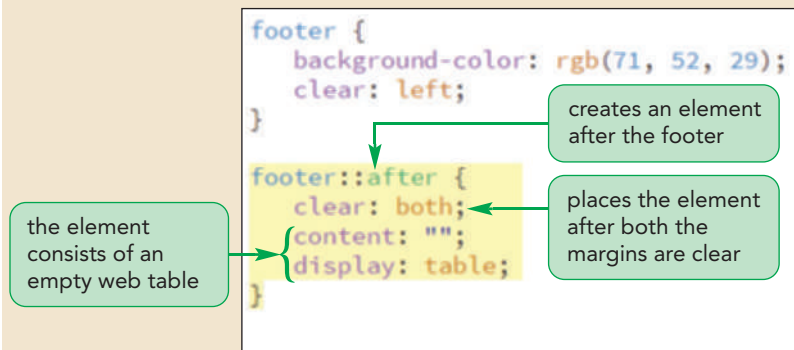
**To keep the footer from collapsing:**

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

© Brenda Carson/Shutterstock.com; © Brent Hofacker/Shutterstock.com; © Jim Bowie/Shutterstock.com; © wacomkaShutterstock.com; © Shebeko/Shutterstock.com; Source: Facebook; Source: Twitter, Inc.

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

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;
}
```

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

# CSS Grid Layouts

Page header covers the grid from column gridline 1 to -1 (the last gridline)

Columns laid out in a proportion of 2:1



aside element placed in the FAQ grid area

footer element placed in the footer grid area

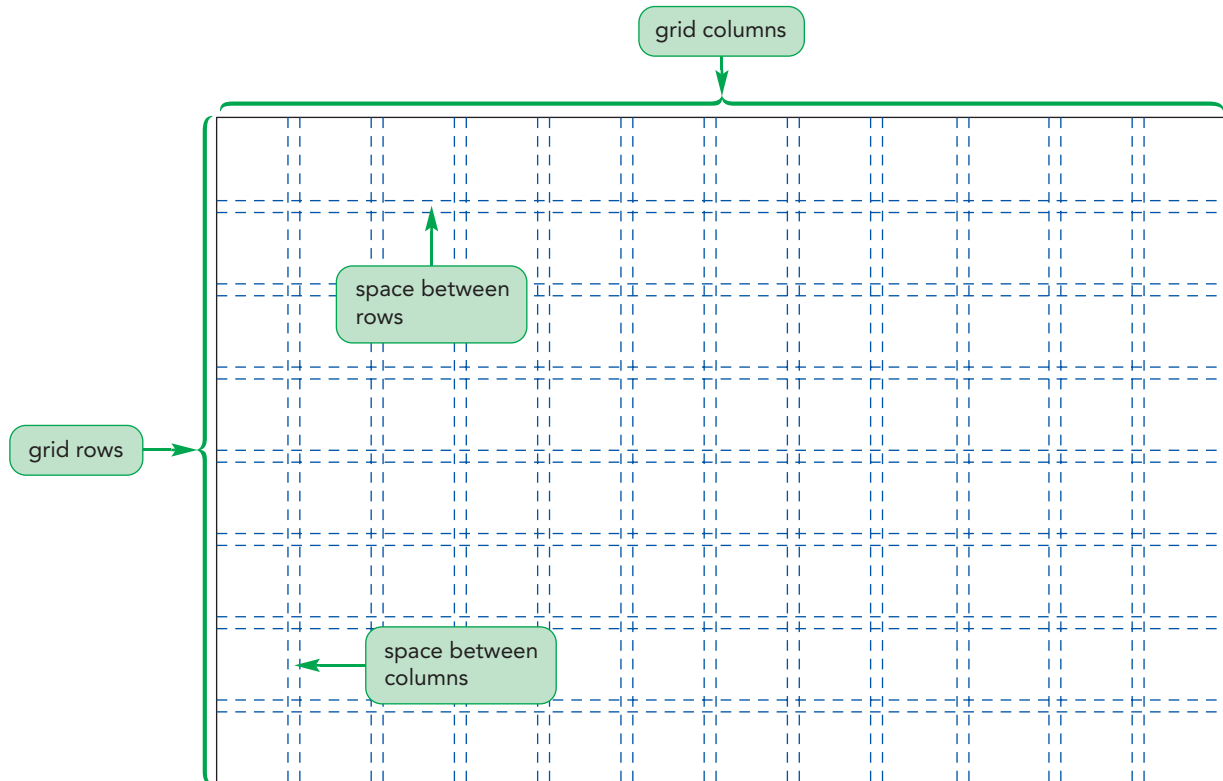
## Introducing Grid Layouts

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.

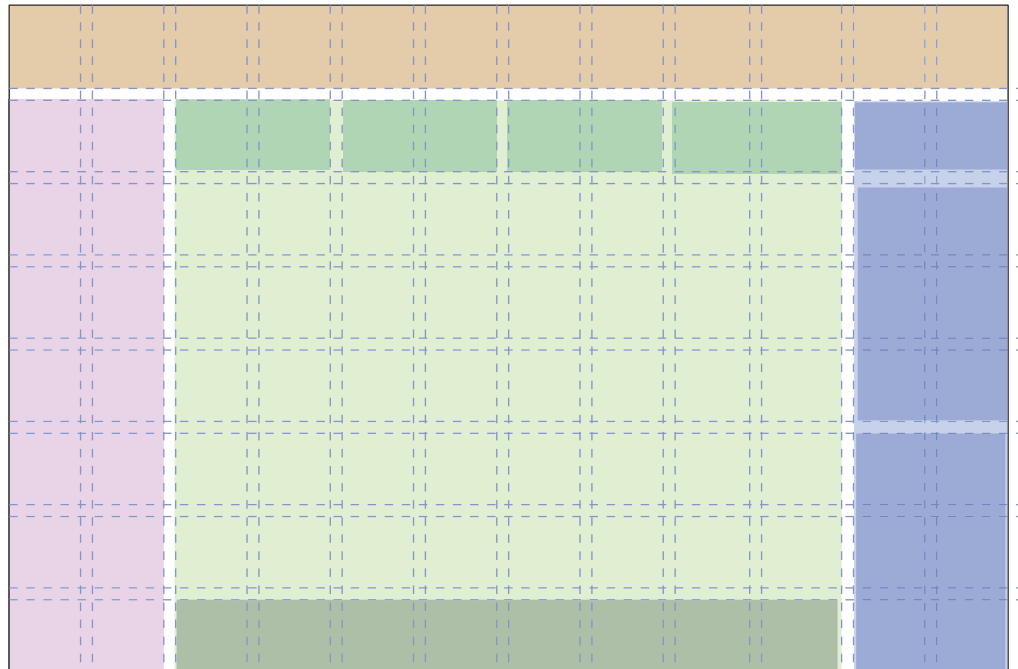
Figure 3–29 Page grid



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](http://getbootstrap.com))
- **Neat** ([neat.bourbon.io](http://neat.bourbon.io))
- **Unsemantic** ([unsemantic.com](http://unsemantic.com))
- **Profound Grid** ([www.profoundgrid.com](http://www.profoundgrid.com))
- **HTML 5 Boilerplate** ([html5boilerplate.com](http://html5boilerplate.com))
- **Skeleton** ([getskeleton.com](http://getskeleton.com))

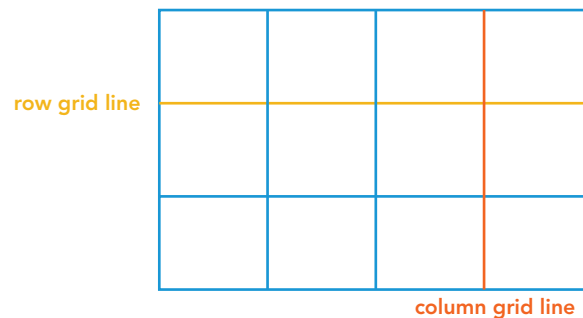
While a framework does a lot of the work in building the grid, you still need to understand how to interact with the underlying code, including the style sheets used to create a grid layout. In place of third-party frameworks, you can design your own grids using grid styles from CSS. Achieving Candidate 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.

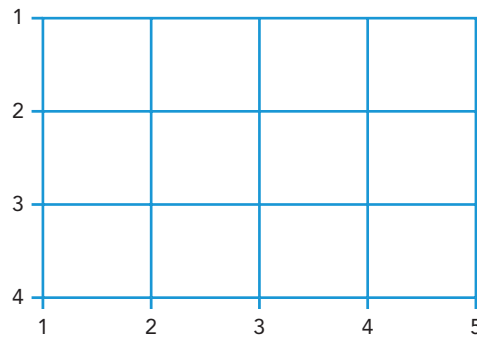
Figure 3–31

Row and column gridlines

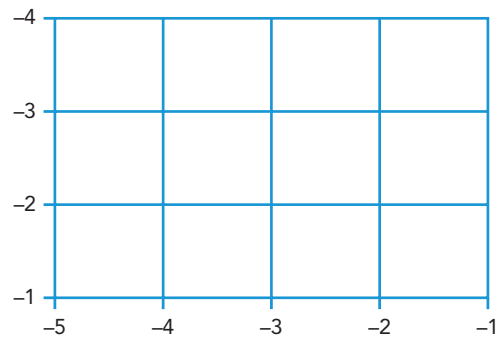


To reference positions within a grid, the CSS grid model numbers the gridlines in the horizontal and vertical directions, starting from the top-left corner of the grid with the row gridlines and then moving left to right with the column gridlines along the bottom. Both gridlines start with a value of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



**Figure 3–32** Numbering gridlines

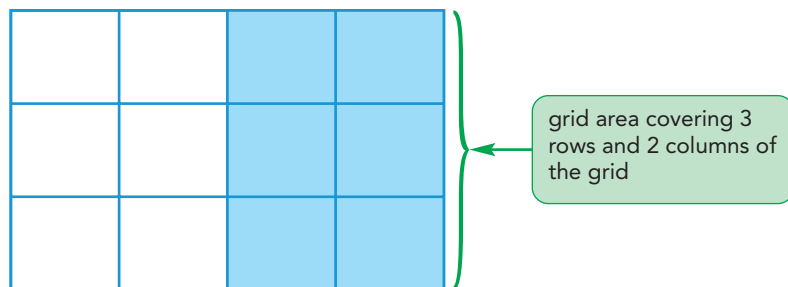
You can reference gridlines in the reverse order starting from the bottom-right corner with the first row and column gridlines in those directions are given a value of “-1” as shown in Figure 3–33.

**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.

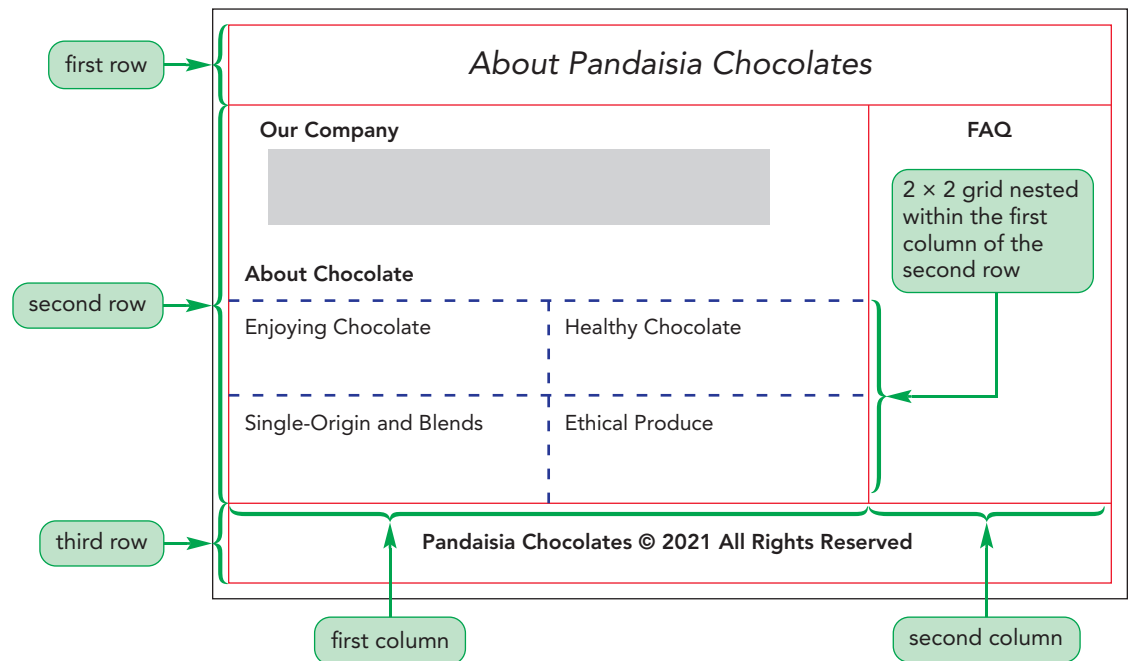
Figure 3–35

### 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.

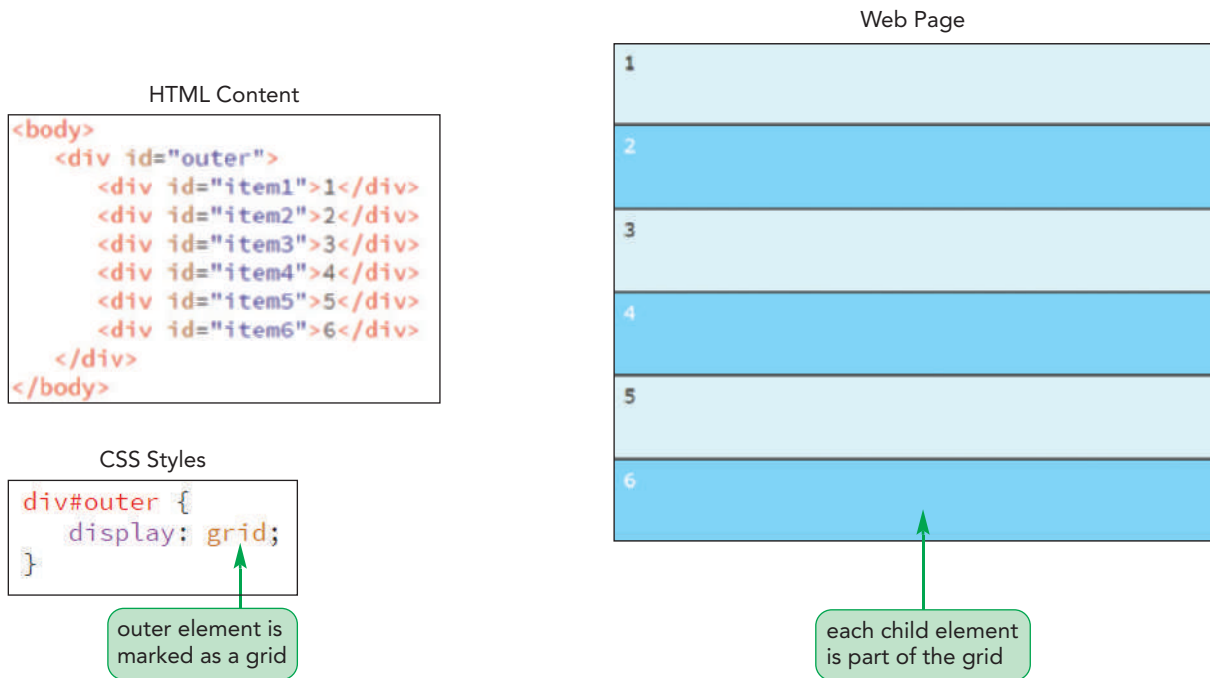
## Creating a CSS Grid

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 that 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;
}
```

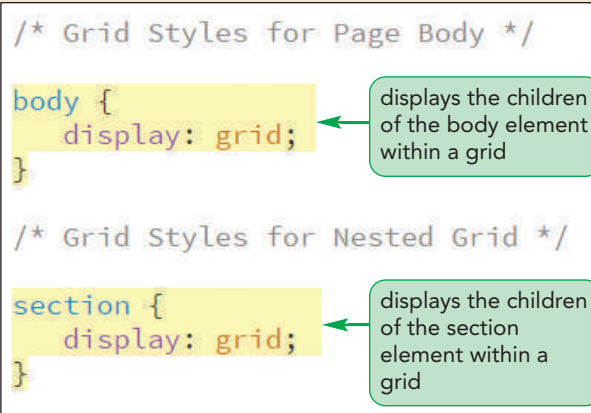
- 3. Within the Grid Styles for Nested Grid section, insert the following style rule to turn the `section` element into a grid:

```
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 ...;
```

### TIP

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

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;
```

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

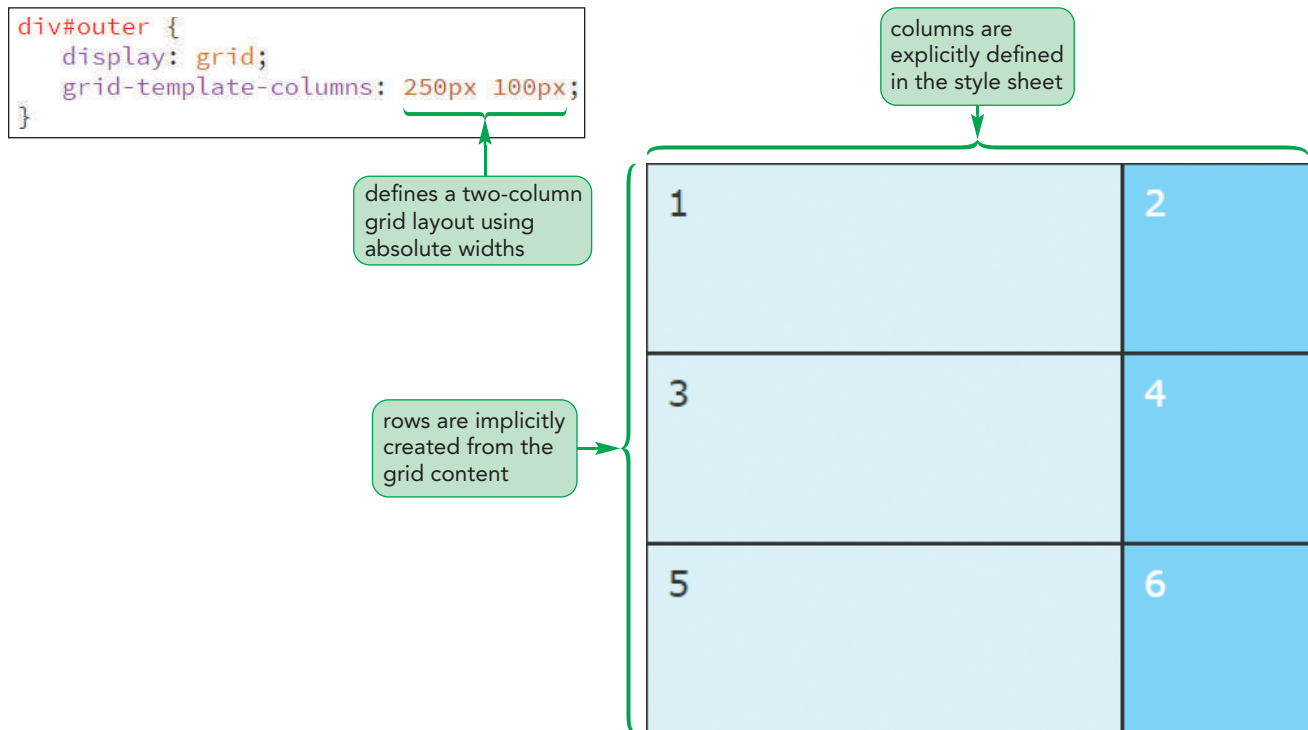


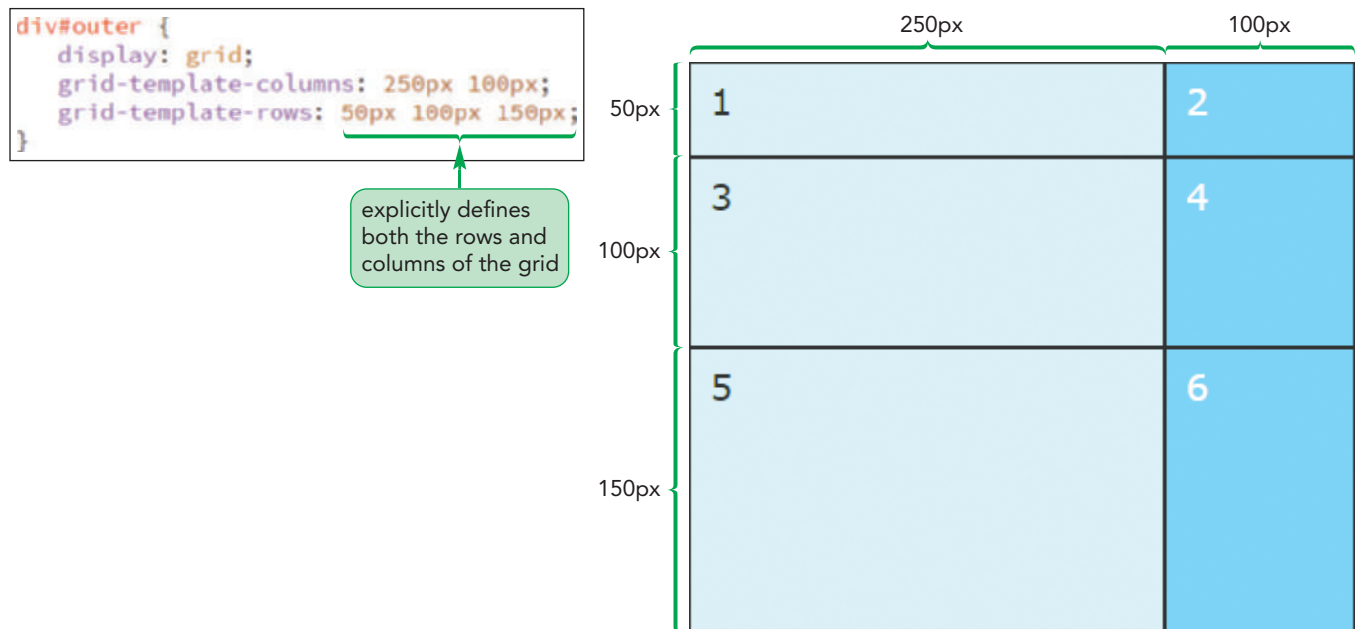
Figure 3–39 highlights an important contrast between explicit grids and implicit grids. An **explicit grid** completely defines the number and size of the grid rows and columns. An **implicit grid** contains rows and/or columns that are generated by the browser as it populates the grid with items from 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. Figure 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



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

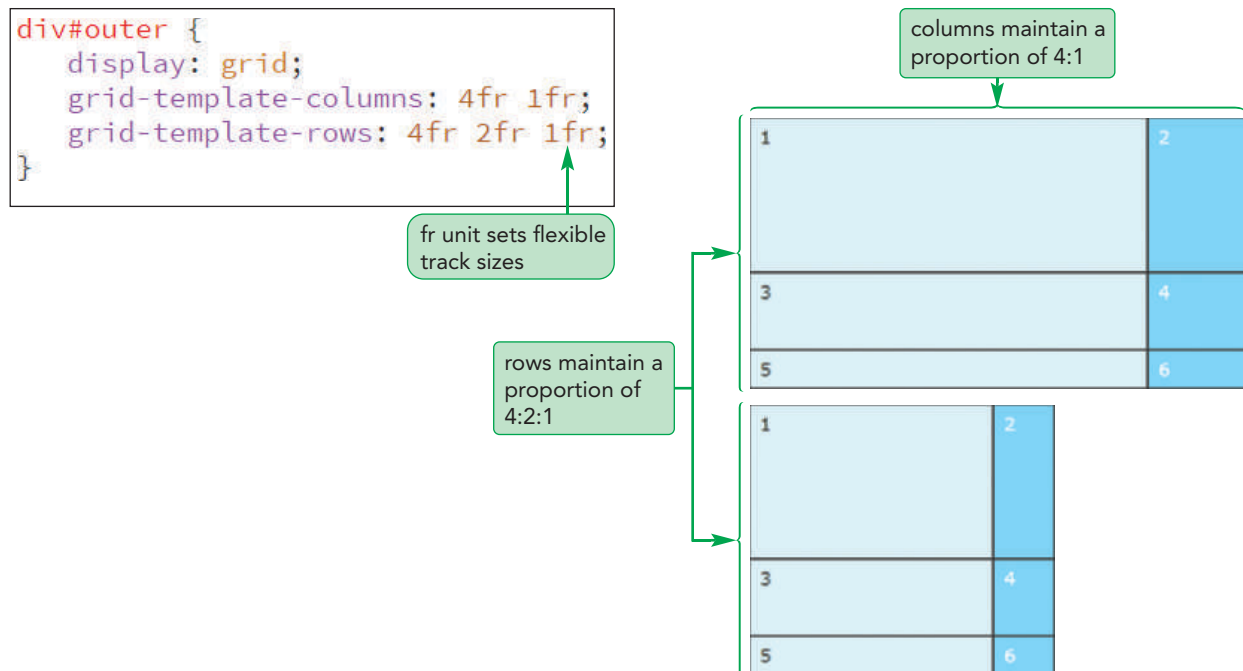
A grid layout will often need to adapt to devices of various screen widths and sizes. One way of accomplishing this is with flexible units. A **fr (fractional) unit**, indicated by the unit abbreviation `fr`, creates grid tracks that expand or contract in size to fill available space while retaining 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

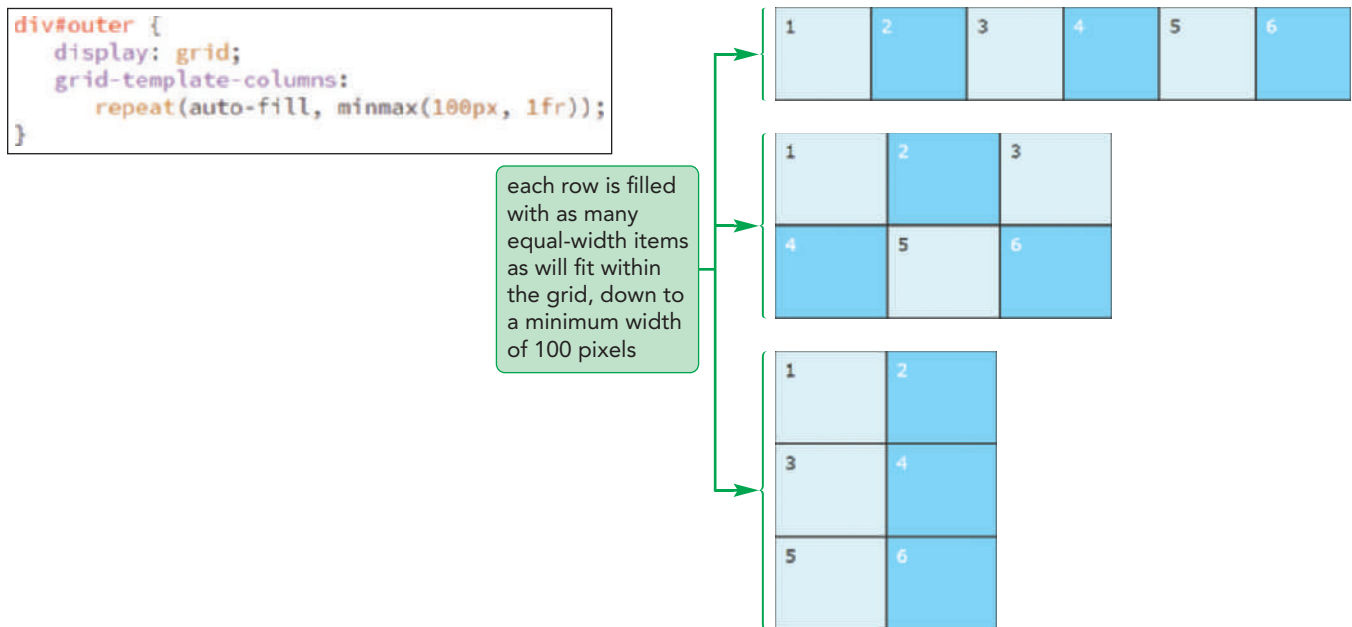
```
minmax(min, max)
```

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

first column is twice the width of the second

nested grid in a 2-column layout

*Pandaisia Chocolates*  
 414 Tree Lane - Essex, VT 05452  
[Home](#) [Online Store](#) [My Account](#) [Specials](#) [Contact Us](#)

*About Pandaisia Chocolates*

**Our Company**  
 We are a company located in Essex, Vermont, dedicated to making delicious chocolate and other treats. For our founder, chocolatier Anne Ambrose, this means using only the finest organic ingredients, incorporating a harmonious blend of rich flavors and smooth textures.  
 Anne learned her trade as part of a three-year apprenticeship program in Switzerland. Her introduction into the world of confectioneries was a springboard to working with leaders in the field. Early in 1993 she brought that expertise back to Vermont and Pandaisia Chocolates was born.

**About Chocolate**  
 We believe that the best chocolate is fresh chocolate. Preservatives change the flavor and texture of chocolate. For the best results, our chocolates should be consumed within a few days of purchase. Store them in a cool, dark place at a temperature of 60° to 70° such as a refrigerator or wine cellar.

**Enjoying Chocolates**  
 We believe in single-origin chocolate made from one variety of cacao harvested from a single region. Single-origin chocolates take on the unique flavors of their region (in the same way that wines adopt regional distinctions.) Other chocolatiers use blends that combine flavors from several regions. Let us know what you prefer.

**FAQ**  
 Do you do weddings?  
 Yes! That's our favorite thing to do. We sell bulk chocolates in a wide variety of box designs perfect for weddings or other special occasions.  
 How long do your chocolates last?  
 We recommend that you store our chocolates in a dark, cool place and consume them within two weeks of purchase.  
 What varieties are you selling?  
 We're constantly updating our product list to match seasonal expectations. Typically, we have between 12 and 18 varieties at any one time.  
 Can I customize my own box?  
 Of course! We have special gift boxes but if you want to create a box of your favorites, we're glad to oblige.

**Healthy Chocolate**  
 Chocolate has a bad reputation because of the poor quality of mass-produced bars loaded with lots of milk, sugar, and butter — which are tasty but not healthy. We start with organic ingredients, keep the processed sugars to a minimum and produce dark chocolate that is 73% cacao. At that level, you can reap the benefits of a chocolate diet!

**Single-Origin and Blends**  
 We believe in single-origin chocolate made from one variety of cacao harvested from a single region. Single-origin chocolates take on the unique flavors of their region (in the same way that wines adopt regional distinctions.) Other chocolatiers use blends that combine flavors from several regions. Let us know what you prefer.

Twin Design/  
Shutterstock.com

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;  
}
```

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.

### TIP

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

### 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.

**TIP**

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

**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.

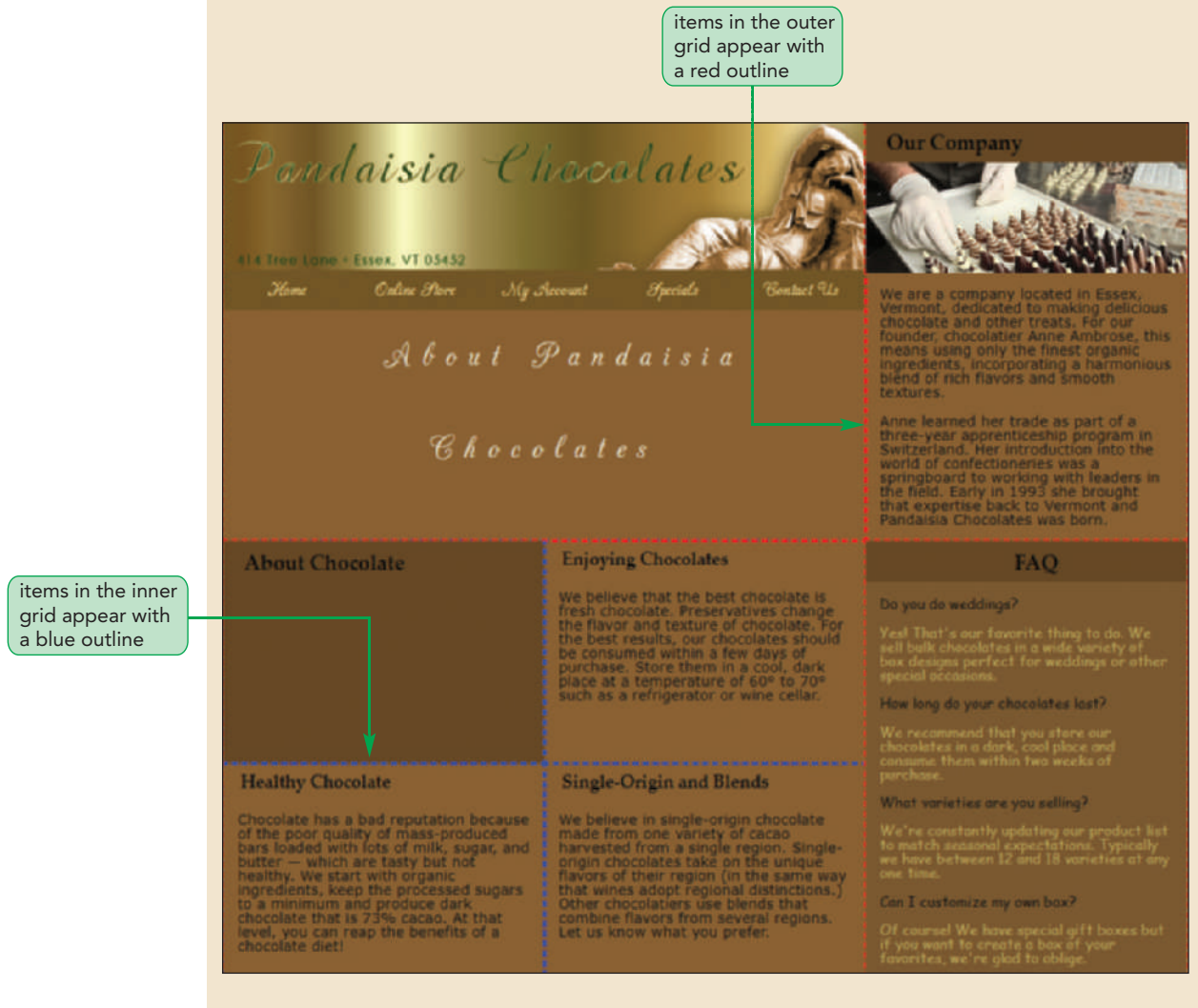
**Figure 3–45****Adding outlines to grid items**

```
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



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 have a single item occupy multiple rows and columns.

### 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:

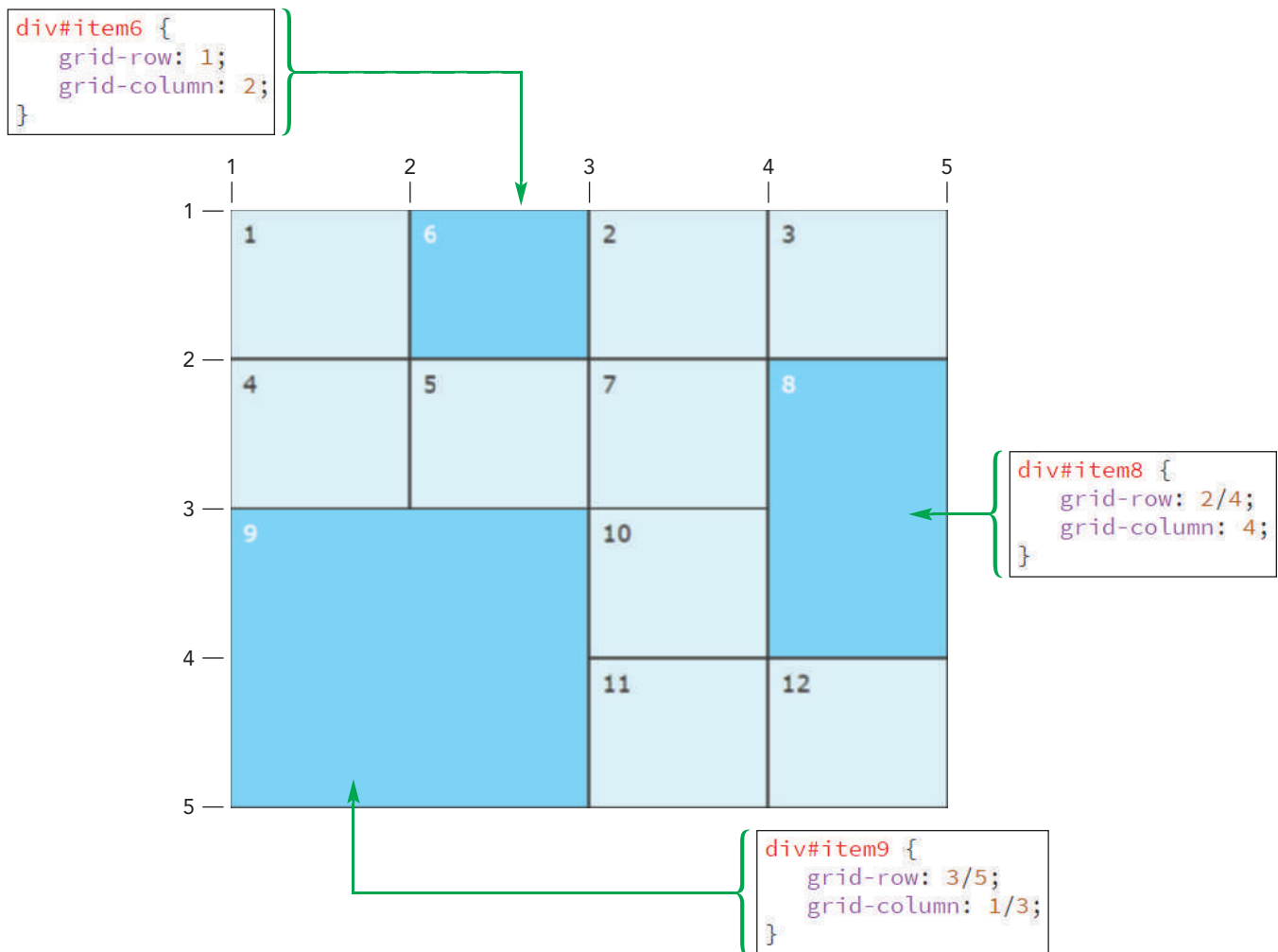
```
article {  
  grid-row: 1;  
  grid-column: 2;  
}
```

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

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

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

header placed in the first grid row

body header extends from the first gridline to the last

number of the first gridline

number of the last gridline

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

body header occupies the first row

Pandaisia Chocolates

414 Tree Lane • Essex, VT 05452

Home Online Store My Account Specials Contact Us

About Pandaisia Chocolates

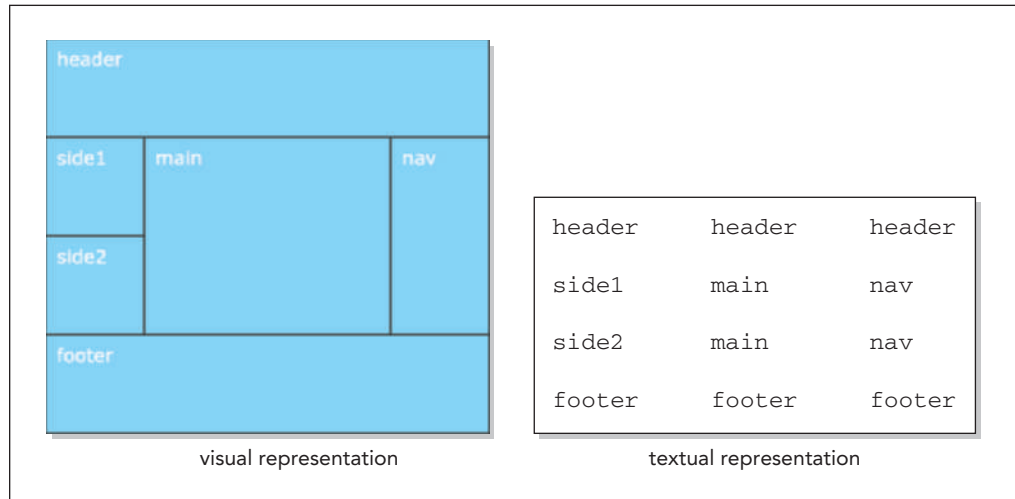
Our Company About Chocolate Enjoying Chocolates

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:

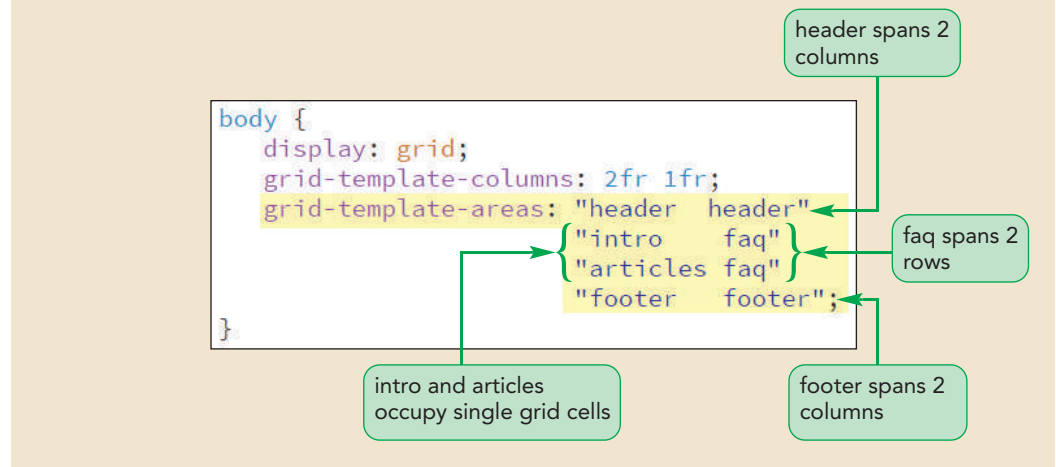
```
grid-template-areas: "header header"
                    "intro faq"
                    "articles faq"
                    "footer footer";
```

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

```

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;}

```

place the article element in the intro area

place the aside element in the faq area

place the section element in the articles area

place the footer element in the footer area

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

```

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

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

```

extends the h1 heading from the first through second row gridlines and the first through third column gridlines

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

*Pandaisia Chocolates*

414 Tree Lane • Essex, VT 05452

Home Online Store My Account Specials Contact Us

*About Pandaisia Chocolates*

### Our Company



We are a company located in Essex, Vermont, dedicated to making delicious chocolate and other treats. For our founder, chocolatier Anne Ambrose, this means using only the finest organic ingredients, incorporating a harmonious blend of rich flavors and smooth textures.

Anne learned her trade as part of a three-year apprenticeship program in Switzerland. Her introduction into the world of confectioneries was a springboard to working with leaders in the field. Early in 1993 she brought that expertise back to Vermont and Pandaisia Chocolates was born.

### About Chocolate

#### Enjoying Chocolates

We believe that the best chocolate is fresh chocolate. Preservatives change the flavor and texture of chocolate. For the best results, our chocolates should be consumed within a few days of purchase. Store them in a cool, dark place at a temperature of 60° to 70° such as a refrigerator or wine cellar.

#### Single-Origin and Blends

We believe in single-origin chocolate made from one variety of cacao harvested from a single region. Single-origin chocolates take on the unique flavors of their region (in the same way that wines adopt regional distinctions.) Other chocolatiers use blends that combine flavors from several regions. Let us know what you prefer.

#### Healthy Chocolate

Chocolate has a bad reputation because of the poor quality of mass-produced bars loaded with lots of milk, sugar, and butter — which are tasty but not healthy. We start with organic ingredients, keep the processed sugars to a minimum and produce dark chocolate that is 73% cacao. At that level, you can reap the benefits of a chocolate diet!

#### Ethical Produce

We work directly with farmers in Peru, Ecuador, and Honduras, where we learn first-hand about the economic struggles they face. We pay above-market premiums to maintain our relationships with farmers and support Fair Trade agreements. We are always striving to foster a sustainable market for fine chocolate and to support the hard-working men and women who are our producers.

### FAQ

Do you do weddings?

Yes! That's our favorite thing to do. We sell bulk chocolates in a wide variety of box designs perfect for weddings or other special occasions.

How long do your chocolates last?

We recommend that you store our chocolates in a dark, cool place and consume them within two weeks of purchase.

What varieties are you selling?

We're constantly updating our product list to match seasonal expectations. Typically we have between 12 and 18 varieties at any one time.

Can I customize my own box?

Of course! We have special gift boxes but if you want to create a box of your favorites, we're glad to oblige.

Can I request a special variety?

We're happy to consider requests, but remember that some varieties are seasonal and cannot always be made to order.

What about peanut allergies?

We can produce box sets without nuts, but our factory is not a nut-free environment, so some fragments might get into your box however much we try to avoid it.

Where is Pandaisia?

Glad you asked. Pandaisia is not a place; it's the name of the Greek goddess of the banquet and what's a banquet without chocolate?

Pandaisia Chocolates © 2021 All Rights Reserved

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.

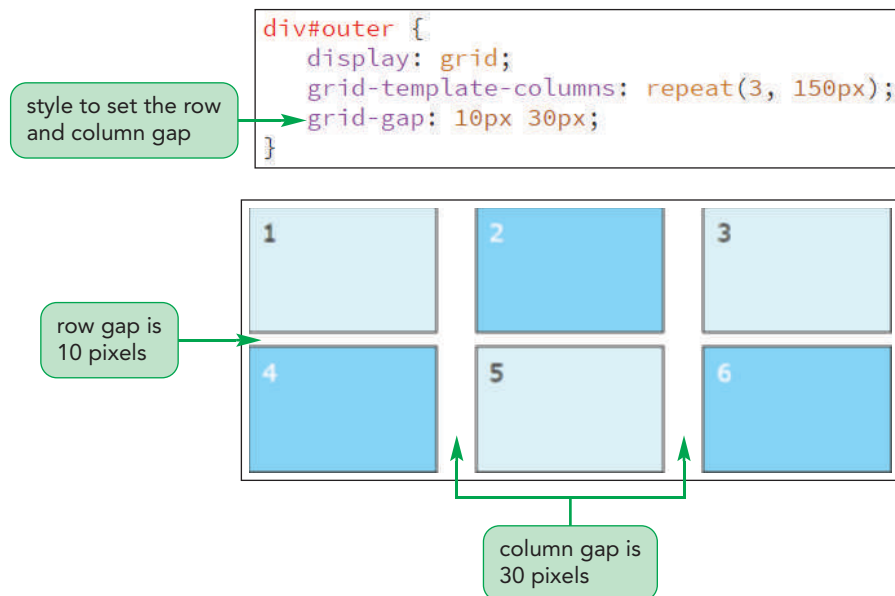
## Defining the Grid Gap

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



15-pixel gap between columns

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

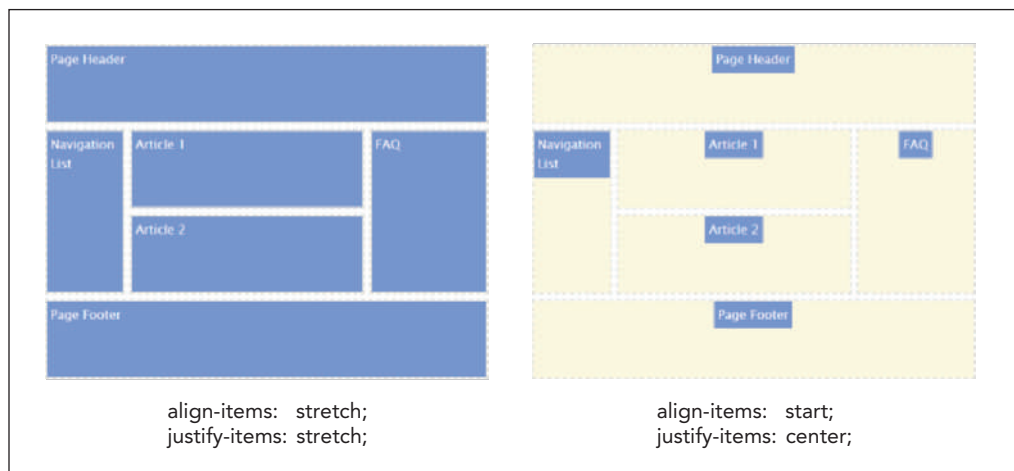
### TRY IT

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



## 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

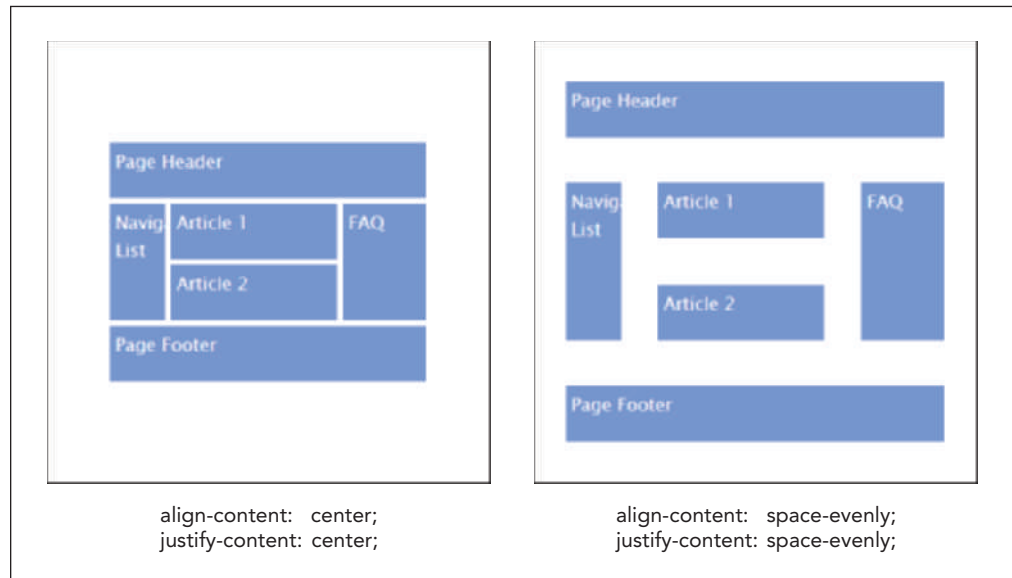
### TRY IT

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.

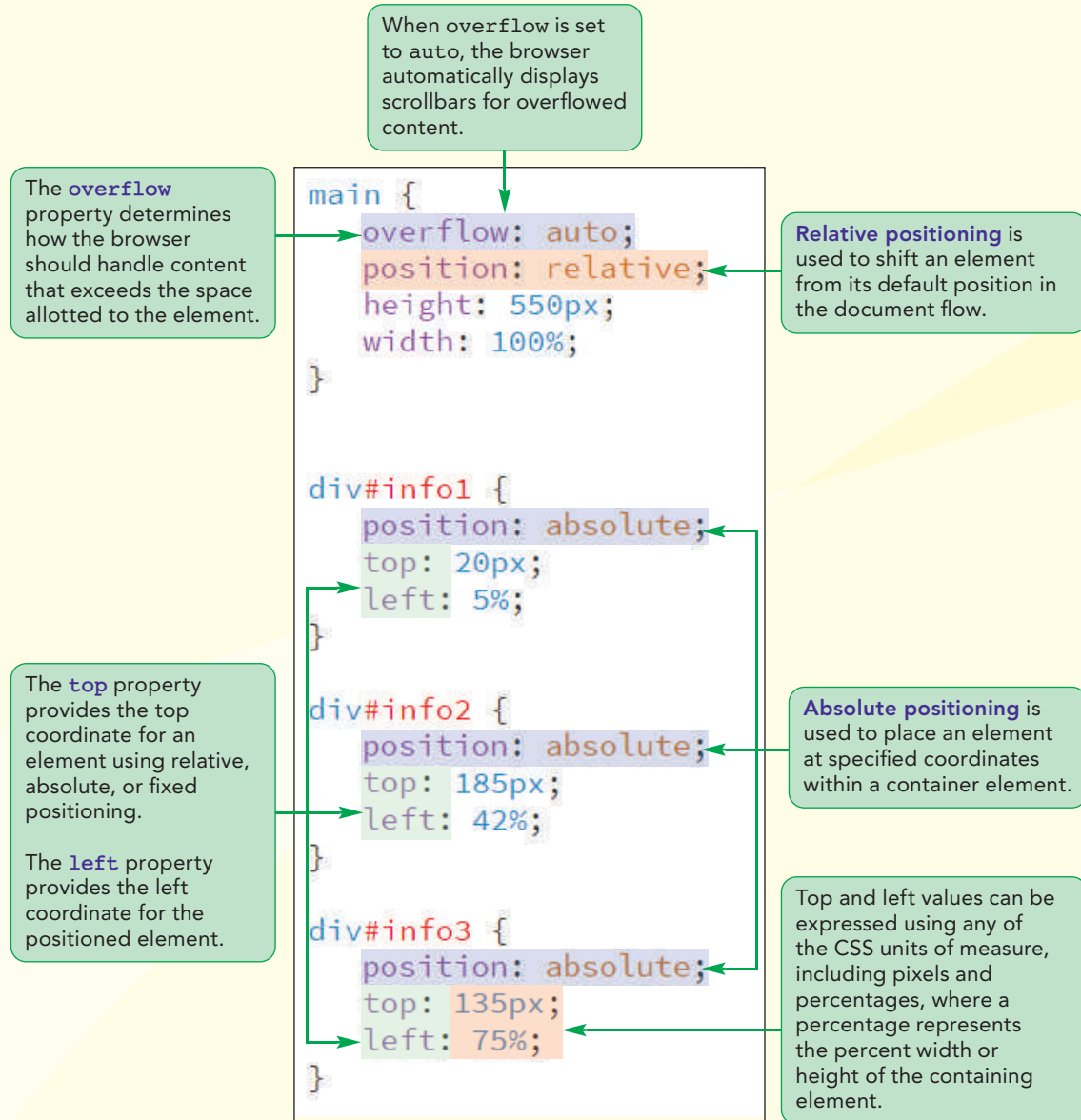
In the next session, you'll explore CSS positioning styles that allow you to place page elements anywhere within the rendered page.

## REVIEW

### Session 3.2 Quick Check

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

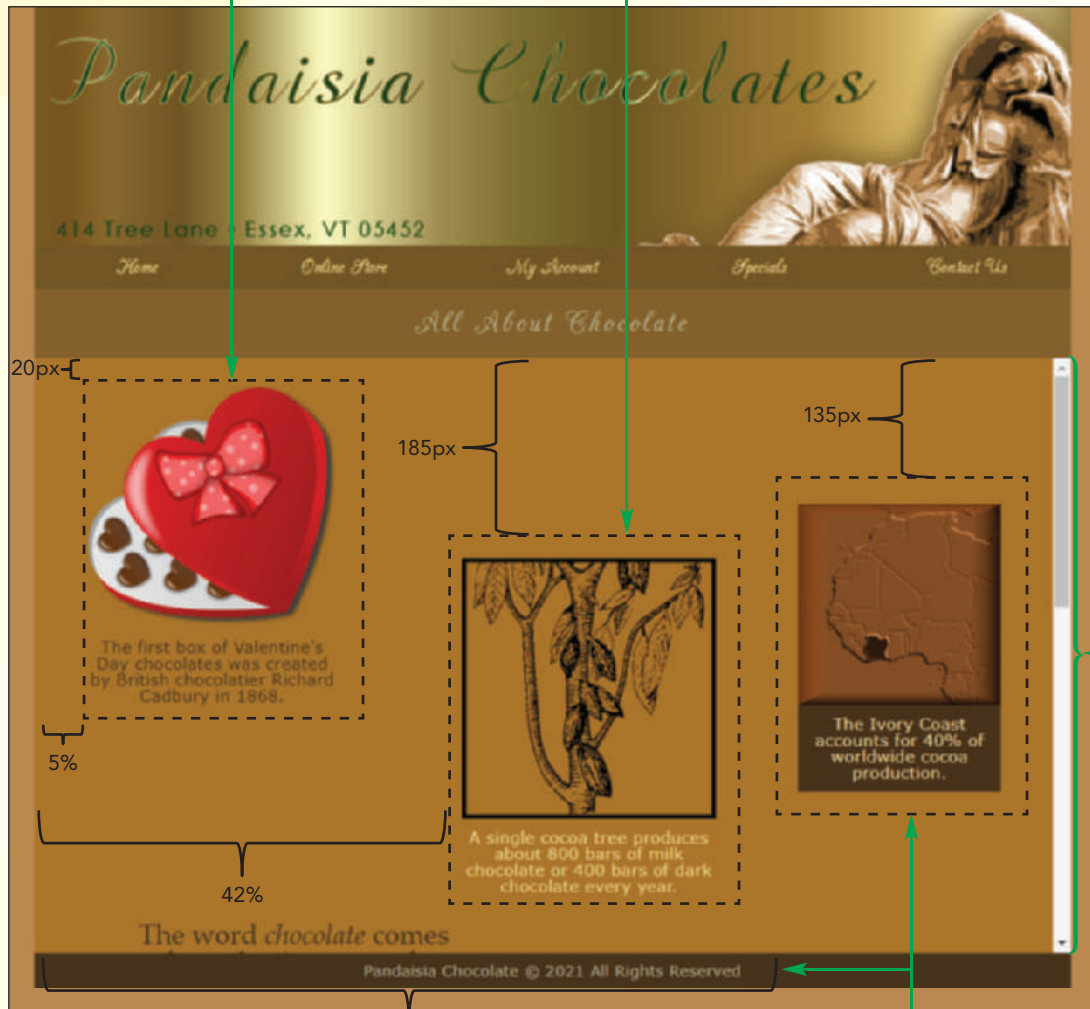
# Session 3.3 Visual Overview:



# Layout with Positioning Styles

info1 is placed 20 pixels from the top of the main element and 5% from the left edge.

info2 is placed 185 pixels from the top and 42% from the left edge of the main element.



Vertical scrollbar is automatically added to view the overflowed content.

info3 is placed 135 pixels from the top and 75% from the left edge.

## 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.

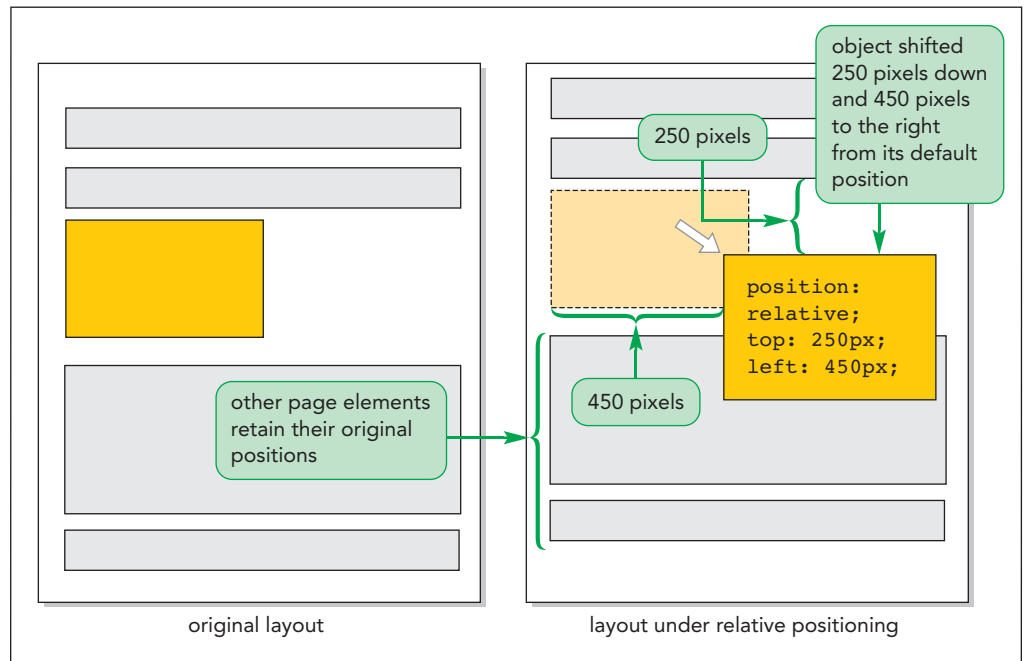
### Relative Positioning

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.

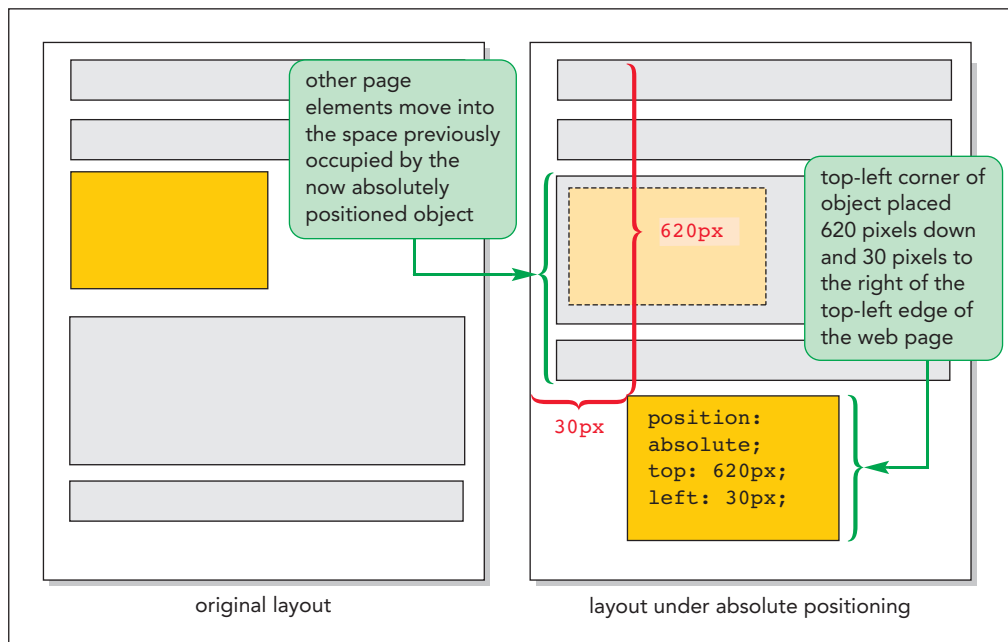
For example, the following style rule places the `header` element 620 pixels from the top edge of its container and 30 pixels from the left edge (see Figure 3–61).

### 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.

```
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;}

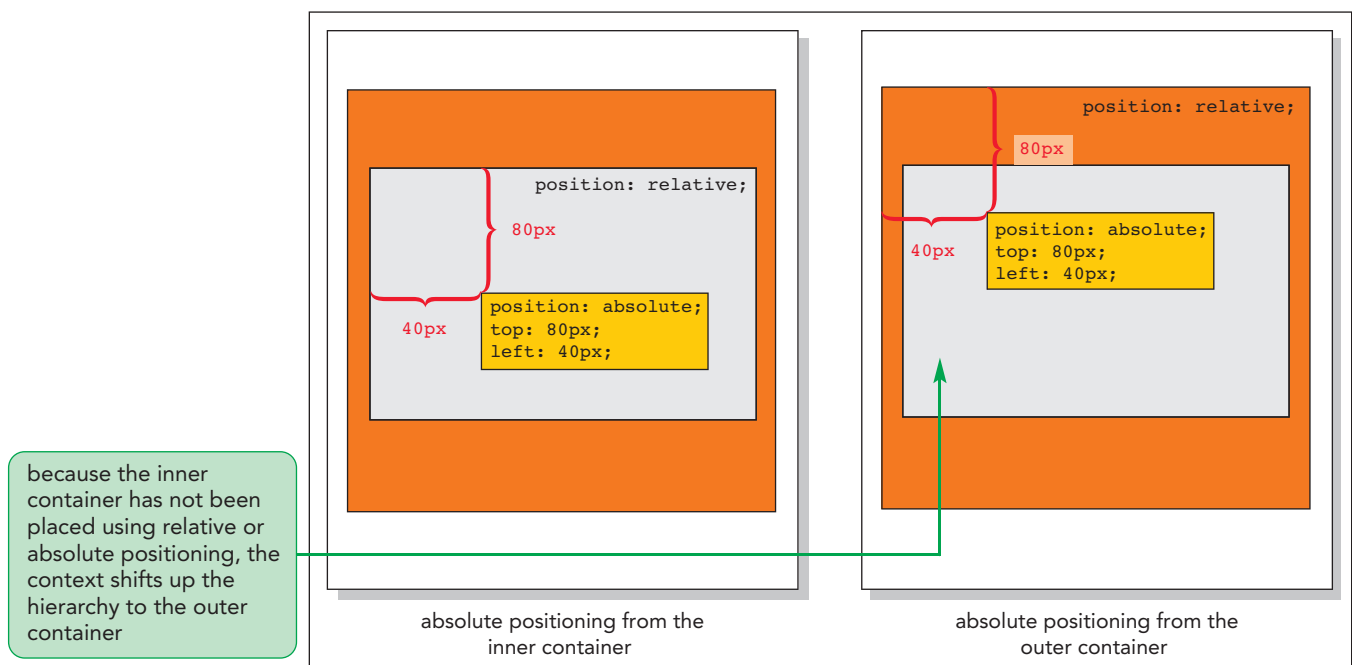
body > section {position: static;}

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**

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

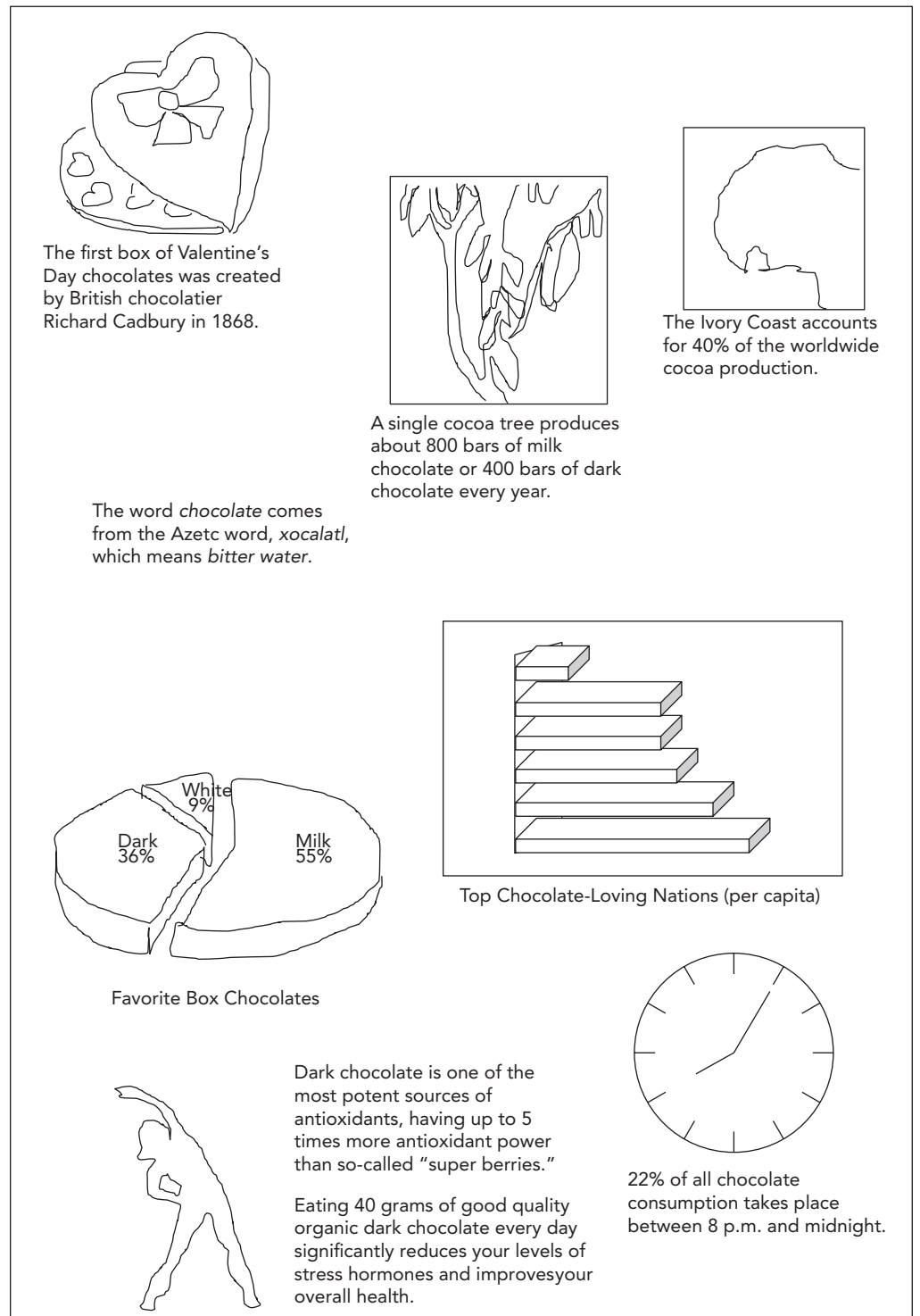
```
position: fixed;
```

## 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`.
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.

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

Figure 3–64

## Setting the display styles of the main element

hides the div elements of the infobox class

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

sets the height of the main element to 1400 pixels and makes it the width of the page body

```
/* Main Styles */

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

places the main element using relative positioning

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.

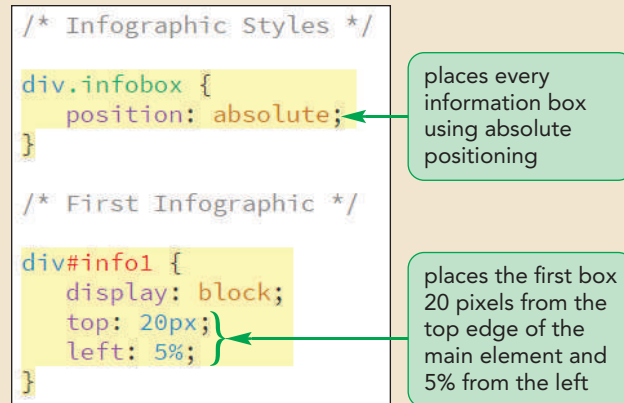
```
div#info1 {
  display: block;
  top: 20px;
  left: 5%;
}
```

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

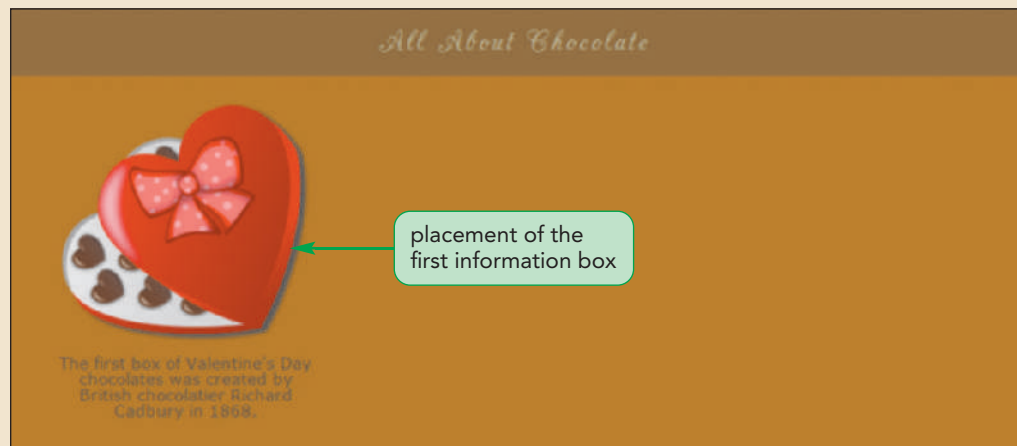
## Placing the first information box



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

places the second box 185 pixels from the top and 42% from the left

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

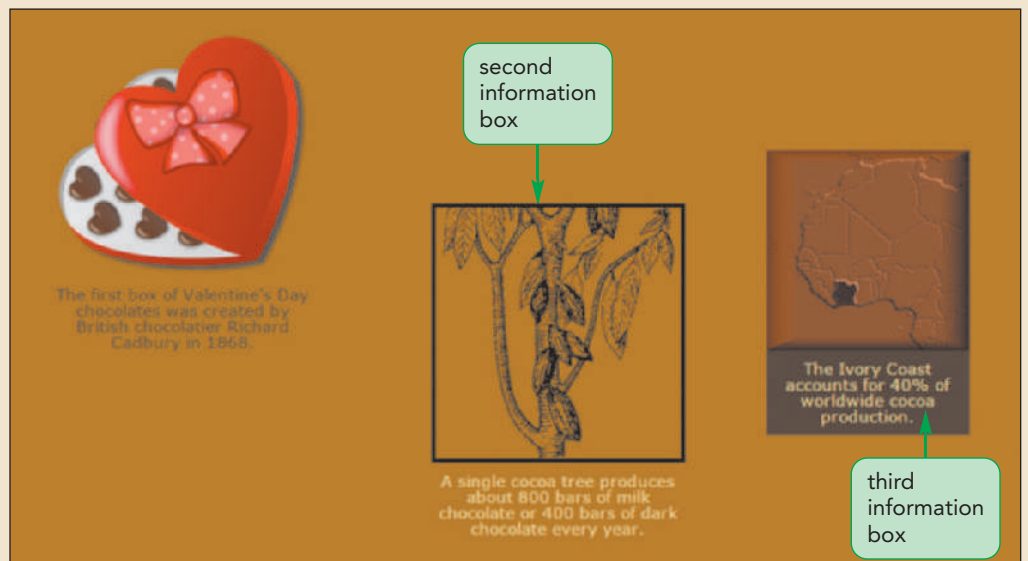
places the third box 135 pixels from the top and 75% from the left

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

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.

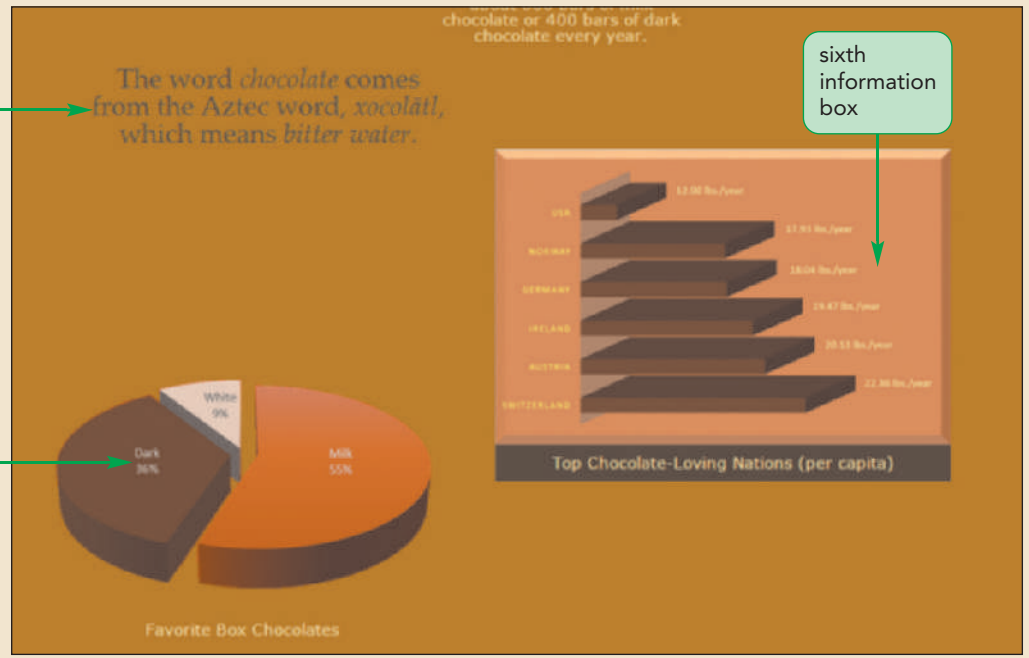
Figure 3–70

Placement of the next three boxes

fourth information box

fifth information box

sixth information box



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

places the seventh box 1000 pixels from the top and 68% from the left

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

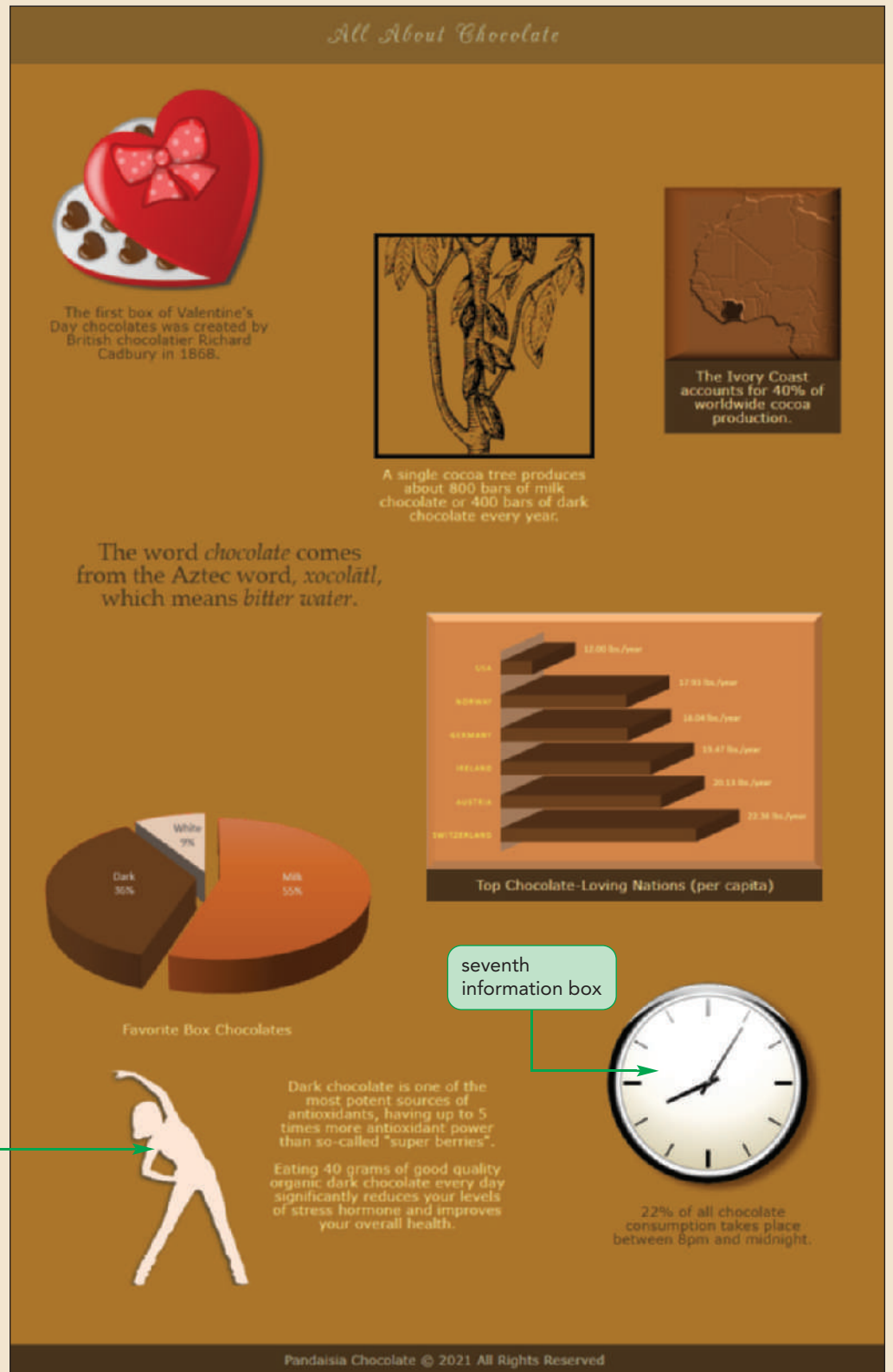
places the eighth box 1100 pixels from the top and 12% from the left

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

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



eighth information box

seventh information box

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

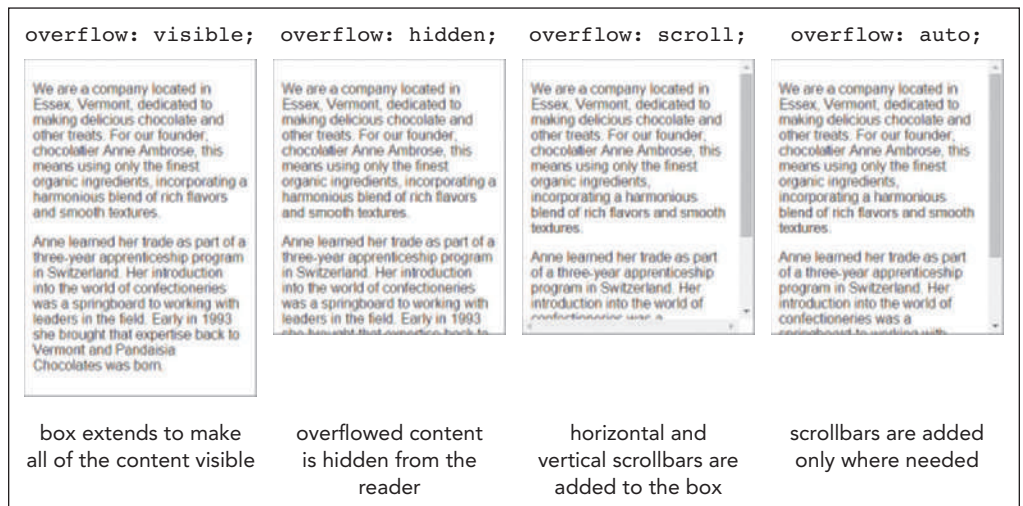
```
overflow: type;
```

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`.

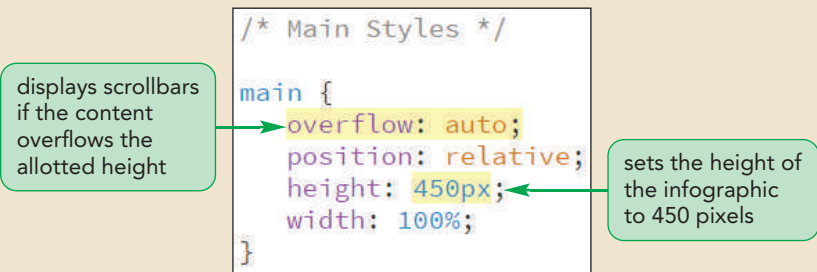
You decide to limit the height of the infographic to 450 pixels and to set the `overflow` property to `auto` so that browsers displays scroll bars as needed for the excess content.

**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



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



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

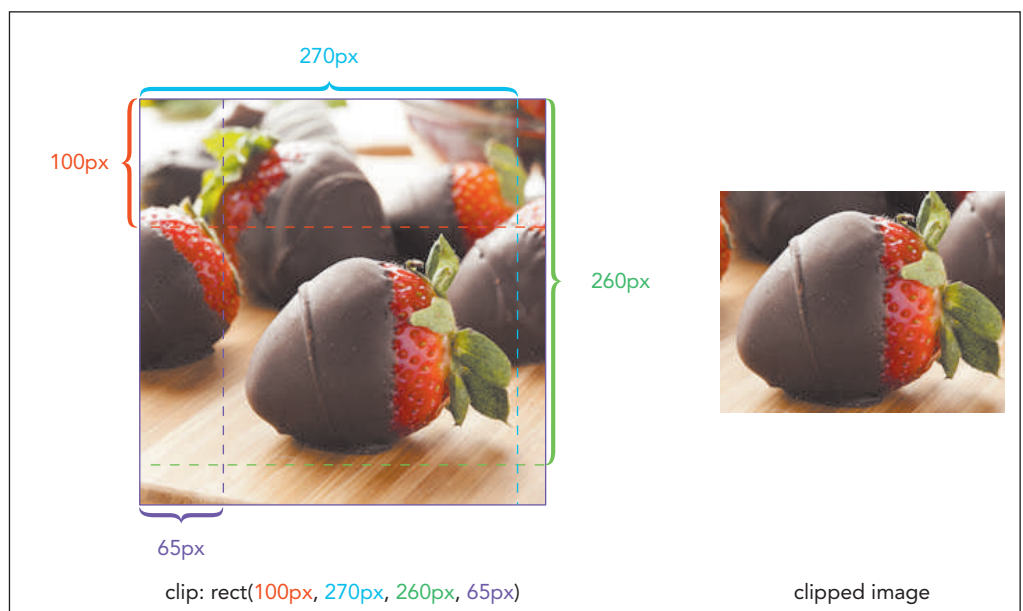
Closely related to the `overflow` property is the `clip` property, which defines a rectangular region through which an element's content can be viewed. Anything that lies outside the boundary of the rectangle is hidden. The syntax of the `clip` property is

```
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  

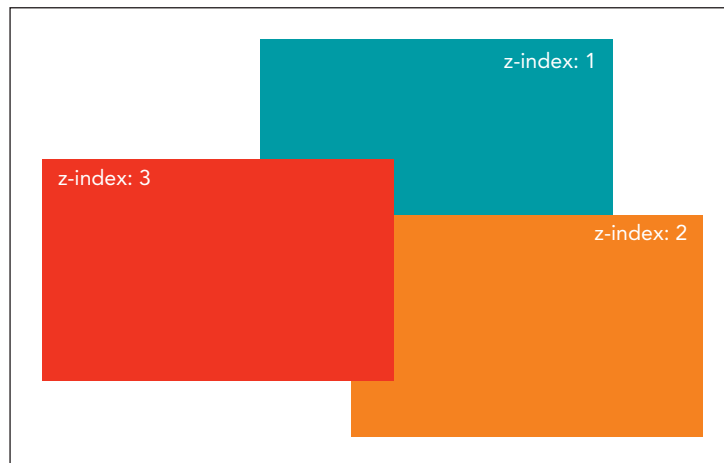
```
clip: auto;
```

## 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;
```

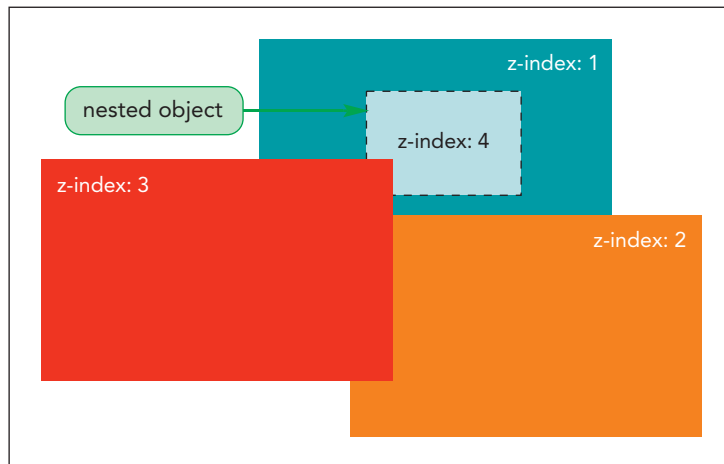
where *value* is a positive or negative integer, or the keyword `auto`. As shown in Figure 3–77, objects with the highest `z-index` values are placed on top of other page objects. A value of `auto` stacks the objects using the default rules.

**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.



### 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

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

5. Provide a style property to display scrollbars when the element content exceeds the element's boundaries.
  - a. `overflow: auto;`
  - b. `overflow: scroll;`
  - c. `overflow: scrollbar;`
  - d. `overflow: true;`
6. An inline image is 400 pixels wide by 300 pixels high. Provide a style rule to clip this image by 10 pixels on each edge.
  - a. `clip: rect(10, 390, 290, 10);`
  - b. `clip: 10;`
  - c. `clip: -10;`
  - d. `clip: 10 390 290 10;`
7. If two elements overlap, the one displayed on top will:
  - a. be listed first in the document order
  - b. have the greater height and width
  - c. have the lower `z-index` value
  - d. have the higher `z-index` value

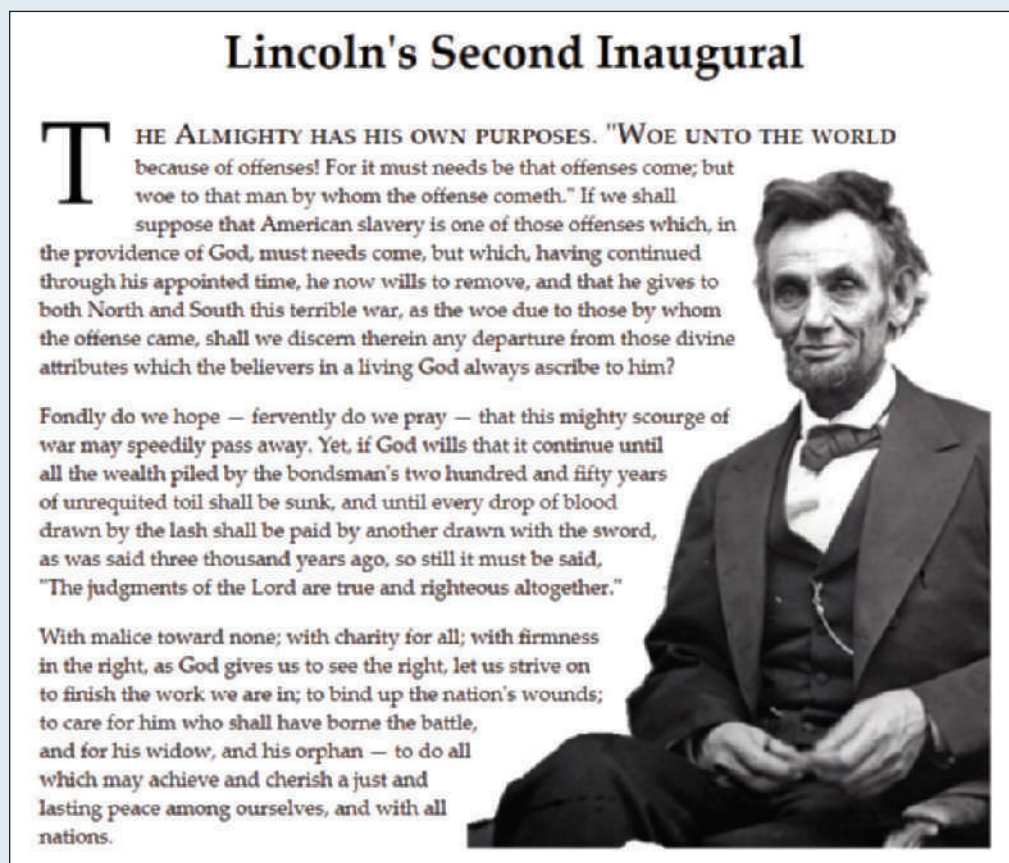


## Coding Challenge 1

Data Files needed for this Coding Challenge: `code3-1_txt.html`, `code3-1_float_txt.css`, `code3-1.css`, `lincoln01.png` - `lincoln10.png`

Figure 3–79 shows an example page containing two applications of floating objects. In the first line of Lincoln's second inaugural speech a drop capital is created by floating the first letter of the first paragraph next to the surrounding text. The text of the speech is wrapped around the image of Lincoln using an irregular line wrap. This effect is created by cutting the Lincoln image into separate strips which are floated and stacked on top of each other. In this Coding Challenge you will explore how to create both effects.

Figure 3–79 Coding Challenge 3-1 example page



Library of Congress, Prints & Photographs Division, Reproduction number LC-DIG-ppmsca-19469 (digital file from original)

Do the following:

1. Open the `code3-1_txt.html` and `code3-1_float_txt.css` files from the `html03 ► code1` folder. Enter *your name* and *the date* in each document and save the files as `code3-1.html` and `code3-1_float.css` respectively.
2. Go to the `code3-1.html` file in your editor. Within the head section insert a `link` element linking the page to the `code3-1_float.css` style sheet file. Take some time to study the content of the page and then save your changes to the file.
3. Go to the `code3-1_float.css` file in your editor.

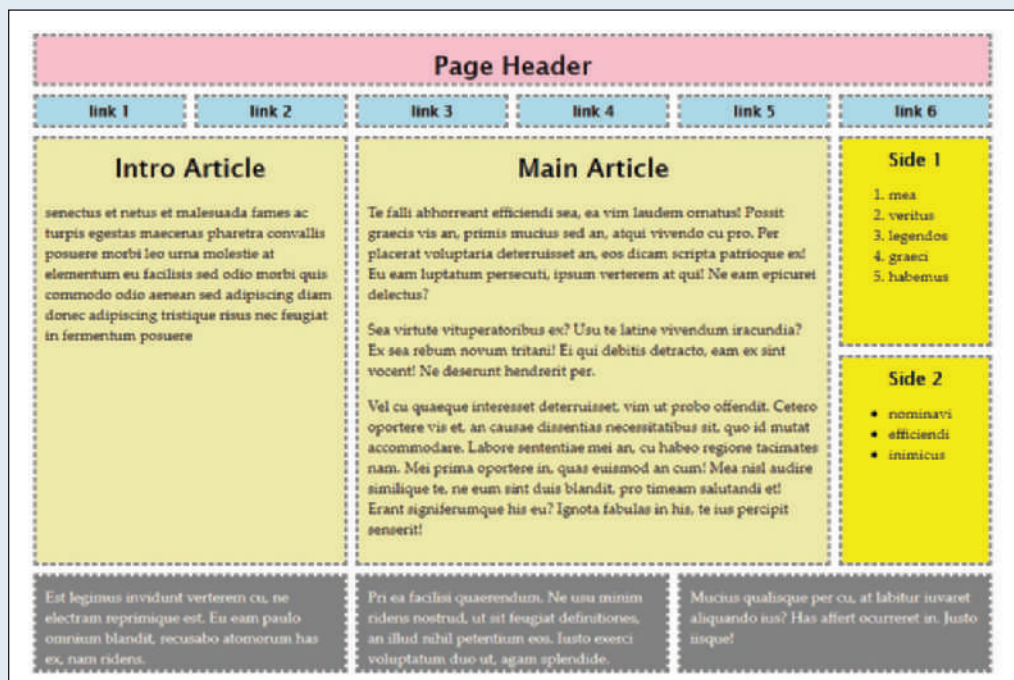
4. To create a drop cap, insert a style rule for the selector `p:first-of-type:first-letter` and add the following styles:
  - a. Float the element on the left margin.
  - b. Set the font size to 4em and the line height to 0.8em.
  - c. Set the size of the right margin and padding space to 0.1em. Set the bottom padding to 0.2em.
5. Display the first line of the speech in small caps by adding a style rule for the selector `p:first-of-type:first-line` that changes the font variant to small-caps and the font size to 1.4em.
6. For all `img` elements create a style rule to set the height of the image to 3.3em and float the image on the right margin, but only when the page is cleared of floats.
7. Save your changes to the style sheet.
8. Open the page in your browser and verify the layout of the page resembles that shown in Figure 3-79.
9. Submit the completed file to your instructor.

## Coding Challenge 2

Data Files needed for this Coding Challenge: `code3-2_txt.html`, `code3-2_grad_txt.css`, `code3-2.css`, `landscape.png`

Figure 3–80 shows a proposed layout for a new web page. At this point the final content is not ready for the web page, so the layout is shown using lorem ipsum text. You’ve been given the HTML code for the page and your challenge is to create the page layout using CSS grid styles.

Figure 3–80 Coding Challenge 3-2 example page



Complete the following:

1. Open the `code3-2_txt.html` and `code3-2_layout_txt.css` files from the `html03 ▶ code2` folder. Enter **your name** and **the date** in each document and save the files as `code3-2.html` and `code3-2_layout.css` respectively.

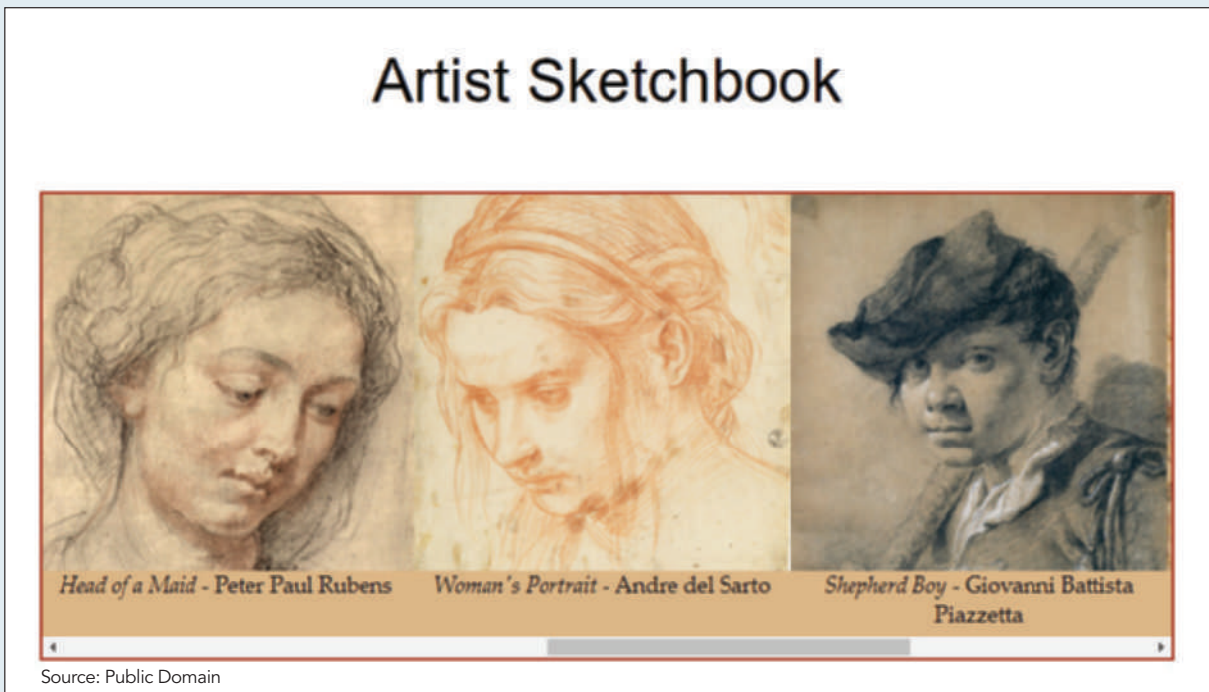
2. Go to the **code3-2.html** file in your editor. Within the head section insert a `link` element linking the page to the `code3-2_layout.css` file. Study the contents of the file, taking note of the structure, element names, and element ids. Save your changes.
3. Go to the **code3-2\_layout.css** file. Create a style rule for the `header`, `footer`, `aside`, `article`, and `a` (hyperlink) elements to set the padding space to 10 pixels and add a 3-pixel gray dashed outline.
4. Create a style rule for the `body` element that:
  - a. Sets the width to 90% of the browser window, ranging from a minimum width of 640 pixels up to a maximum width of 1024 pixels.
  - b. Sets the top/bottom margin to 30 pixels and the left/right margin to `auto`.
  - c. Displays the body as a CSS grid.
  - d. Creates six grid columns each with a width of `1fr`.
  - e. Creates five grid rows with widths of 50 pixels, 30 pixels, `1fr`, `1fr`, and 100 pixels.
  - f. Adds a grid gap of 15 pixels.
5. Display the `a` (hyperlink) element as a block.
6. Set the size of the grid items as follows:
  - a. Have the `header` element span from gridline 1 to gridline -1.
  - b. Have the `article#intro` element span two rows and two columns.
  - c. Have the `article#main` element two rows and three columns.
  - d. Have the `footer` element span two columns.
7. Save your changes to the style sheet.
8. Open the page in your browser and verify the layout of the page resembles that shown in Figure 3–80.
9. Submit the completed file to your instructor.

### Coding Challenge 3

Data Files needed for this Coding Challenge: `code3-3_txt.html`, `code3-3_scroll_txt.css`, `code3-3_styles.css`, `image01.png` - `image09.png`

You can use the CSS positioning and overflow styles to create a scrolling slideshow. Figure 3–81 shows an example of a slideshow consisting of nine sketches by Renaissance masters. You've been given the HTML code for this document and you've been asked to write the style rules to generate the slideshow.

Figure 3–81 Coding Challenge 3-3 example page



Source: Public Domain

Complete the following to create the web page:

1. Open the **code3-3\_txt.html** and **code3-3\_scroll\_txt.css** files from the html03 ► code3 folder. Enter **your name** and **the date** in each document and save the files as **code3-3.html** and **code3-3\_scroll.css** respectively.
2. Go to the **code3-3.html** file in your editor. Link the file to the code3-3\_scroll.css style sheet file. Review the contents of the document and then save your changes to the file.
3. Go to the **code3-3\_scroll.css** file in your editor. Create a style rule for the `section` element with the id "container" with the following styles:
  - a. Set the width of the element to 900 pixels and the height to 370 pixels.
  - b. Horizontally center the element by adding a 10-pixel top/bottom margin and set the left/right margin to `auto`.
  - c. Place the element with relative positioning, setting the top value to 30 pixels and the left value to 0 pixels.
  - d. Add a 2-pixel solid brown outline to the element.
  - e. Have the browser automatically display scrollbars for any overflow content.
4. Create a style rule for every `div` element, setting the width to 300 pixels and the height to 330 pixels. Position the element with absolute positioning.
5. Display every inline image as a block-level element with a width and height of 300 pixels.
6. There are nine `div` elements with ids ranging from "slide1" to "slide9". Set the left position of the elements in 300-pixel increments starting with 0 pixels for slide1, 300 pixels for slide2, 600 pixels for slide3, and so forth up to 2400 pixels for slide9.
7. Save your changes to the style sheet.
8. Open the page in your browser. Verify that the nine images are displayed within a scroll box and that you can using a horizontal scrollbar to scroll through the image list.
9. Submit the completed file to your instructor.



## Coding Challenge 4

Data Files needed for this Coding Challenge: `code3-4_txt.html`, `debug3-4_txt.css`, `code3-4_.css`, `redball.png`

Figure 3–82 shows a completed web page that uses CSS to design the page layout. You've been given the initial HTML and CSS code for this web page, but there are several errors in the CSS stylesheet. Use your knowledge of CSS to locate and fix the errors.

Figure 3–82 Coding Challenge 3-4 example page



Maxim Maksutov/ssutterstock.om; @Shebeko/Shutterstock.com

Do the following:

1. Open the `code3-4_txt.html` and `debug3-4_txt.css` files from the `html03 ▶ code4` folder. Enter **your name** and **the date** in each document and save the files as `code3-4.html` and `debug3-4.css` respectively.
2. Go to the `code3-4.html` file in your editor. Link the page to the `debug3-4.css` style sheet file. Study the contents of the file and then save your changes.
3. Go to the `debug3-4.css` file in your browser.
4. The `body` element should have a width that is 90% of the width of the browser window ranging from a minimum of 600 pixels up to a maximum of 1024 pixels. Fix the syntax errors in the body style rule that defines the width of the web page.
5. The style rule for the `body` element sets up a grid layout for the page. However, there are several errors in defining the grid areas, grid columns, and grid gaps. Fix the syntax errors in the style rule.
6. Go to the style rules in the Grid Areas section that assigns page elements to areas of the grid. Locate and fix the errors in assigning elements to grid areas.

7. The style rules for the horizontal navigation list and the `section` element also define grid styles for those elements. Locate and fix errors in the code that set up the grid columns.
8. The last paragraph within the `section div` selector should be placed with absolute positioning 1 pixel and 5 pixels from the bottom right corner of the container element. However, there is an error in defining the selector. Find and fix the error.
9. Save your changes and open the **code3-4.html** file in your browser. Verify that design of the page resembles that shown in Figure 3–82.
10. Submit the completed file to your instructor.

## PRACTICE

## Review Assignments

**Data Files needed for the Review Assignments: pc\_specials\_txt.html, pc\_specials\_txt.css, 2 CSS files, 8 PNG files, 1 TTF file, 1 WOFF file**

Anne wants you to work on another page for the Pandaisia Chocolates website. This page will contain information on some of the specials offered by the company in March; it will also display a list of some awards that the company has won. As you work on the page, you will use clip art images as placeholders until photographs of the awards are available. A preview of the completed page is shown in Figure 3–83.

Figure 3–83 March Specials web page



© Arina P Habich/Shutterstock.com;

© Alexander Chaikin/Shutterstock.com; © ESB Professional/Shutterstock.com

Anne has already created the page content and some of the design styles to be used in the page. Your job will be to come up with the CSS style sheet to set the page layout.

Complete the following:

1. Use your editor to open the **pc\_specials\_txt.html** and **pc\_specials\_txt.css** files from the `html03 ▶ review` folder. Enter *your name* and *the date* in the comment section of each file, and save them as **pc\_specials.html** and **pc\_specials.css** respectively.
2. Go to the **pc\_specials.html** file in your editor. Within the document head, create links to the `pc_reset2.css`, `pc_styles4.css`, and `pc_specials.css` style sheets.
3. Take some time to study the content and structure of the document, paying careful attention to the use of ids and class names in the file. Save your changes to the file.
4. Go to the **pc\_specials.css** file in your editor. Within the Page Body Styles section, add a style rule for the `body` element that sets the width of the page body to 95% of the browser window width within the range of 640 to 960 pixels. Horizontally center the page body within the window by setting the left and right margins to `auto`.
5. Go to the Image Styles section and create a style rule that displays all `img` elements as blocks with a width of 100%.
6. Anne wants the navigation list to be displayed horizontally on the page. Go to the Horizontal Navigation Styles section and create a style rule for every list item within a horizontal navigation list that displays the list item as a block floated on the left margin with a width of 16.66%.
7. Display every hypertext link nested within a navigation list item as a block.
8. Next, you will create the grid styles for the March Specials page. Go to the Grid Styles section and create a style rule for the `body` element that displays the element as a grid with two columns in the proportion of 2:1 (using `fr` units) with a column grid gap of 20 pixels.
9. Create a style rule for the `header` and `footer` elements that has both elements span the grid from the gridline number 1 to gridline number -1.
10. Create a style rule for the `section` element with the id "sub" that displays that element as a grid consisting of three columns of equal width by repeating the column width `1fr` three times.
11. Go to the Specials Styles section. In this section, you will create styles for the monthly specials advertised by the company. Create a style rule for all `div` elements of the `specials` class that sets the minimum height to 400 pixels and adds a 1 pixel dashed outline around the element with a color value of `rgb(71, 52, 29)`.
12. Go to the Award Styles section. In this section, you will create styles for the list of awards won by Pandaisia Chocolates. Information boxes for the awards are placed within an `aside` element. Create a style rule for the `aside` element that places it using relative positioning, sets its height to 650 pixels, and automatically displays scrollbars for any overflow content.
13. Every information box in the `aside` element is stored in a `div` element. Create a style rule that places these elements with absolute positioning and sets their width to 30%.
14. Position the individual awards within the `awardList` box by creating style rules for the `div` elements with id values ranging from `award1` to `award5` at the following (*top, left*) coordinates: `award1` (80px, 5%), `award2` (280px, 60%), `award3` (400px, 20%), `award4` (630px, 45%), and `award5` (750px, 5%). (Hint: In the `pc_specials.html` file, the five awards have been placed in a `div` element belonging to the `awards` class with id values ranging from `award1` to `award5`.)
15. Save your changes to the style sheet and then open the **pc\_specials.html** file in your browser. Verify that the layout and design styles resemble the page shown in Figure 3–83.

## Case Problem 1

Data Files needed for this Case Problem: `sp_home_txt.html`, `sp_layout_txt.css`, 2 CSS files, 11 PNG files

**Slate & Pencil Tutoring** Karen Cooke manages the website for *Slate & Pencil Tutoring*, an online tutoring service for high school and college students. Karen is overseeing the redesign of the website and has hired you to work on the layout of the site's home page. Figure 3–84 shows a preview of the page you'll create for Karen.

Figure 3–84 Slate & Pencil Tutoring home page



© Monkey Business Images/Shutterstock.com;  
© Courtesy Patrick Carey

Karen has supplied you with the HTML file and the graphic files. She has also given you a base style sheet to initiate your web design and a style sheet containing several typographic styles. Your job will be to write up a layout style sheet according to Karen's specifications.

Complete the following:

1. Using your editor, open the `sp_home_txt.html` and `sp_layout_txt.css` files from the `html03 ▶ case1` folder. Enter **your name** and **the date** in the comment section of each file, and save them as `sp_home.html` and `sp_layout.css` respectively.
2. Go to the `sp_home.html` file in your editor. Within the document head, create links to the `sp_base.css`, `sp_styles.css`, and `sp_layout.css` style sheet files. Study the content and structure of the file and then save your changes to the document.



3. Go to the `sp_layout.css` file in your editor. Go to the Window and Body Styles section. Create a style rule for the `html` element that sets the height of the browser window at 100%.
4. Create a style rule for the page body that sets the width to 95% of the browser window ranging from 640 pixels up to 960 pixels. Horizontally center the page body within the browser window. Finally, Karen wants to ensure that the height of the page body is always at least as high as the browser window itself. Set the minimum height of the browser window to 100%.
5. Add a style rule to display all inline images as blocks.
6. Within the CSS Grid Styles section create a style rule that displays the `body` element as a grid with four columns of length 1fr.
7. Create a style rule for the `img` element with id "logo" so that the logo image spans three columns and has a width of 100%.
8. For the horizontal navigation list and the `footer` element create a style rule so that those elements span four columns. Create a style for the `aside` element to span two columns.
9. Within the Horizontal Navigation List Styles section create a style rule for `li` elements nested within the horizontal navigation list that display each element as a block with a width of 12.5% and floated on the left margin.
10. Within the Section Styles section create a style rule for inline images within the `section` element that sets the width of the image to 50% and centers the image using a top/bottom margin of 0 and a left/right margin of `auto`.
11. Create a style rule for paragraphs within the `section` element that sets the width of the paragraph to 70% and centers the paragraph using a top/bottom margin of 0 and a left/right margin of `auto`.
12. Go to the Customer Comment Styles section and create a style rule for the `aside` element setting the width to 75% and the bottom padding to 30 pixels.
13. The six `aside` elements will be displayed in two columns. For odd-numbered `aside` elements, use the `justify-self` grid property to place the element on the end (right) margin. (Hint: Use the `nth-of-type(odd)` pseudo-class to select the odd-numbered `aside` elements.)
14. Float inline images nested within the `aside` element on the left with a width of 20%.
15. Float paragraphs nested within the `aside` element on the left with a width of 75% and a left margin of 5%.
16. Save your changes to the file and then open the `sp_home.html` file in your browser. Verify that the layout and appearance of the page elements resemble that shown in Figure 3–84.

## Case Problem 2

**Data Files needed for this Case Problem:** `ss_dday_txt.html`, `ss_layout_txt.css`, 1 CSS file, 3 PNG files

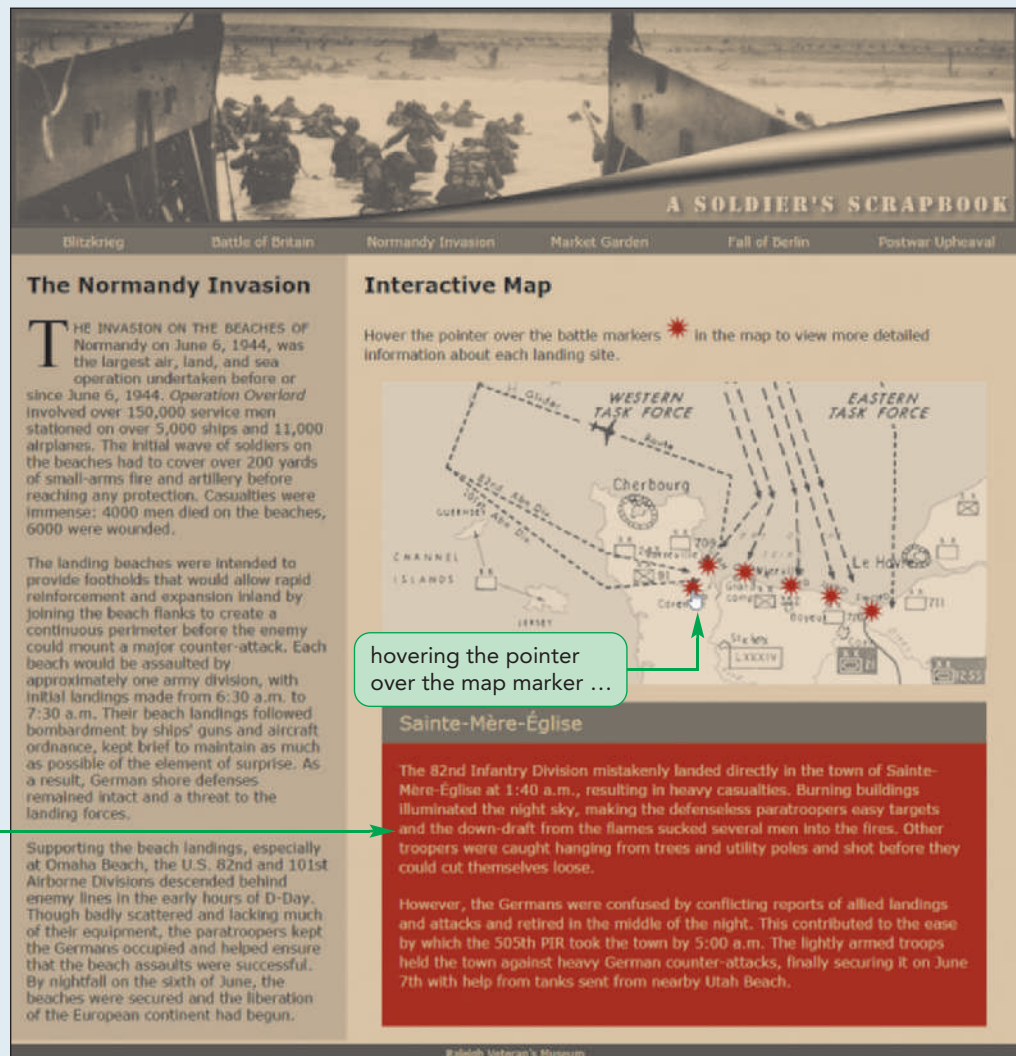
### CHALLENGE

**A Soldier's Scrapbook** Jakob Bauer is a curator at the Veteran's Museum in Raleigh, North Carolina. Currently he is working on an exhibit called *A Soldier's Scrapbook* containing mementos, artifacts, journals, and other historic items from the Second World War. You've been asked to work on a page for an interactive kiosk used by visitors to the exhibit. Jakob has already supplied much of the text and graphics for the kiosk pages but he wants you to complete the job by working on the page layout.

The page you will work on provides an overview of the Normandy beach landings on June 6<sup>th</sup>, 1944. Since this page will be displayed only on the kiosk monitor, whose screen dimensions are known, you'll employ a fixed layout based on a screen width of 1152 pixels.

Jakob also wants you to include an interactive map of the Normandy coast where the user can hover a mouse pointer over location markers to view information associated with each map point. To create this effect, you'll mark each map point as a hypertext link so that you can apply the `hover` pseudo-class to the location. In addition to the interactive map, Jakob wants you to create a drop cap for the first letter of the first paragraph in the article describing the Normandy invasion. Figure 3–85 shows a preview of the page you'll create.

Figure 3–85 Normandy Invasion kiosk page



Source: Chief Photographer's Mate (CPHOM) Robert F. Sargent, U.S. Coast Guard/National Archives and Records Administration;  
Source: U.S. Department of Defense/Wikimedia Commons; © Patrick Carey

Complete the following:

- Using your editor, open the `ss_dday_txt.html` and `ss_layout_txt.css` files from the `html03 ▶ case2` folder. Enter **your name** and **the date** in the comment section of each file, and save them as `ss_dday.html` and `ss_layout.css` respectively.
- Go to the `ss_dday.html` file in your editor. Within the document head, create links to the `ss_styles.css` and `ss_layout.css` style sheet files. Study the content and structure of the document. Note that within the `aside` element is an image for the battle map with the `id` `mapImage`. Also note that there are six marker images enclosed within hypertext links with `ids` ranging from `marker1` to `marker6`. After each marker image are `div` elements of the `mapInfo` class with `IDs` ranging from `info1` to `info6`. Part of your style sheet will include style rules to display these `div` elements in response to the mouse pointer hovering over each of the six marker images.
- Save your changes to the file and then go to the `ss_layout.css` file in your editor.
- Go to the Article Styles section. Within this section, you'll lay out the article describing the Normandy Invasion. Create a style rule to float the `article` element on the left margin and set its width to 384 pixels.

- ✚ **Explore** 5. Jakob wants the first line from the article to be displayed in small capital letters. Go to the First Line and Drop Cap Styles section and create a style rule for the first paragraph of the article element and the first line of that paragraph, setting the font size to 1.25em and the font variant to small-caps. (Hint: Use the `first-of-type` pseudo-class for the paragraph and the `first-line` pseudo-element for the first line of that paragraph.)
- ✚ **Explore** 6. Jakob also wants the first letter of the first line in the article's opening paragraph to be displayed as a drop cap. Create a style rule for the article's first paragraph and first letter that applies the following styles: (a) sets the size of the first letter to 4em in a serif font and floats it on the left, (b) sets the line height to 0.8em, and (c) sets the right and bottom margins to 5 pixels. (Hint: Use the `first-letter` pseudo-element for the first letter of that paragraph.)
- 7. The interactive map is placed within an `aside` element that Jakob wants displayed alongside the Normandy Invasion article. Go the Aside Styles section and create a style rule that sets the width of the aside element to 768 pixels and floats it on the left margin.
- 8. Next, you will lay out the interactive map. The interactive map is placed within a `div` element with the ID `battleMap`. Go to the Map Styles section and create a style rule for this element that sets its width to 688 pixels. Center the map by setting its top/bottom margins to 20 pixels and its left/right margins to `auto`. Place the map using relative positioning.
- 9. The actual map image is placed within an `img` element with the ID `mapImage`. Create a style rule for this element that displays it as a block with a width of 100%.
- 10. Go to the Interactive Map Styles section. Within this section, you'll create style rules that position each of the six map markers onto the battle map. The markers are placed within hypertext links. Create a style rule for every `a` element of the `battleMarkers` class that places the hypertext link using absolute positioning.
- 11. Create style rules for the six `a` elements with IDs ranging from `marker1` to `marker6`, placing them at the following (*top, left*) coordinates:
  - `marker1` (220, 340)
  - `marker2` (194, 358)
  - `marker3` (202, 400)
  - `marker4` (217, 452)
  - `marker5` (229, 498)
  - `marker6` (246, 544)
- 12. The information associated with each map marker has been placed in `div` elements belonging to the `mapInfo` class. Go to the Map Information Styles section and create a style rule that hides this class of elements so that this information is not initially visible on the page.
- ✚ **Explore** 13. To display the information associated with each map maker, you need to create a style rule that changes the map information's `display` property in response to the mouse pointer hovering over the corresponding map marker. Since the map information follows the map marker in the HTML file, use the following selector to select the map information corresponding to the hovered map marker: `a.battleMarkers:hover + div.mapInfo`. Write a style rule for this selector that sets its `display` property to `block`.
- 14. Save your changes to the style sheet and then load `ss_dday.html` in your browser. Verify that a drop cap appears for the first letter of the Normandy Invasion article and the first line of the first paragraph is displayed in small caps. Test the interactive map by first verifying that none of the information about the six battle locations appears on the page unless you hover your mouse pointer over the marker on the battle map. Further verify that when you are not hovering over the battle marker, the information is once again not visible on the page.

## OBJECTIVES

**Session 4.1**

- Create a figure box
- Add a background image
- Add a border to an element
- Create rounded borders
- Create a graphic border

**Session 4.2**

- Create a text shadow
- Create a box shadow
- Create linear and radial gradients
- Set the opacity of an element

**Session 4.3**

- Apply a 2D and 3D transformation
- Apply a CSS filter
- Create an image map

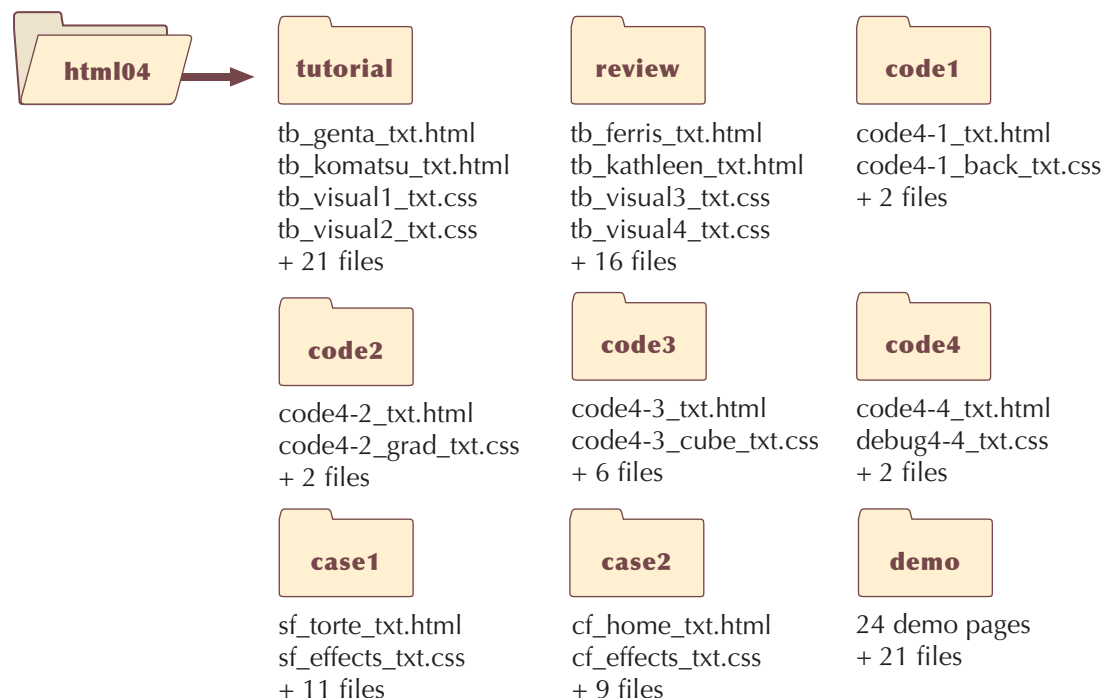
# Graphic Design with CSS

## Creating a Graphic Design for a Genealogy Website

### Case | *Tree and Book*

Kevin Whitmore is the founder of *Tree and Book*, a social networking website for people interested in documenting their family histories, creating online photo albums, and posting stories and information about members of their extended families. He has come to you for help in upgrading the site's design. Kevin wants to take advantage of some of the CSS styles that can be used to add interesting visual effects to his site in order to give his website more impact and visual interest.

## STARTING DATA FILES



# Session 4.1 Visual Overview:

The **background-image** property applies an image file to the element background.

The **background** property defines all background options, including the use of multiple backgrounds.

The **cover** keyword specifies that the background image should completely cover the background.

The **border-left** and **border-right** properties add borders to the left and right edge of the element.

The **border-radius** property creates rounded corners with the specified radius.

The **border-image** property defines an image file used to create a graphic border.

The border images are based on an image file, the size of the slice from the image, and how slices are displayed along the element edge.

The **padding-box** keyword specifies that the background extends through the padding space.

The **content-box** keyword specifies that the background extends only over the element content.

The **no-repeat** keyword specifies that no tiling is done with the background image.

Every border is defined by its width, style, and color.

The **border** property adds a border around all sides of the element.

```

/* Background Styles */
html {
  background-image: url(tb_back1.png);
}

article {
  background: url(tb_back2.png) bottom right / 15% no-repeat content-box,
             url(tb_back3.png) bottom left / 15% no-repeat content-box,
             url(tb_back4.png) 100%/cover no-repeat padding-box;
}

/* Border Styles */
body {
  border-left: 1px solid rgb(51, 51, 51);
  border-right: 1px solid rgb(51, 51, 51);
}

aside {
  border: 4px double rgb(45, 93, 62);
  border-radius: 30px;
}

body > article > header > figure {
  border-width: 25px;
  border-style: solid;
  border-image: url(tb_border.png) 50 repeat;
}

```

Logo Design Studio Pro;  
Source: wiki Media;  
© imtmphoto/Shutterstock.com



# Backgrounds and Borders

The `tb_back1.png` image is tiled to fill the element background.

The background image `tb_back4.png` covers the entire article's padding space.

The border image is based on the `tb_border.png` file.

The background image `tb_back3.png` is placed at the lower-left corner of the article.

The aside element has a rounded corner with a radius of 30 pixels.

The aside element has a 4-pixel wide double border.

The background image `tb_back2.png` is placed at the right corner of the article.

The screenshot shows a web page for 'The Komatsu Family' with the following content:

- Header: 'Tree and Book' logo, navigation links (Home, Register, Directory, Genealogy, Search), and 'Contact Us | My Account'.
- Main Title: 'The Komatsu Family'.
- Central Image: A family photo of Genta, Mika, and their children.
- Text: 'Genta Komatsu was born in Madano, Japan in 1946, the son of Goro Komatsu and Hisako (Sato). The family emigrated to San Francisco in 1960. Genta became a U.S. citizen in 1968 and attended the University of San Francisco, receiving a Master's degree in Electrical Engineering. In 1974 he met and married Mika (Aoki), daughter of Yori and Marie Aoki. Mika was a registered nurse and worked at several hospitals in the Bay Area. Genta passed away from pancreatic cancer in 2019 and is survived by his brother Michi, wife Mika, son Ikko, daughter-in-law Suzuko, and grandson Hiroji. Mika lives in retirement in Phoenix near her family and her energetic grandson.'
- Family Links: A rounded box containing links for Family Tree, Scrapbook, Timeline, Message Board, and Extended Family, along with birth dates for Genta (1946-2019), Mika (b. 1921), Yori (b. 1922), Suzuko (Endo) (b. 1961), and Hiroji (b. 2011).
- Footer: Navigation links (Family Histories, Family Trees, Galleries, Calendars, Forums, Getting Started, Contributor FAQ, Research Sites, Interest Groups, Government Sites, About Tree and Book, Descendants, Privacy, Terms, Help) and copyright notice 'Tree and Book © 2021 English (US)'.

Source: Wikimedia Commons; Design Studio Pro; imtmphoto/Shutterstock.com; imtmphoto/Shutterstock.com

## Creating Figure Boxes

So far your work with CSS visual design styles has been limited to typographical styles and styles that modify the page's color scheme. In this tutorial, you'll explore other CSS styles that allow you to add figure boxes, background textures, background images, and three-dimensional effects to your web pages.

You'll start by examining how to work with figure boxes. In books and magazines, figures and figure captions are often placed within a separate box that stands apart from the main content of the article, using the following `figure` and `figcaption` elements:

```
<figure>
  content
  <figcaption>caption text</figcaption>
</figure>
```

where `content` is the content that will appear within the figure box and `caption text` is the description text that accompanies the figure. The `figcaption` element is optional and can be placed either directly before or directly after the figure box content. For example, the following code marks a figure box containing the `tb_komatsu.png` image file with the caption *(L-R): Ikko, Mika, Hiroji, Genta, Suzuko*.

```
<figure>
  
  <figcaption>(L-R): Ikko, Mika, Hiroji, Genta, Suzuko</figcaption>
</figure>
```

### TIP

The semantic difference between the `figure` and `aside` elements is that the `figure` element should be used for content that is directly referenced from within an article while the `aside` element is used for extraneous content.

While the `figure` element is used to contain an image file, it can also be used to mark any page content that you want to stand apart from the main content of an article. For instance, the `figure` element could contain a text excerpt, as the following code demonstrates:

```
<figure>
  <p>'Twas brillig, and the slithy toves<br />
    Did gyre and gimble in the wabe;<br />
    All mimsy were the borogoves,<br />
    And the mome raths outgrabe.</p>
  <figcaption>
    <cite>Jabberwocky, Lewis Carroll, 1832-98</cite>
  </figcaption>
</figure>
```

Kevin plans on using figure boxes throughout the Tree and Book website to mark up family and individual photos along with descriptive captions. He's created a set of sample pages for the Komatsu family that you will work on to learn about HTML and CSS visual elements and styles. Open the family's home page and create a figure box displaying the family portrait along with a descriptive caption.

### To create a figure box:

1. Use your editor to open the `tb_komatsu_txt.html` file from the `html04` ► tutorial folder. Enter **your name** and **the date** in the comment section of the file and save it as `tb_komatsu.html`.

For this web page, you'll work with a new style sheet named `tb_visual1.css`. Kevin has already created a reset style sheet and a typographical style sheet in the `tb_reset.css` and `tb_styles1.css` files respectively.

2. Within the document head, insert the following `link` elements to link the page to the `tb_reset.css`, `tb_styles1.css`, and `tb_visual1.css` style sheet files.
 

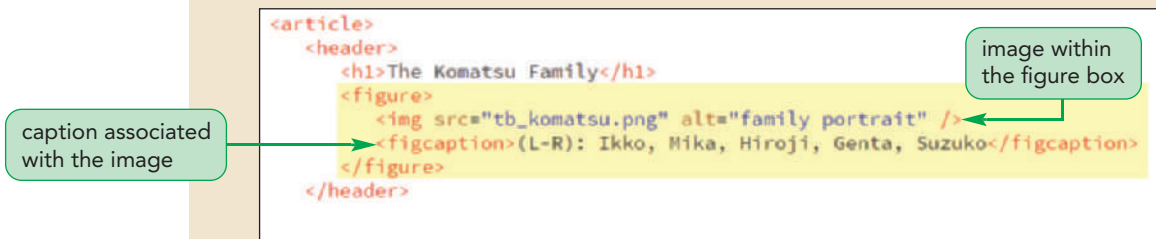
```
<link href="tb_reset.css" rel="stylesheet" />
<link href="tb_styles1.css" rel="stylesheet" />
<link href="tb_visual1.css" rel="stylesheet" />
```
3. Scroll down to the `article` element and, directly after the `h1` element, insert the following code for the figure box displaying the Komatsu family portrait.
 

```
<figure>
  
  <figcaption>(L-R): Ikko, Mika, Hiroji,
            Genta, Suzuko
</figcaption>
</figure>
```

Figure 4–1 highlights the code for the family portrait figure box.

Figure 4–1

## Inserting a figure box



4. Take some time to review the content and structure of the rest of the document and then save your changes to the file.

Format the appearance of the figure box by adding new style rules to the `tb_visual1.css` style sheet file.

## To format and view the figure box:

1. Use your editor to open the `tb_visual1.txt.css` files from the `html04` ► tutorial folder. Enter **your name** and **the date** in the comment section of the file and save it as `tb_visual1.css`.
2. Scroll down to the Figure Box Styles section at the bottom the document and insert the following style rule for the `figure` element:
 

```
figure {
  margin: 20px auto 0px;
  width: 80%;
}
```
3. Add the following style to format the appearance of the image within the figure box:
 

```
figure img {
  display: block;
  width: 100%;
}
```



4. Finally, insert the following rule for the figure caption:

```
figure figcaption {
  background-color: white;
  font-family: 'Palatino Linotype', Palatino,
               'Times New Roman', serif;
  font-style: italic;
  padding: 10px 0;
  text-align: center;
}
```

Figure 4–2 highlights the style rules for the figure box, image, and caption.

**Figure 4–2****Formatting the figure box and caption**

figure box is 80% of the width of the header and centered horizontally

figure image is displayed as a block with a width equal to the figure box

figure caption is centered and displayed in a serif italic font on a white background

```
/* Figure Box Styles */
figure {
  margin: 20px auto 0px;
  width: 80%;
}
figure img {
  display: block;
  width: 100%;
}
figure figcaption {
  background-color: white;
  font-family: 'Palatino Linotype', Palatino, 'Times New Roman', serif;
  font-style: italic;
  padding: 10px 0;
  text-align: center;
}
```

5. Save your changes to the file and then open the **tb\_komatsu.html** file in your browser. Figure 4–3 shows the initial appearance of the page.

Figure 4–3

## Initial design of the Komatsu family page

Home Register Directory Genealogy Search

## The Komatsu Family

(L-R): Mika, Mika, Hiroji, Genta, Suzuko

Genta Komatsu was born in Hadano, Japan in 1946, the son of Goro Komatsu and Hisako (Sato). The family emigrated to San Francisco in 1960. Genta became a U.S. citizen in 1968 and attended the University of San Francisco, receiving a Master's degree in Electrical Engineering. In 1974 he met and married Mika (Aoki), daughter of Yori and Marie Aoki. Mika was a registered nurse and worked at several hospitals in the Bay Area.

Genta passed away from pancreatic cancer in 2019 and is survived by his brother Michi, wife Mika, son Ikko, daughter-in-law Suzuko, and grandson Hiroji. Mika lives in retirement in Phoenix near her family and her energetic grandson.

### Family Links

- [Family Tree](#)
- [Scrapbook](#)
- [Timeline](#)
- [Message Board](#)
- [Extended Family](#)

- [Genta \(1946 - 2019\)](#)
- [Mika \(b. 1951\)](#)
- [Ikko \(b. 1977\)](#)
- [Suzuko \(Endo\) \(b. 1981\)](#)
- [Hiroji \(b. 2011\)](#)

Family Histories Getting Started About Tree and Book Tree and Book © 2021 English (US)

Family Trees Genealogy FAQ Developers

Galleries Research Sites Privacy

Calendars Interest Groups Terms

Forums Government Sites Help

Source: Design Studio Pro; Source: Wikimedia Commons; © imtmphoto/Shutterstock.com

With all of the content for the Komatsu Family page now added, you will start working on enhancing the page's appearance, starting with the CSS background styles.

### Choosing your Graphic File Format

Graphic files on the web fall into two basic categories: vector images and bitmap images. A **vector image** is an image comprised of lines and curves that are based on mathematical functions. The great advantage of vector images is that they can be easily resized without losing their clarity and vector files tend to be compact in size. The most common vector format for the web is **SVG (Scalable Vector Graphics)**, which is an XML markup language that can be created using a basic text editor and knowledge of the SVG language.

A **bitmap image** is an image that is comprised of pixels in which every pixel is marked with a different color. Because a graphic file can be comprised of thousands of pixels, the file size of a bitmap image is considerably larger than the file size of a vector image. The most common bitmap formats on the web are GIF, JPEG, and PNG.

**GIF (Graphic Interchange Format)** is the oldest standard with a palette limited to 256 colors. GIF files, which tend to be large, have two advantages: first, GIFs support transparent colors and second, GIFs can be used to create animated images. Because GIFs have a limited color palette, they are unsuitable for photos. The most popular photo format is **JPEG (Joint Photographic Experts Group)**, which supports a palette of over 16 million colors. JPEGs also support file compression, allowing a bitmap image to be stored at a smaller file size than would be possible with other bitmap formats. JPEGs do not support transparent colors or animations.

The **PNG (Portable Network Graphics)** format was designed to replace GIFs with its support for several levels of transparent colors and palette of millions of colors. A PNG file can also be compressed, creating a file that is considerably smaller and, therefore, takes up considerably less space than its equivalent GIF file. PNG files also contain color correction information so that PNGs can be accurately rendered across a variety of display devices.

In choosing a graphic format for your website, the most important consideration is often file size; you want to choose the smallest size that still gives you an acceptable image. This combination means that users will view a quality image but they will not have to wait for the graphic file to download. In addition to file size, you want to choose a format that supports a large color palette. For these reasons, most graphics on the web are now in either JPEG or PNG format, though GIFs are still often found on legacy sites.

## Exploring Background Styles

Thus far, your design choices for backgrounds have been limited to color using either the RGB or HSL color models. CSS also supports the use of images for backgrounds through the following `background-image` style:

```
background-image: url(url);
```

where `url` specifies the name and location of the background image. For example, the following style rule uses the `trees.png` file as the background of the page body.

```
body {  
    background-image: url(trees.png);  
}
```

This code assumes that the `trees.png` file is in the same folder as the style sheet; if the figure is not in the same folder, then you will have to include path information pointing to the folder location in which the image file resides.

## Tiling a Background Image

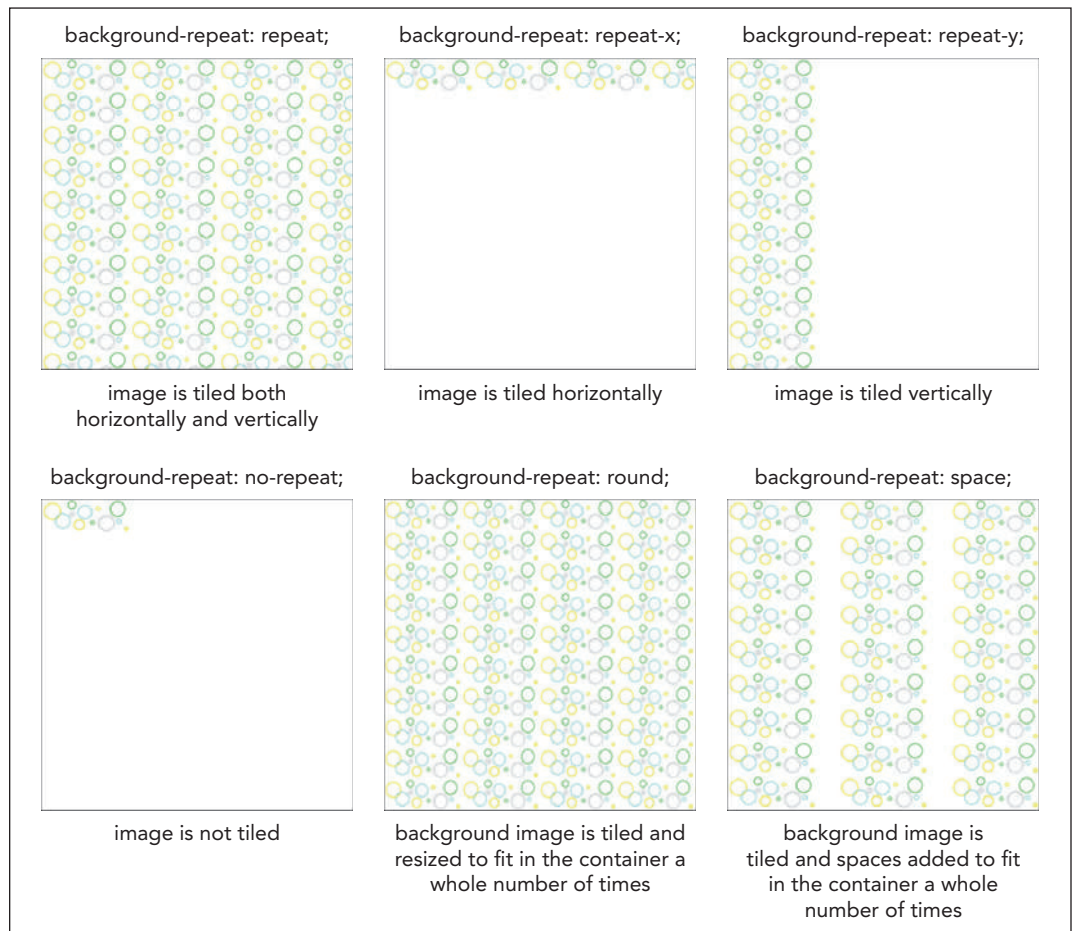
The default browser behavior is to place the background image at the top-left corner of the element and repeat the image in both the vertical and horizontal direction until the background is filled. This process is known as **tiling** because of its similarity to the process of filling up a floor or other surface with tiles.

You can specify the type of tiling to be applied to the background image, or even turn off tiling, by applying the following `background-repeat` style:

```
background-repeat: type;
```

where *type* is `repeat` (the default), `repeat-x`, `repeat-y`, `no-repeat`, `round`, or `space`. Figure 4–4 displays the effect of each `background-repeat` type.

**Figure 4–4** Examples of `background-repeat` types



### Adding a Background Image

- To add an image to the background, use the CSS style

```
background-image: url(url);
```

where *url* specifies the name and location of the background image.

- To specify how the image should be tiled, use

```
background-repeat: type;
```

where *type* is *repeat* (the default), *repeat-x*, *repeat-y*, *no-repeat*, *round*, or *space*.

Kevin has supplied you with an image file, `tb_back1.png` to fill the background of the browser window. Use the default option for tiling so that the image is displayed starting from the top-left corner of the window and repeating until the entire window is filled.

### To add a background image to the browser window:

1. Return to the `tb_visual1.css` file in your editor.
2. Go to the HTML Styles section and add the following style rule to change the background of the browser window:

```
html {  
    background-image: url(tb_back1.png);  
}
```

Note that because you are using the default setting for tiling the background image, you do not need to include the `background-repeat` style rule. Figure 4–5 highlights the new style rule.

Figure 4–5

### Defining a background image

tiles the `tb_back1.png` image file across the browser window background

```
/* HTML Styles */  
html {  
    background-image: url(tb_back1.png);  
}
```

3. Save your changes to the file and then reload `tb_komatsu.html` in your browser. Figure 4–6 shows the tiled background in the browser window.

Figure 4–6

Tiled background image in the browser window



Note that the page body covers part of the tiled images in the browser window. However, even though the background images are hidden, the tiling still continues behind the page body.

## Attaching the Background Image

A background image is attached to its element so that as you scroll through the element content, the background image scrolls with it. You can change the attachment using the following `background-attachment` property

```
background-attachment: type;
```

where *type* is `scroll` (the default), `fixed`, or `local`. The `scroll` type sets the background to scroll with the element content. The `fixed` type creates a background that stays in place even as the element content is scrolled horizontally or vertically. Fixed backgrounds are sometimes used to create **watermarks**, which are translucent graphics displayed behind the content with a message that the content material is copyrighted or in draft form or some other message directed to the reader. The `local` type is similar to `scroll` except that it is used for elements, such as scroll boxes, to allow the element background to scroll along with the content within the box.

## Setting the Background Image Position

By default, browsers place the background image in the element's top-left corner. You can place the background image at a different position using the following `background-position` property:

```
background-position: horizontal vertical;
```

### TIP

Background coordinates are measured from the top-left corner of the background to the top-left corner of the image.

where *horizontal* and *vertical* provide the coordinates of the image within the element background expressed using one of the CSS units of measure or as a percentage of the element's width and height. For example, the following style places the image 10% of the width of the element from the left edge of the background and 20% of the element's height from the background's top edge.

```
background-position: 10% 20%;
```



If you specify a single value, the browser applies that value to both the horizontal and vertical position. Thus, the following style places the background image 30 pixels from the element's left edge and 30 pixels down from the top edge.

```
background-position: 30px;
```

You can also place the background image using the keywords `left`, `center`, and `right` for the horizontal position and `top`, `center`, and `bottom` for the vertical position. The following style places the background image in the bottom-right corner of the element.

```
background-position: right bottom;
```

Typically, the `background-position` property is only useful for non-tiled images because, if the image is tiled, the tiled image fills the background and it usually doesn't matter where the tiling starts.

## Defining the Extent of the Background

You learned in Tutorial 2 that every block element follows the Box Model in which the element content is surrounded by a padding space and beyond that a border space (see Figure 2-38). However, the element's background is defined, by default, to extend only through the padding space and not to include the border space. You can change this definition using the following `background-clip` property:

```
background-clip: type;
```

### TRY IT

You can explore the impact of different CSS background styles using the `demo_background.html` file in the `html04` demo folder.

where `type` is `content-box` (to extend the background only through the element content), `padding-box` (to extend the background through the padding space), or `border-box` (to extend the background through the border space). For example, the following style rule defines the background for the page body to extend only as far as the page content. The padding and border spaces would not be considered part of the background and thus would not show any background image.

```
body {  
    background-clip: content-box;  
}
```

Because the background extends through the padding space by default, all coordinates for the background image position are measured from the top-left corner of that padding space. You can choose a different context by applying the following `background-origin` property:

```
background-origin: type;
```

where `type` is once again `content-box`, `padding-box`, or `border-box`. Thus, the following style rule places the background image at the bottom-left corner of the page body content and not the bottom-left corner of the padding space (which would be the default).

```
body {  
    background-position: left bottom;  
    background-origin: content-box;  
}
```

Based on this style rule, the padding space of page body would not have any background image or color, other than what would be defined for the browser window itself.

## Sizing and Clipping an Image

The size of the background image is equal to the size stored in the image file. To specify a different size, apply the following `background-size` property:

```
background-size: width height;
```

where *width* and *height* are the width and height of the image in one of the CSS units of length or as a percentage of the element's width and height. The following style sets the size of the background image to 300 pixels wide by 200 pixels high.

```
background-size: 300px 200px;
```

CSS also supports the sizing keywords `auto`, `cover`, and `contain`. The `auto` keyword tells the browser to automatically set the width or height value based on the dimensions of the original image. The following style sets the height of the image to 200 pixels and automatically scales the width to keep the original proportions of the image:

```
background-size: auto 200px;
```

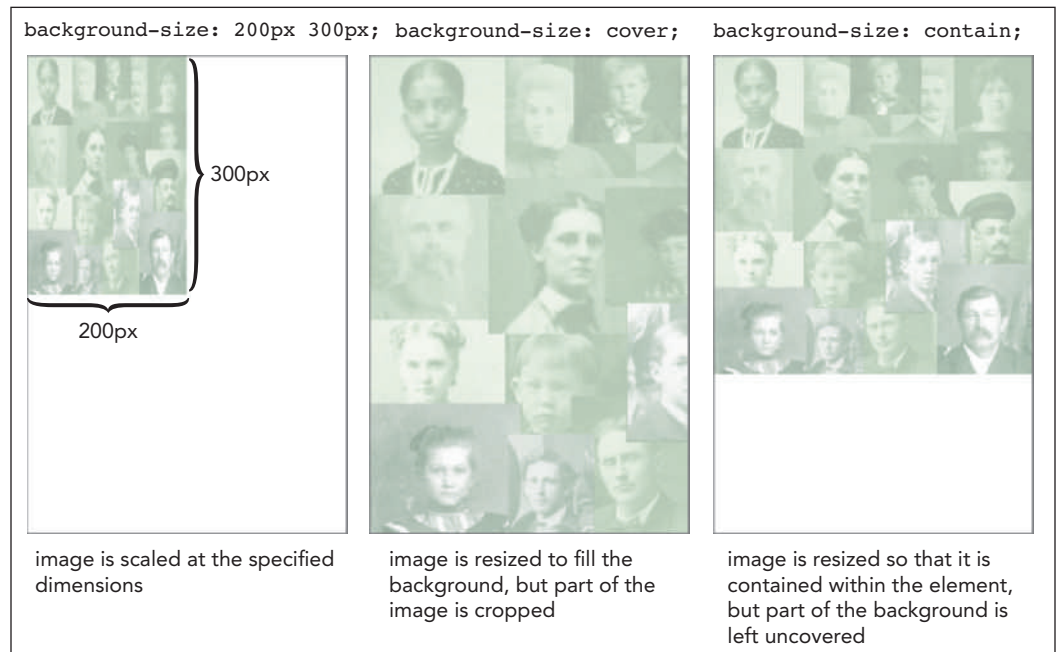
### TIP

If you specify only one size value, the browser applies it to the image width and scales the height proportionally.

The `cover` keyword tells the browser to resize the image to cover all of the element background while still retaining the image proportions. Depending on the size of the element, this could result in some of the background image being cropped. The `contain` keyword scales the image so that it's completely contained within the element, even if that means that not all of the element background is covered. Figure 4–7 displays examples of a background set to a specific size, as well as resized to either cover the background or to have the image completely contained within the background.

Figure 4–7

### Examples of background-size types



Source: Wikimedia Commons



## REFERENCE

### Setting Background Image Options

- To specify how the image is attached to the background, use  
`background-attachment: type;`  
 where *type* is *scroll* (the default), *fixed*, or *local*.
- To set the position of the background image, use  
`background-position: horizontal vertical;`  
 where *horizontal* and *vertical* provide the coordinates of the image within the element background.
- To define the extent of the background, use  
`background-clip: type;`  
 where *type* is *content-box*, *padding-box* (the default), or *border-box*.
- To define how position coordinates are measured, use  
`background-origin: type;`  
 where *type* is *content-box*, *padding-box* (the default), or *border-box*.

## The background Property

All of these different background options can be organized in the following `background` property:

```
background: color url(url) position / size repeat attachment
origin clip;
```

### TRY IT

You can explore the CSS background style using the `demo_background2.html` file in the `html04 ▶ demo` folder.

where *color* is the background color, *url* is the source of the background image, *position* is the image's position, *size* sets the image size, *repeat* sets the tiling of the image, *attachment* specifies whether the image scrolls with the content or is fixed, *origin* defines how positions are measured on the background, and *clip* specifies the extent over which the background is spread. For example, the following style rule sets the background color to ivory and then uses the `draft.png` file as the background image fixed at the horizontal and vertical center of the page body and sized at 10% of the body's width and height:

```
body {
    background: ivory url(draft.png)
               center center / 10% 10%
               no-repeat fixed content-box content-box;
}
```

The rest of the property sets the image not to repeat and to use the content box for defining the background origin and clipping. Note that the page body will have an ivory background color at any location where the `draft.png` image is not displayed. If you don't specify all of the option values, the browser will assume the default values for the missing options. Thus, the following style rule places the `draft.png` at the horizontal and vertical center of the page body without tiling:

```
body {
    background: ivory url(draft.png) center center no-repeat;
}
```

**TIP**

The background property includes the "/" character only when you need to separate the image position value from the image size value.

Since no *size*, *attachment*, *origin*, and *clip* values are specified, the size of the image will be based on the dimensions from the image file, the image will scroll with the body content, and the background origin and clipping will extend through the page body's padding space.

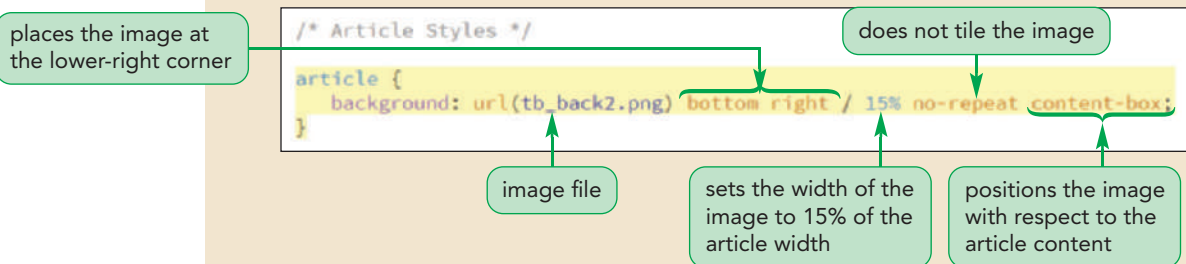
Kevin wants you to include a semi-transparent image of the family patriarch, Genta Komatsu, as a background image placed in the lower-right corner of the article on the Komatsu family. Add a style rule to the `tb_visual1.css` file to display the `tb_back2.png` image within that element without tiling.

**To add a background image to the page article:**

1. Return to the `tb_visual1.css` file in your editor and scroll down to the Article Styles section.
2. Add the following style rule:

```
article {
    background: url(tb_back2.png) bottom right / 15%
               no-repeat content-box;
}
```

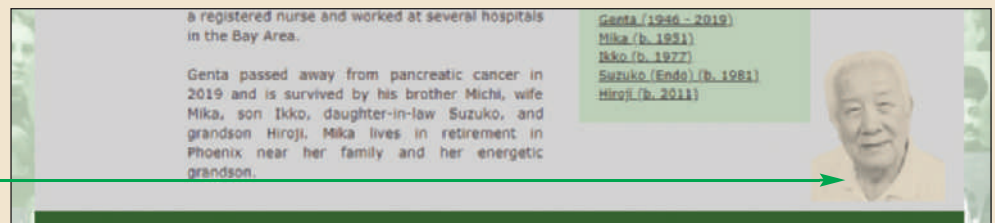
Figure 4–8 highlights the style rule applied to the page article.

**Figure 4–8****Adding a background to the page article**

3. Save your changes and then reload `tb_komatsu.html` in your browser. Figure 4–9 shows the placement of the background image.

**Figure 4–9****Placement of the background image**

background image placed in lower-right corner of the article content with no tiling



Source: Wikimedia Commons; © imtmphoto/Shutterstock.com

Kevin likes the addition of the image of Genta Komatsu and would like you to add another background image showing the family matriarch, Mika Komatsu, and a third image giving the article a paper-textured background.

## Adding Multiple Backgrounds

To add multiple backgrounds to the same element, you list the backgrounds in the following comma-separated list:

```
background: background1, background2, ...;
```

where *background1*, *background2*, and so on are the properties for each background. For example the following style rule applies three different backgrounds to the `header` element:

```
header {
  background: url(back2.png) top left no-repeat,
             url(back1.png) bottom right no-repeat,
             rgb(191, 191, 191);
}
```

### TIP

Always list the background color last so that it provides the foundation for your background images.

Backgrounds are added in the reverse order in which they're listed in the style rule. In this style rule, the background color is applied first, the `back1.png` background image is placed on top of that, and finally the `back2.png` background image is placed on top of those two backgrounds.

Individual background properties can also contain multiple options placed in a comma-separated list. The following style rule creates the same multiple backgrounds for the `header` element without using the `background` property:

```
header {
  background-image: url(back2.png), url(back1.png);
  background-position: top left, bottom right;
  background-repeat: no-repeat;
  background-color: rgb(191, 191, 191);
}
```

Note that if a background style is listed once, it is applied across all of the backgrounds. Thus the `background-color` and the `background-repeat` properties are used in all the backgrounds.

Revise the style rule for the `article` element to add two more backgrounds.

### To add a background image to the page article:

1. Return to the `tb_visual1.css` file in your editor and return to the Article Styles section.
2. Type a comma after the first background listed for the `article` element and before the semicolon (;), then press **Enter**.
3. Be sure the insertion point is before the semicolon (;), then add the following code to display two more background images followed by a background color:

```
url(tb_back3.png) bottom left / 15% no-repeat content-box,
url(tb_back4.png) 100%/cover no-repeat,
rgb(211, 211, 211)
```

The background color acts as a fallback design element and will not be displayed except for browsers that are incapable of displaying background images. Figure 4–10 displays the code for the multiple backgrounds applied to the page article.

The properties for multiple backgrounds need to be separated by commas.

**Figure 4–10** Adding multiple background images

places the second background image at the lower-left corner of the article content with no tiling and a width of 15%

places the third background image, scaled to cover all of the padding box of the article without repeating

```
/* Article Styles */
article {
  background: url(tb_back2.png) bottom right / 15% no-repeat content-box,
             url(tb_back3.png) bottom left / 15% no-repeat content-box,
             url(tb_back4.png) 100%/cover no-repeat,
             rgb(211, 211, 211);
}
```

commas used to separate one background from the next

uses a gray color as the background if the browser doesn't support background images

**Trouble?** Be sure your code matches the code in Figure 4–10, including the commas used to separate the components in the list and the ending semicolon.

4. Save your changes and then reload `tb_komatsu.html` in your browser. Figure 4–11 shows the three background images displayed with the article.

**Figure 4–11** Revised background for the page article

Source: Wikimedia Commons; © imtmphoto/Shutterstock.com

Kevin is pleased with the revised backgrounds for the browser window and the page article. Next, you will explore how to work with CSS border properties.

### Blending Backgrounds

Multiple backgrounds are stacked on top of each other, with the first background listed in the code placed on top of subsequent backgrounds. By default, the backgrounds on the top of the stack will obscure the lower-ordered backgrounds unless there is gap or a transparent color that allows the backgrounds at the bottom of the stack to appear. Thus, each background acts as its own background layer, separate in appearance from other backgrounds.

To combine multiple backgrounds into a single background, use the following `background-blend-mode` style:

```
background-blend-mode: type;
```

where *type* defines the method by which the backgrounds are combined. Possible *type* values include:

- `normal` backgrounds are stacked (the default)
- `multiply` background colors are multiplied, leading to a darker image combined of several colors
- `overlay` background colors are mixed to reflect the lightness and darkness of the colors
- `darken` backgrounds are replaced with the darkest of background colors
- `lighten` backgrounds are replaced with the lightest of the background colors

Several other *type* options are also available. You can specify a comma-separated list of blend values, so that each background layer is blended in a different way with all the other backgrounds.

#### TRY IT

To explore the impact of the `background-blend-mode` style on multiple backgrounds, open the `demo_blend.html` file in the `html04` ► `demo` folder.

## Working with Borders

So far, you have only worked with the content, padding, and margin spaces from the CSS Box model. Now, you will examine the border space that separates the element's content and padding from its margins and essentially marks the extent of the element as it is rendered on the page.

### Setting Border Width and Color

CSS supports several style properties that are used to format the border around each element. As with the margin and padding styles, you can apply a style to the top, right, bottom, or left border, or to all borders at once. To define the thickness of a specific border, use the property

```
border-side-width: width;
```

where *side* is either `top`, `right`, `bottom`, or `left` and *width* is the width of the border in one of the CSS units of measure. For example, the following style sets the width of the bottom border to 10 pixels.

```
border-bottom-width: 10px;
```

Border widths also can be expressed using the keywords `thin`, `medium`, or `thick`; the exact application of these keywords depends on the browser. You can define the border widths for all sides at once using the `border-width` property

```
border-width: top right bottom left;
```

where *top*, *right*, *bottom*, and *left* are the widths of the matching border. As with the `margin` and `padding` properties, if you enter one value, it's applied to all four

borders; two values set the width of the top/bottom and left/right borders, respectively; and three values are applied to the top, left/right, and bottom borders, in that order. Thus, the following property sets the widths of the top/bottom borders to 10 pixels and the left/right borders to 20 pixels:

```
border-width: 10px 20px;
```

The color of each individual border is set using the property

```
border-side-color: color;
```

where *side* once again specifies the border side and *color* is a color name, color value, or the keyword `transparent` to create an invisible border. The color of the four sides can be specified using the following `border-color` property

```
border-color: top right bottom left;
```

where *top right bottom left* specifies the side to which the color should be applied. Thus, the following style uses gray for the top and left borders and black for the right and bottom borders:

```
border-color: gray black black gray;
```

If no border color is specified, the border will use the text color assigned to the element.

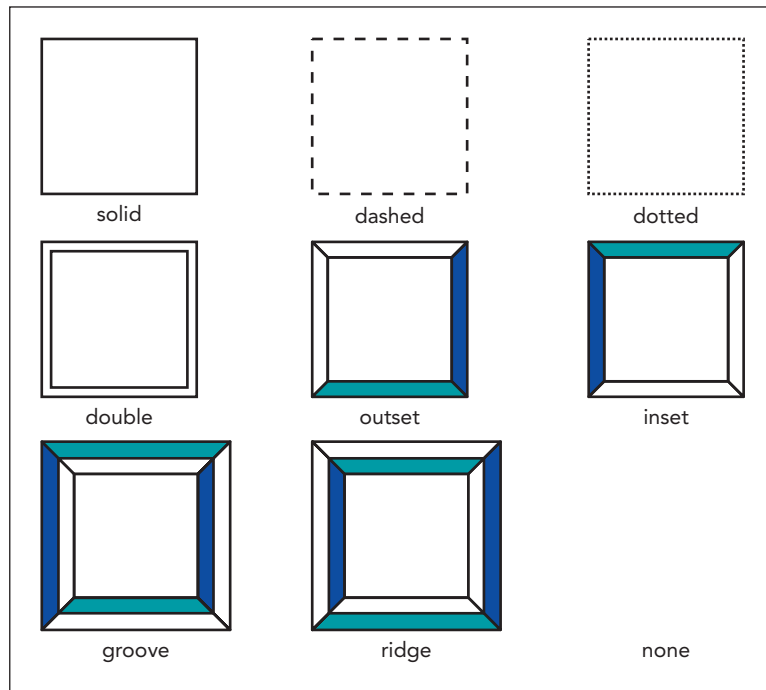
## Setting the Border Design

CSS allows you to further define the appearance of borders using the following border styles:

```
border-side-style: style;
```

where *side* once again indicates the border side and *style* specifies one of the nine border styles displayed in Figure 4–12.

Figure 4–12 Examples of border styles



Or to specify styles for all four borders use the property:

```
border-style: top right bottom left;
```

As with the other border rules, you can modify the style of all borders or combinations of the borders. For example, the following style uses a double line for the top/bottom borders and a single solid line for the left/right borders.

```
border-style: double solid;
```

All of the border styles discussed above can be combined into the following property that formats the width, style, and color of all of the borders

```
border: width style color;
```

### TRY IT

You can explore the CSS border style using the `demo_border.html` file in the `html04` ► demo folder.

where *width* is the thickness of the border, *style* is the style of the border, and *color* is the border color. The following style rule inserts a 2-pixel-wide solid blue border around every side of each h1 heading in the document:

```
h1 {border: 2px solid blue;}
```

To modify the width, style, and color of a single border, use the property

```
border-side: width style color;
```

where *side* is either *top*, *right*, *bottom*, or *left*.

### REFERENCE

#### Adding a Border

- To add a border around every side of an element, use the CSS property

```
border: width style color;
```

where *width* is the width of the border, *style* is the design style, and *color* is the border color.

- To apply a border to a specific side, use

```
border-side: width style color;
```

where *side* is *top*, *right*, *bottom*, or *left* for the top, right, bottom, and left borders.

- To set the width, style, or color of a specific side, use the properties

```
border-side-width: width;
```

```
border-side-style: style;
```

```
border-side-color: color;
```

Kevin wants the page body to stand out better against the tiled images used as the background for the browser window. He suggests you add solid borders to the left and right edges of the page body and that you add a double border around the `aside` element containing links to other Komatsu family pages.

#### To add borders to the page elements:

1. Return to the `tb_visual1.css` file in your editor and go to the Page Body Styles section.

2. Add the following style rule for the page body:

```
body {
    border-left: 1px solid rgb(51, 51, 51);
    border-right: 1px solid rgb(51, 51, 51);
}
```

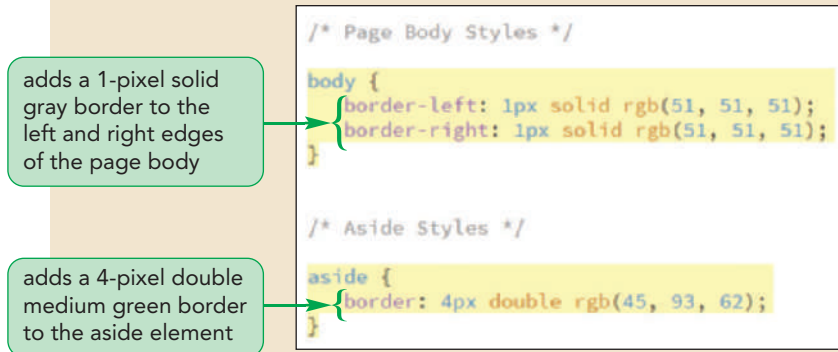
3. Go to the Aside Styles section and add the following style rule for the `aside` element:

```
aside {
    border: 4px double rgb(45, 93, 62);
}
```



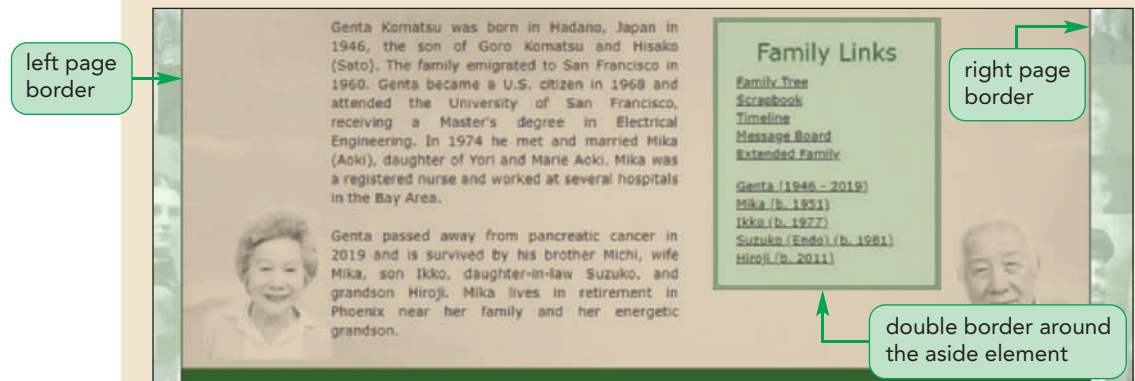
Figure 4–13 highlights the style rules that create borders for the page body and `aside` element.

**Figure 4–13** Adding borders to the page body and aside element



4. Save your changes to the file and then reload `tb_komatsu.html` in your browser. Figure 4–14 shows the appearance of the page with the newly added borders. Note that the background color and other styles associated with the `aside` element are in the `tb_styles1.css` file.

**Figure 4–14** Page design with borders



Source: Wikimedia Commons; © imtmphoto/Shutterstock.com

Kevin is concerned that the design of the page is too boxy and he wants you to soften the design by adding curves to some of the page elements. You can create this effect using rounded corners.

## Creating Rounded Corners

To round off any of the four corners of a border, apply the following `border-radius` property:

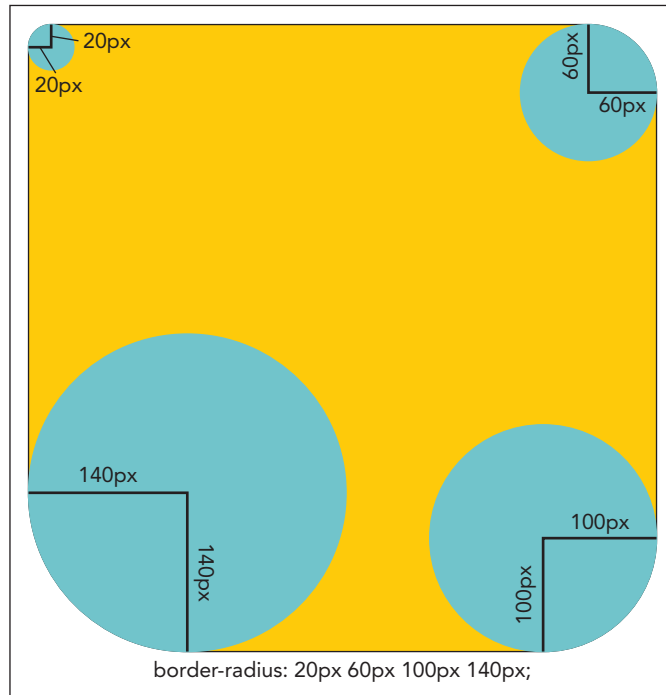
```
border-radius: top-left top-right bottom-right bottom-left;
```

where *top-left*, *top-right*, *bottom-right*, and *bottom-left* are the radii of the individual corners. The radii are equal to the radii of hypothetical circles placed at the corners of the box with the arcs of the circles defining the rounded corners (see Figure 4–15).



Figure 4–15

## Setting rounded corners based on corner radii

**TRY IT**

You can explore the CSS `border-radius` style using the `demo_radius.html` file in the `html04` ► `demo` folder.

If you enter only one radius value, it is applied to all four corners; if you enter two values, the first is applied to the top-left and bottom-right corners, and the second is applied to the top-right and bottom-left corners. If you specify three radii, they are applied to the top-left, top-right/bottom-left, and bottom-right corners, in that order. For example, the following style rule creates rounded corners for the `aside` element in which the radii of the top-left and bottom-right corners is 50 pixels and the radii of the top-right and bottom-left corners is 20 pixels.

```
aside {border-radius: 50px 20px;}
```

To set the curvature for only one corner, use the property:

```
border-corner-radius: radius;
```

where *corner* is either `top-left`, `top-right`, `bottom-right`, or `bottom-left`.

**REFERENCE****Creating a Rounded Corner**

- To create rounded corners for an element border, use

```
border-radius: top-left top-right bottom-right bottom-left;
```

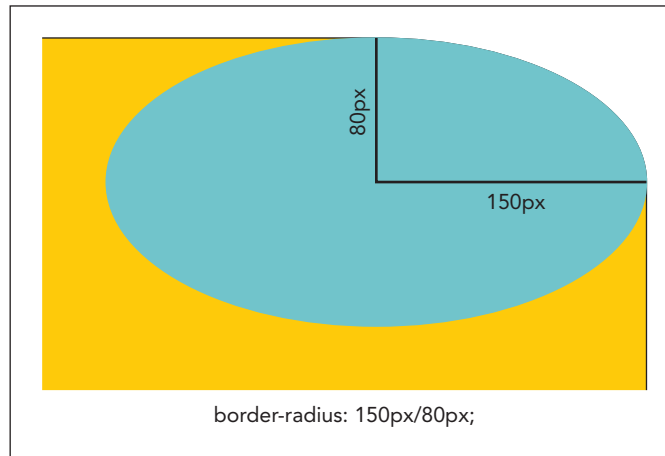
where *radius* is the radius of the rounded corner in one of the CSS units of measurement and *top-left*, *top-right*, *bottom-right*, and *bottom-left* are the radii of the individual corners.

The corners do not need to be circular. Elongated or elliptical corners are created by specifying the ratio of the horizontal radius to the vertical radius using the style:

```
border-radius: horizontal/vertical;
```

where *horizontal* is the horizontal radius of the corner and *vertical* is the vertical radius of the same corner (see Figure 4–16).

**Figure 4–16** Creating an elongated corner



Thus, the following style rule creates elongated corners in which the ratio of the horizontal to vertical radius is 50 pixels to 20 pixels.

```
border-radius: 50px/20px;
```

#### TIP

To create a circular border, use a square element with an equal width and height and the corner radii set to 50%.

Note that using percentages for the radius value can result in elongated corners if the element is not perfectly square. The following style rule sets the horizontal radius to 15% of element width and 15% of the element height. If the element is twice as wide as it is high for example, the corners will not be rounded but elongated.

```
border-radius: 15%;
```

When applied to a single corner, the format to create an elongated corner is slightly different. You remove the slash between the horizontal and vertical values and use the following syntax:

```
border-corner-radius: horizontal vertical;
```

For example, the following style creates an elongated bottom-left corner with a horizontal radius of 50 pixels and a vertical radius of 20 pixels.

```
border-bottom-left-radius: 50px 20px;
```

#### TRY IT

You can explore how to create elliptical corners using the `demo_ellipse.html` file in the `html04` demo folder.

Rounded and elongated corners do not clip element content. If the content of the element extends into the corner, it will still be displayed as part of the background. Because this is often unsightly, you should avoid heavily rounded or elongated corners unless you can be sure they will not obscure or distract from the element content.

Add rounded corners with a radius of 30 pixels to the `aside` element.

**To add rounded corners to an element:**

1. Return to the `tb_visual1.css` file in your editor and go to the `Aside Styles` section.
2. Add the following style to the style rule for the `aside` element:

```
border-radius: 30px;
```

Figure 4–17 highlights the style to create the rounded corners for the `aside` border.

Figure 4–17

**Adding rounded corners to the aside element border**

sets the radius at each border corner to 30 pixels

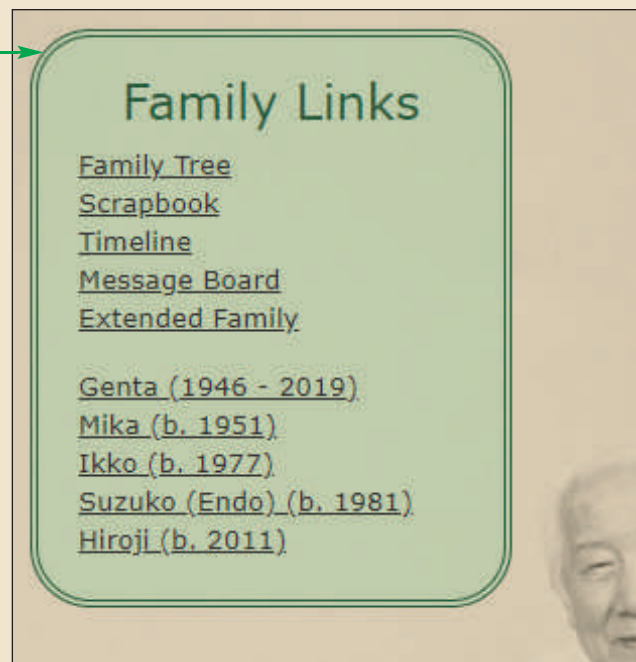
```
aside {  
  border: 4px double rgb(45, 93, 62);  
  border-radius: 30px;  
}
```

3. Save your changes to the file and reload `tb_komatsu.html` in your browser. Figure 4–18 shows the rounded corners for the `aside` element border.

Figure 4–18

**Aside element border with rounded corners**

rounded corner



© imtmphoto/Shutterstock.com

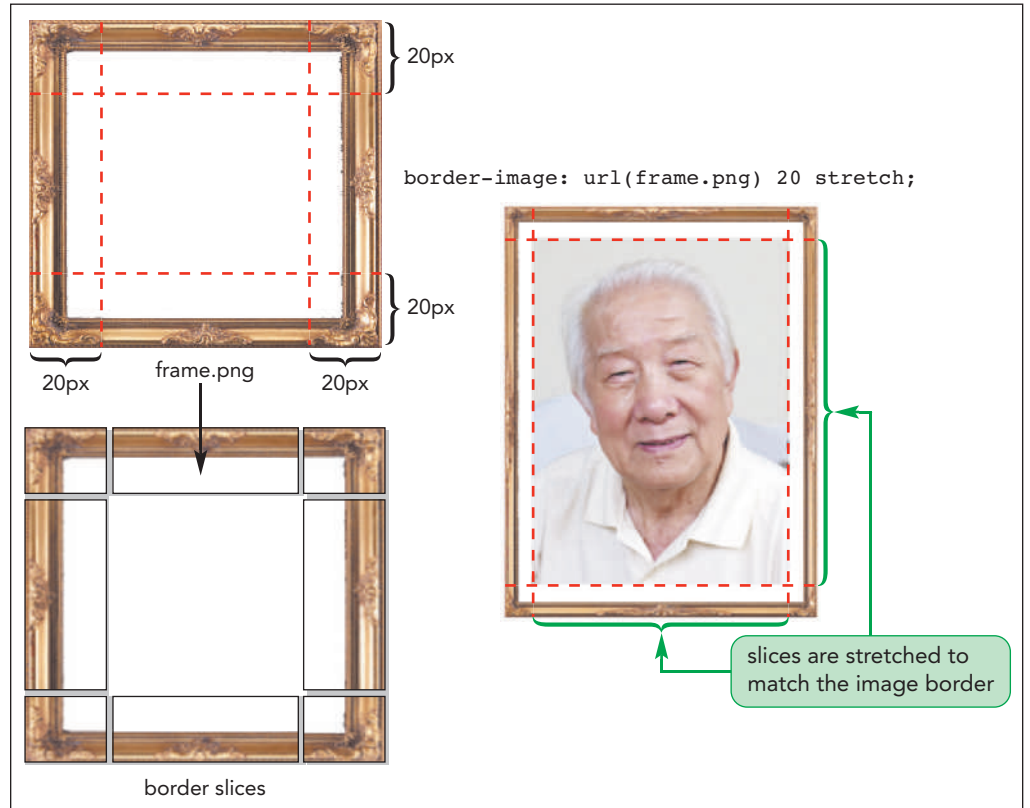
Kevin likes the revision to the border for the `aside` element. He also wants you to add a border to the family portrait on the Komatsu Family page. However, rather than using one of the styles shown in Figure 4–12, Kevin wants you to use a graphic border that makes it appear as if the figure box came from a torn piece of paper. You can create this effect using border images.

## Applying a Border Image

A border image is a border that is based on a graphic image. The graphic image is sliced into nine sections representing the four corners, the four sides, and the interior piece. The interior piece is discarded because that is where the content of the object will appear; the four corners become the corners of the border and the four sides are either stretched or tiled to fill in the border's top, right, bottom, and left sides. Figure 4–19 shows an example of an image file, `frame.png`, sliced into nine sections to create a border image.

Figure 4–19

### Slicing a graphic image to create a border



© imtmphoto/Shutterstock.com

To apply a border image, use the following property

```
border-image: url(url) slice repeat fill;
```

where `url` is the source of the graphic image, `slice` is the width or height of the slices used to create the sides and corners, `repeat` indicates whether the side slices should be stretched or tiled to cover the border's four sides, and `fill` is an optional attribute that fills the image background with the graphic image file. The `repeat` option supports the following values:

- `stretch`: The slices are stretched to fill each side.
- `repeat`: The slices are tiled to fill each side.
- `round`: The slices are tiled to fill each side; if they don't fill the sides with an integer number of tiles, the slices are rescaled until they do.
- `space`: The slices are tiled to fill each side; if they don't fill the sides with an integer number of tiles, extra space is distributed around the tiles.

For example, the following style cuts 10-pixel-wide slices from the `frame.png` image file with the four side slices stretched to cover the length of the four sides of the object's border:

```
border-image: url(frame.png) 10 stretch;
```

The size of the slices is measured either in pixels or as a percentage of the image file width and height. A quirk of this property is that you should *not* specify the pixel unit if you want the slices measured in pixels but you must include the % symbol when slices are measured in percentages.

You can create slices of different widths or heights by entering the size values in a space-separated list. For instance, the following style slices the graphic image 5 pixels on the top, 10 pixels on the right, 15 pixels on the bottom, and 25 pixels on the left:

```
border-image: url(frame.png) 5 10 15 25 stretch;
```

### TRY IT

You can explore how to create a border image using the `demo_frame.html` file in the `html04` demo folder.

The slice sizes follow the same top/right/bottom/left syntax used with all of the CSS border styles. Thus, the following style slices 5% from the top and bottom sides of the graphic image, and 10% from the left and right sides:

```
border-image: url(frame.png) 5% 10% stretch;
```

You can also apply different repeat values to different sides of the border. For example, the following style stretches the border slices on the top and bottom but tiles the left and right slices:

```
border-image: url(frame.png) 10 stretch repeat;
```

### REFERENCE

#### Creating a Graphic Border

- To create a border based on a graphic image, use

```
border-image: url(url) slice repeat fill;
```

where *url* is the source of the border image file, *slice* is the size of the border image cut off to create the borders, *repeat* indicates whether the side borders should be either stretched or tiled to cover the object's four sides, and *fill* is an optional attribute that fills the image background with the graphic image file.

The torn paper image that Kevin wants to use is based on the graphic image file `tp_border.png` file. Use the `border-image` property to add a border image around the figure box on the Komatsu Family page, tiling the border slices to fill the sides. Note that in order for the border image to appear you must include values for the `border-width` and `border-style` properties.

#### To create a graphic border:

1. Return to the `tb_visual1.css` file in your editor and scroll to the Figure Box Styles at the top of the file.
2. Add the following style to the style rule for the figure box:

```
border-style: solid;
border-width: 25px;
border-image: url(tb_border.png) 50 repeat;
```

Figure 4–20 displays the styles used to create the graphic border.

Figure 4–20

## Adding a border image

border width and style values are required for the border image

slices 50 pixels from each side of the border image

```
figure {
  border-style: solid;
  border-width: 25px;
  border-image: url(tb_border.png) 50 repeat;
  margin: 20px auto 0px;
  width: 80%;
}
```

uses the tb\_border.png file for the graphical border

tiles the side slices to fill the border sides

3. Save your changes and reload tb\_komatsu.html in your browser. Figure 4–21 shows the appearance of the border image.

Figure 4–21

## Figure box with border image

graphic image slices are tiled to fill the border sides



(L-R): Ikko, Mika, Hiroji, Genta, Suzuko

border image created from the tb\_border.png file

© imtmphoto/Shutterstock.com

Kevin appreciates the effect you created, making it appear as if the family portrait was torn from an album and laid on top of the web page.



### Problem Solving: Graphic Design and Legacy Browsers

Adding snazzy graphics to your page can be fun, but you must keep in mind that the fundamental test of your design is not how cool it looks but how usable it is. Any design you create needs to be compatible across several browser versions if you want to reach the widest user base. To support older browsers, your style sheet should use progressive enhancement in which the older properties are listed first, followed by browser extensions, and then by the most current CSS properties. As each property supersedes the previous properties, the browser will end up using the most current property that it supports.

For example, the following style rule starts with a basic 5-pixel blue border that will be recognized by every browser. It is followed by browser extensions for Opera, Mozilla, and WebKit to support older browsers that predate adoption of the CSS `border-image` property. Finally, the style list ends with the CSS `border-image` property, recognized by every current browser. In this way, every browser that opens the page will show some type of border.

```
border: 5px solid blue;
-o-border-image: url(paper.png) 30 repeat;
-moz-border-image: url(paper.png) 30 repeat;
-webkit-border-image: url(paper.png) 30 repeat;
border-image: url(paper.png) 30 repeat;
```

Be aware, however, that the syntax for an extension may not match the syntax for the final CSS specification. For example, the following list of styles creates a rounded top-right corner that is compatible across a wide range of browser versions:

```
-moz-border-radius-top-right: 15px;
-webkit-border-top-right-radius: 15px;
border-top-right-radius: 15px;
```

Note that the syntax for the Mozilla extension does not match the syntax for the WebKit extension or for the final CSS specification. As always, you need to do your homework to learn exactly how different browser versions handle these CSS design styles.

In the next session, you'll continue to work with the CSS graphic styles to add three-dimensional effects through the use of drop shadows and color gradients. If you want to take a break, you can close your open files and documents now.



### Session 4.1 Quick Check

1. The element to create a figure caption is:
  - a. caption
  - b. figurecap
  - c. figcaption
  - d. alt
2. Lines and curves based on mathematical functions comprise a:
  - a. bitmap image
  - b. png image
  - c. figure image
  - d. vector image
3. To tile a background image in the horizontal direction only, use:
  - a. repeat-x
  - b. repeat-y
  - c. repeat-horizontal
  - d. repeat-h
4. To display the background image only within a content box, apply the style:
  - a. background-origin: content-box;
  - b. background: content-box;
  - c. background-display: content-box;
  - d. background-clip: content-box;
5. To create a 5-pixel wide brown border with a dotted line, apply the style:
  - a. border-style: 5px brown dotted;
  - b. border-type: brown 5px dotted;
  - c. border-outline: dotted brown 5px;
  - d. border: 5px brown dotted;
6. To use rounded corners with a radius of 15 pixels in a border, apply the style:
  - a. border-width: 15px;
  - b. border-radius: 15px;
  - c. border-arc: 15px;
  - d. corner-radius: 15px;
7. To create an elongated corner with a horizontal radius of 10 pixels and a vertical radius of 5 pixels, use:
  - a. border-radius: 10px 5px;
  - b. border-radius: 5px 10px;
  - c. border-radius: 10px/5px;
  - d. border-radius: 5px/10px;
8. To create a border image using the border.png file with a slice size of 30 pixels and the slices stretched along the borders, use:
  - a. border-image: url(border.png) 30 stretch;
  - b. border: url(border.png) 30 stretch;
  - c. border-img: url(border.png) 30 stretch;
  - d. border-slice: url(border.png) 30 stretch;

# Session 4.2 Visual Overview:

The diagram illustrates various CSS properties and their usage in code snippets, with callouts explaining their functions:

- box-shadow:** The `box-shadow` property adds a drop shadow to a block element. The color value sets the shadow color. The distance values set the shadow offsets. This value sets the shadow blur.
- inset:** The `inset` keyword places the shadow inside the element.
- text-shadow:** The `text-shadow` property adds a drop shadow to a text string. This value sets the shadow size. These values set the shadow offsets.
- radial-gradient:** The `radial-gradient` function creates a color gradient proceeding outward from a central point. These color values set the shadow color. This value sets the shadow blur.
- linear-gradient:** The `linear-gradient` function creates a color gradient proceeding along a straight line. This value sets the opacity of the figure to 55%. The `color-stop` defines the extent of a color within a gradient.
- opacity:** The `opacity` property makes an object semi-transparent.

```

/* Shadow Styles */
body {
  box-shadow: rgb(51, 51, 51) 15px 8px 25px,
             rgb(51, 51, 51) -15px 8px 25px;
}

article {
  box-shadow: inset rgb(71, 71, 71) -10px -10px 25px,
             inset rgb(71, 71, 71) 10px 10px 25px;
}

aside {
  box-shadow: rgba(51, 91, 51, 0.4) 8px 8px 20px 10px;
}

article header h1 {
  text-shadow: rgb(181, 211, 181) 2px 2px 1px,
             rgba(21, 21, 21, 0.66) 5px 5px 25px;
}

/* Gradient Styles */
aside {
  background: radial-gradient(white, rgb(151, 222, 151), rgb(81, 125, 81));
}

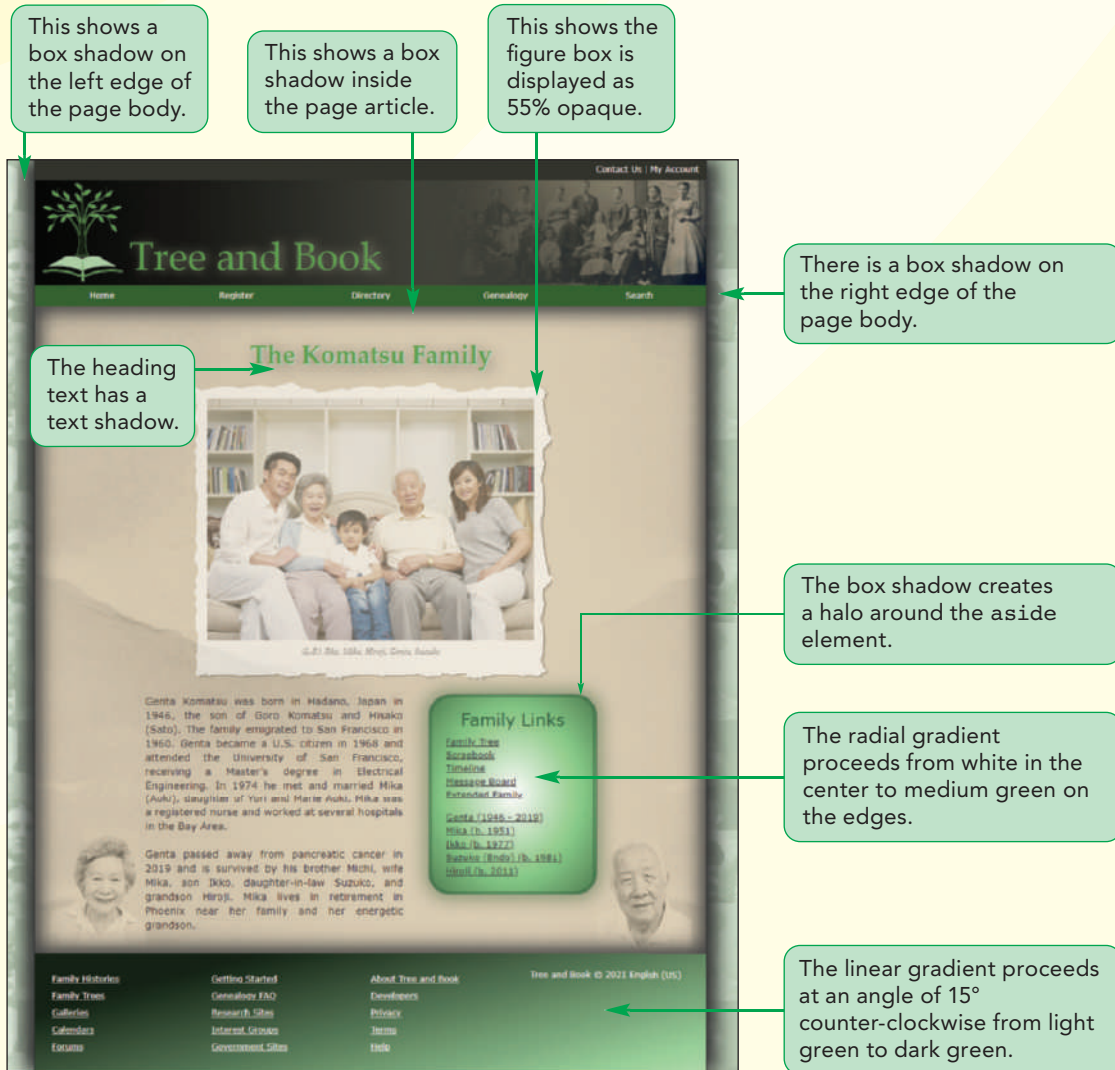
footer {
  background: linear-gradient(345deg, rgb(172, 232, 172), rgb(21, 35, 21) 80%);
}

/* Opacity Styles */
body > article > header > figure {
  opacity: 0.55;
}

```

Logo Design Studio Pro; Source: wiki Media;  
© imtmphoto/Shutterstock.com

# Shadows and Gradients



Source: Design Studio Pro; Source: Wikimedia Commons; imtphoto/Shutterstock.com

## Creating Drop Shadows

In this session, you will examine some design styles that create 3D effects, making the page content appear to jump out of the browser window. The first styles you'll explore are used to create drop shadows around text strings and element boxes.

### Creating a Text Shadow

To give the text on your page visual impact, you can use CSS to add a shadow using the following `text-shadow` property

```
text-shadow: color offsetX offsetY blur;
```

where *color* is the shadow color, *offsetX* and *offsetY* are the distances of the shadow from the text in the horizontal and vertical directions, and *blur* defines the amount by which the shadow spreads out, creating a blurred effect. The shadow offset values are expressed so that positive values push the shadow to the right and down while negative values move the shadow to the left and up. The default *blur* value is 0, creating a shadow with distinct hard edges; as the blur value increases, the edge of the shadow becomes less distinct and blends more in the text background.

The following style creates a red text shadow that is 10 pixels to the right and 5 pixels down from the text with blur of 8 pixels:

```
text-shadow: red 10px 5px 8px;
```

Multiple shadows can be added to text by including each shadow definition in the following comma-separated list.

```
text-shadow: shadow1, shadow2, shadow3, ...;
```

where *shadow1*, *shadow2*, *shadow3*, and so on are shadows applied to the text with the first shadow listed displayed on top of subsequent shadows when they overlap. The following style rule creates two shadows with the first red shadow placed 10 pixels to the left and 5 pixels up from the text and the second gray shadow is placed 3 pixels to the right and 4 pixels down from the text. Both shadows have a blur of 6 pixels:

```
text-shadow: red -10px -5px 6px,  
            gray 3px 4px 6px;
```

#### TRY IT

You can explore the `text-shadow` style and creating multiple text shadows by using the `demo_text.html` file from the `html04 ▶ demo` folder.

Figure 4–22 shows examples of how the `text-shadow` style can be used to achieve a variety of text designs involving single and multiple shadows.

Figure 4–22

## Examples of text shadows



### Creating a Text Shadow

**REFERENCE**

- To add a shadow to a text string, use the property

```
text-shadow: color offsetX offsetY blur;
```

where *color* is the shadow color, *offsetX* and *offsetY* are the distances of the shadow from the text in the horizontal and vertical directions, and *blur* defines the amount by which the shadow is stretched.

Kevin wants you to add two text shadows to the h1 heading *The Komatsu Family*. The first text shadow will be a light-green highlight with hard edges and the second shadow will be semi-transparent gray and blurred.

#### To add a text shadow:

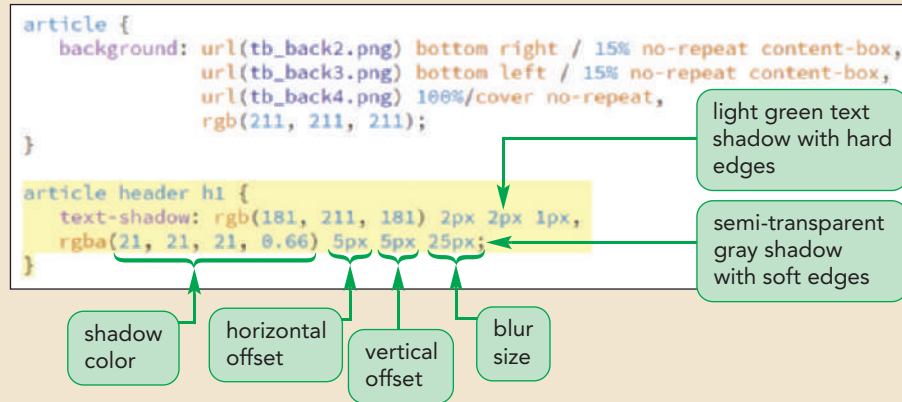
- If you took a break after the previous session, reopen or return to the `tb_visual1.css` file in your editor and scroll to the Article Styles section.
- Add the following style for the h1 heading in the article header:

```
article header h1 {
  text-shadow: rgb(181, 211, 181) 2px 2px 1px,
              rgba(21, 21, 21, 0.66) 5px 5px 25px;
}
```

Figure 4–23 highlights the style to add text shadows to the h1 heading.

Figure 4–23

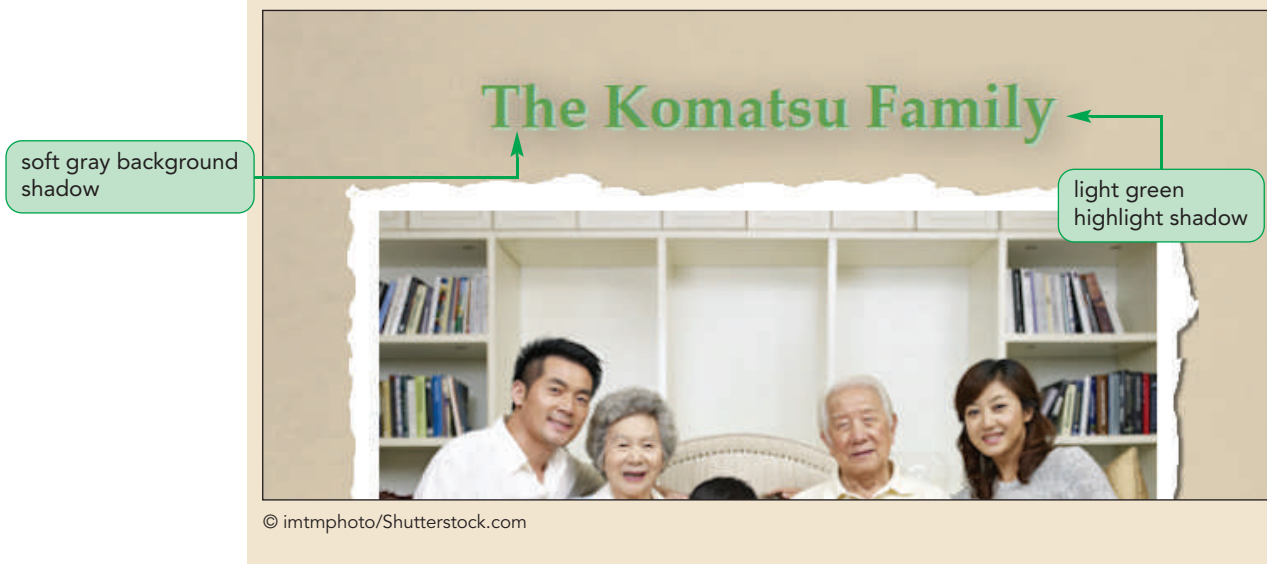
## Adding text shadows



3. Save your changes and reload `tb_komatsu.html` in your browser. Figure 4–24 shows the shadow effect added to the h1 heading.

Figure 4–24

## Article heading with text shadows



Kevin likes the shadow effect and the use of the light green shadow, which appears to give a highlight to the heading text. Next, he wants you to add shadows to other page objects.

## Creating a Box Shadow

Shadows can be added to any block element in the web page by using the `box-shadow` property

```
box-shadow: color offsetX offsetY blur;
```

where `color`, `offsetX`, `offsetY`, and `blur` have the same meanings for box shadows as they do for text shadows. As with text shadows, you can add multiple shadows by including them in the following comma-separated list

```
box-shadow: shadow1, shadow2 ...;
```

where once again the first shadow listed is displayed on top of subsequent shadows.

In the last session, you used left and right borders to set off the page body from the browser window background. Kevin would like you to increase this visual distinction by adding drop shadows to the left and right sides of the page body.

### TIP

If no shadow color is provided, the browser uses black as the default color.

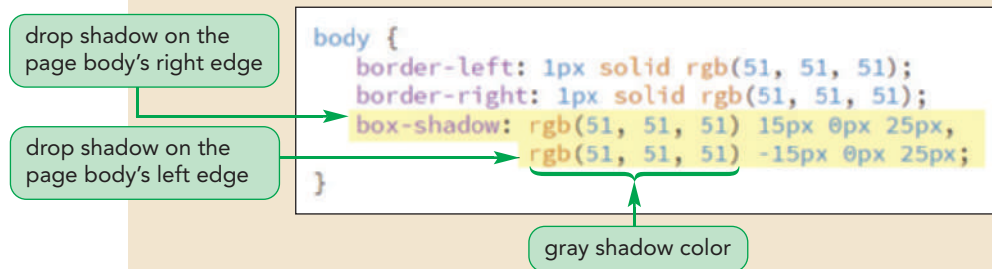
**To add a box shadow:**

1. Return to the `tb_visual1.css` file in your editor and go to the Page Body Styles section.
2. Within the style rule for the `body` element, insert the following styles:

```
box-shadow: rgb(51, 51, 51) 15px 0px 25px,
           rgb(51, 51, 51) -15px 0px 25px;
```

Figure 4–25 highlights the style to add box shadows to the page body.

**Figure 4–25** Adding box shadows



3. Save your changes and reload `tb_komatsu.html` in your browser. Figure 4–26 shows the drop shadows added to the page body.

**Figure 4–26** Page body with drop shadows



Source: Design Studio Pro; Source: Wikimedia Commons; © imtmphoto/Shutterstock.com

Box shadows can be placed inside the element as well as outside. By adding an interior shadow you can create the illusion of a beveled edge in which the object appears to rise out of its background. To create an interior shadow, add the `inset` keyword to the `box-shadow` property

```
box-shadow: inset color offsetX offsetY blur;
```

**TRY IT**

Explore multiple box shadows with the `demo_box.html` file from the `html04 ▶ demo` folder.

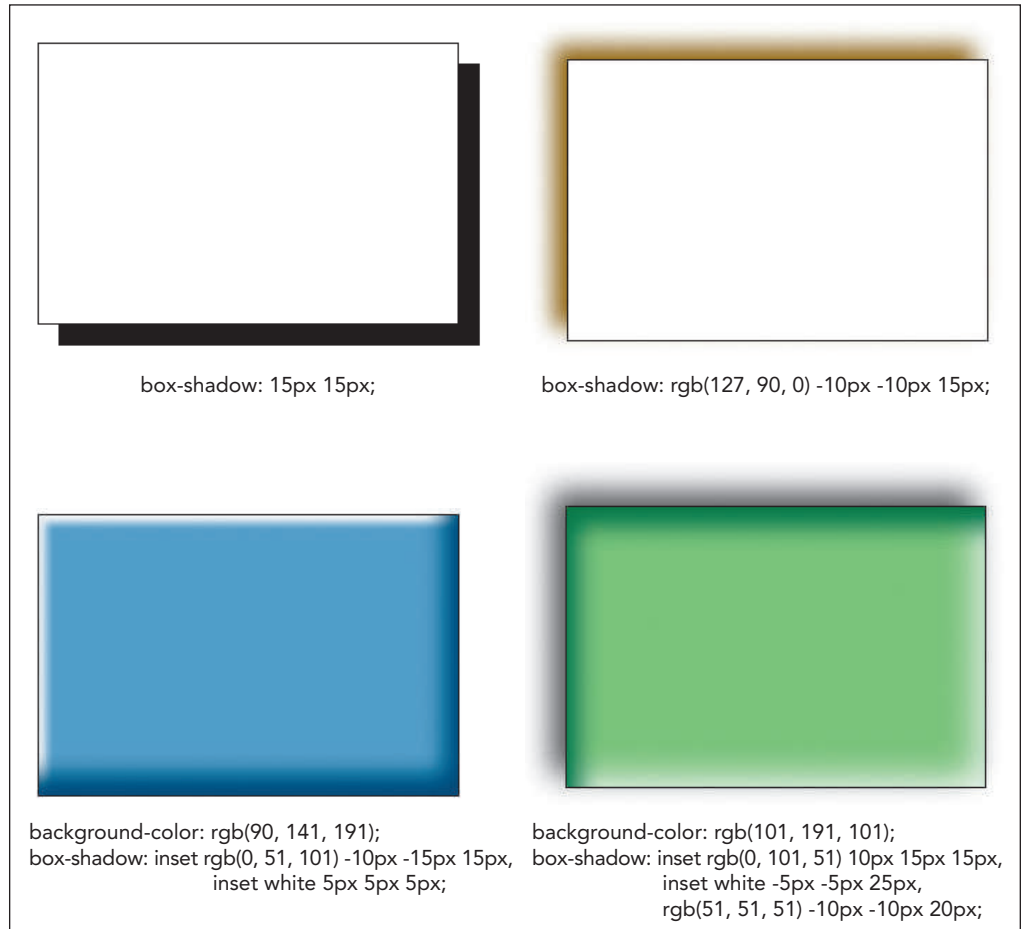
where the meanings of the `offsetX` and `offsetY` values are switched when applied to interior shadowing so that positive `offsetX` and `offsetY` values move the shadow to the left and up within the box, while negative `offsetX` and `offsetY` values move the shadow to the right and down.



An object can contain a mixture of exterior and interior shadows. Figure 4–27 shows examples of box shadows, including one example that mixes both interior and exterior shadows.

Figure 4–27

## Examples of box shadows



Kevin suggests that you add inset shadows to the `article` element, placing medium gray shadows within the article to make it appear raised up from the surrounding page content.

**To add inset shadows:**

1. Return to the `tb_visual1.css` file in your editor and go to the Article Styles section.
2. Within the style rule for the `article` element, insert the following `box-shadow` style:

```
box-shadow: inset rgb(71, 71, 71) -10px -10px 25px,  
inset rgb(71, 71, 71) 10px 10px 25px;
```

Figure 4–28 highlights the newly added code for the inset box shadow.

Positive and negative offset values for interior shadows have the opposite meaning from positive and negative offset values for exterior shadows.

Figure 4–28

## Adding an inset shadow

places a medium-gray shadow in the lower-right interior corner

inset keyword places shadow inside the object

```
article {
  background: url(tb_back2.png) bottom right / 15% no-repeat content-box,
             url(tb_back3.png) bottom left / 15% no-repeat content-box,
             url(tb_back4.png) 100%/cover no-repeat,
             rgb(211, 211, 211);
  box-shadow: inset rgb(71, 71, 71) -10px -10px 25px,
             inset rgb(71, 71, 71) 10px 10px 25px;
}
```

places a medium-gray shadow in the upper-left interior corner

3. Save your changes and reload `tb_komatsu.html` in your browser. The inset shadow for the page body element is shown in Figure 4–29.

Figure 4–29

## Page article with interior shadowing

interior shadow placed on the left and up based on positive offset values



interior shadow placed on the right and down based on negative offset values

Source: Wikimedia Commons; © imtmphoto/Shutterstock.com

By default, a box shadow has the same size and dimensions as its page object offset in the horizontal and vertical direction. To change the shadow size, add the `spread` parameter to the `box-shadow` property, specifying the size of the shadow relative to the size of the page object. A positive value increases the size of the shadow, while a negative value decreases it. For example, the following style creates a gray shadow that is offset from the page object by 5 pixels in both the vertical and horizontal direction

with no blurring but with a shadow that is 15 pixels larger in the horizontal and vertical directions than the object:

```
box-shadow: gray 5px 5px 0px 15px;
```

On the other hand, the following style creates a shadow that is 15 pixels smaller than the page object:

```
box-shadow: gray 5px 5px 0px -15px;
```

### Creating a Box Shadow

#### REFERENCE

- To add a shadow to a block element, use

```
box-shadow: color offsetX offsetY blur spread;
```

where *color* is the shadow color, *offsetX* and *offsetY* are the distances of the shadow from the element in the horizontal and vertical directions, *blur* defines the amount by which the shadow is stretched and *spread* sets the size of the shadow relative to the size of the block element. If no *spread* is specified, the shadow has the same size as the block element.

- To create an interior shadow, include the `inset` keyword

```
box-shadow: inset color offsetX offsetY blur spread;
```

- To create multiple shadows place them in a comma-separated list:

```
box-shadow: shadow1, shadow2, ...;
```

where *shadow1*, *shadow2*, and so on are definitions for individual shadows with the first shadows listed displayed on top of subsequent shadows.

One application of the *spread* parameter is to create a visual effect in which the object appears to be surrounded by a halo. This is achieved by setting the shadow offsets to 0 pixels while making the shadow larger than the page object itself. Kevin suggests that you use this technique to add a green halo to the `aside` element.

#### To increase the shadow size:

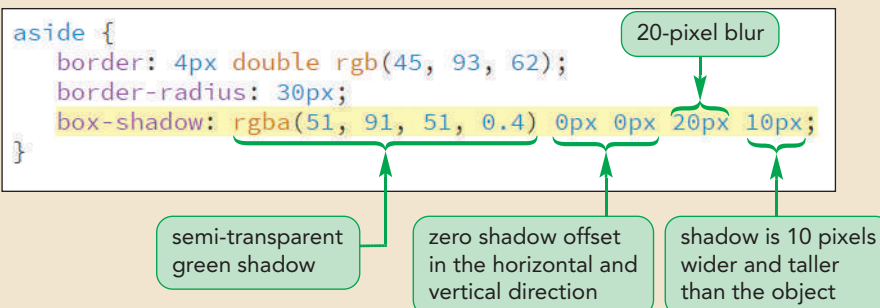
- Return to the `tb_visual1.css` file in your editor and go to the `Aside Styles` section.
- Within the style rule for the `aside` element, insert the following style:

```
box-shadow: rgba(51, 91, 51, 0.4) 0px 0px 20px 10px;
```

Figure 4–30 highlights the style to add a halo to the `aside` element.

Figure 4–30

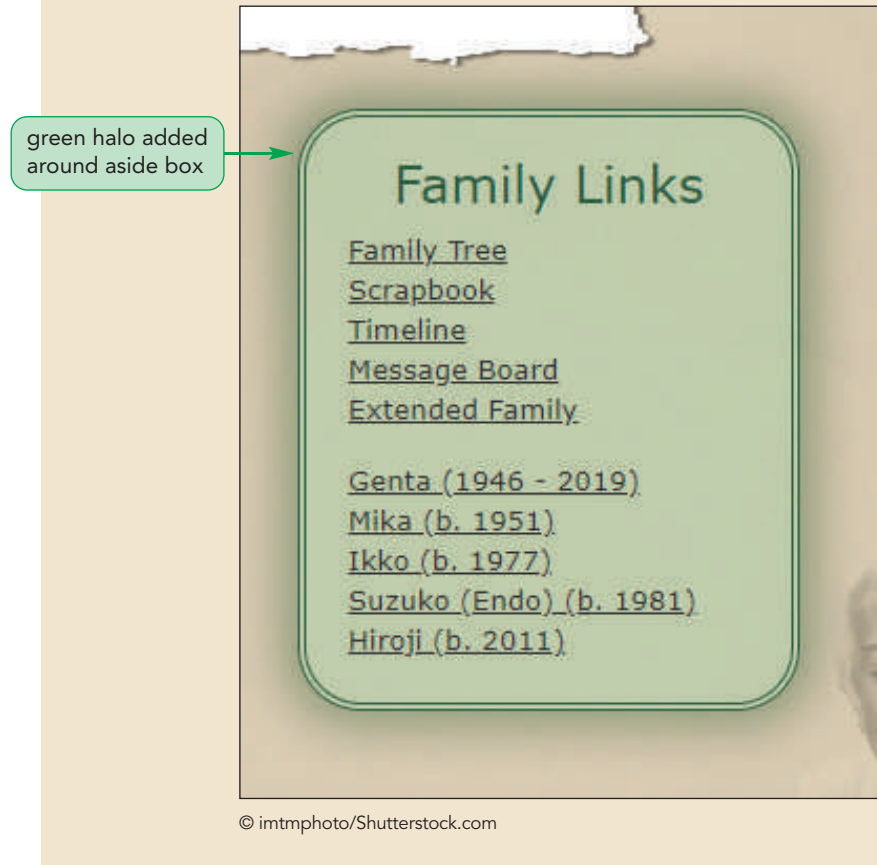
#### Creating a spreading shadow



3. Save your changes and reload `tb_komatsu.html` in your browser. Figure 4–31 shows the revised appearance of the `aside` element with the glowing green shadow.

Figure 4–31

## Aside element with glowing effect



## INSIGHT

### Creating a Reflection

WebKit, the rendering engine for Safari and Google Chrome, includes support for adding reflections to page objects through the following property

```
-webkit-box-reflect: direction offset mask-box-image;
```

where *direction* is the placement of the reflection using the keywords above, *below*, *left*, or *right*; *offset* is the distance of the reflection from the edge of the element box, and *mask-box-image* is an image that can be used to overlay the reflection. For example, the following style rule creates a reflection that is 10 pixels below the inline image:

```
img {
  -webkit-box-reflect: below 10px;
}
```

There is no equivalent `reflect` property in the official W3C CSS specifications. Before using the `reflect` property, you should view the current browser support for the `-webkit-box-reflect` property at [caniuse.com](http://caniuse.com).

## Applying a Color Gradient

So far you have worked with backgrounds consisting of a single color, though that color can be augmented through the use of drop shadows. Another way to modify the background color is through a **color gradient** in which one color gradually blends into another color or fades away if transparent colors are used. CSS supports linear gradients and radial gradients.

### Creating a Linear Gradient

A linear gradient is a color gradient in which the background color transitions from a starting color to an ending color along a straight line. Linear gradients are created using the `linear-gradient` function

```
linear-gradient(color1, color2, ...)
```

where `color1`, `color2`, and so on are the colors that blend into one another starting from `color1`, through `color2`, and onto the last color listed. The default direction for a linear color gradient is vertical, starting from the top of the object and moving to the bottom.

#### TIP

When using multiple backgrounds, gradients can be combined with solid colors and background images to create interesting visual effects; one gradient can also be overlaid on top of another.

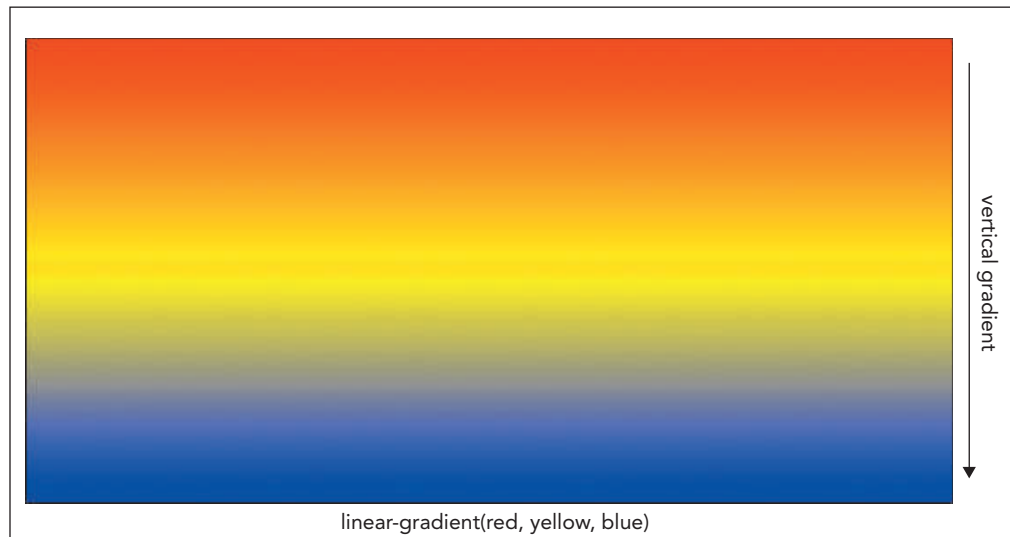
Gradients are treated like background images and thus can be used with any CSS property that accepts an image such as the `background`, `background-image`, and `list-style-image` properties. For example, to create a linear gradient as a background for the page body, you could apply the following style rule:

```
body {  
    background: linear-gradient(red, yellow, blue);  
}
```

Figure 4–32 shows the appearance of this vertical gradient as the background color transitions gradually from red down to yellow and then from yellow down to blue.

Figure 4–32

Linear gradient with three colors



To change from the default vertical direction, you add a `direction` value to the `linear-gradient` function

```
linear-gradient(direction, color1, color2, ...)
```

where `direction` is the direction of the gradient using keywords or angles. Direction keywords are written in the form `to position` where `position` is either a side of the

object or a corner. For example the following linear gradient moves in a straight line to the left edge of the object blending from red to yellow to blue:

```
background: linear-gradient(to left, red, yellow, blue);
```

To move toward the corner, include both corner edges. The following style moves the gradient in the direction of the object's bottom right corner:

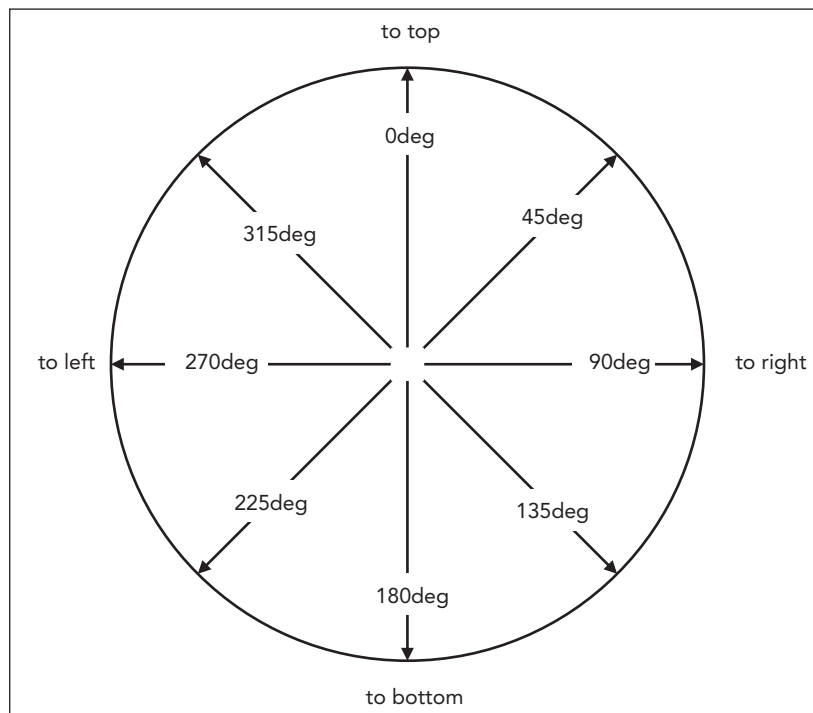
```
background: linear-gradient(to bottom right, red, yellow, blue);
```

### TIP

For square objects, a direction of 45deg is equivalent to a direction of to right top.

To move in a direction other than a side or corner, you can express the direction using an angle value. Angles are measured in degrees with 0deg equal to to top, 90deg equal to to right, 180deg equal to to bottom, and 270deg equal to to left (see Figure 4–33.)

**Figure 4–33** Linear gradient directions



For example, the following gradient points at a 60 degree angle:

```
background: linear-gradient(60deg, red, yellow, blue);
```

Figure 4–34 shows other examples of linear gradients moving in different directions using both syntaxes.

## INSIGHT

### Transparency and Gradients

Interesting gradient effects can be achieved using transparent colors so that the background color gradually fades away as it moves in the direction of the gradient. For example, the following style creates a linear gradient that gradually fades away from its initial solid red color:

```
linear-gradient(rgba(255, 0, 0, 1), rgba(255, 0, 0, 0))
```

Note that since the final color is completely transparent it will adopt the background color of the parent element.

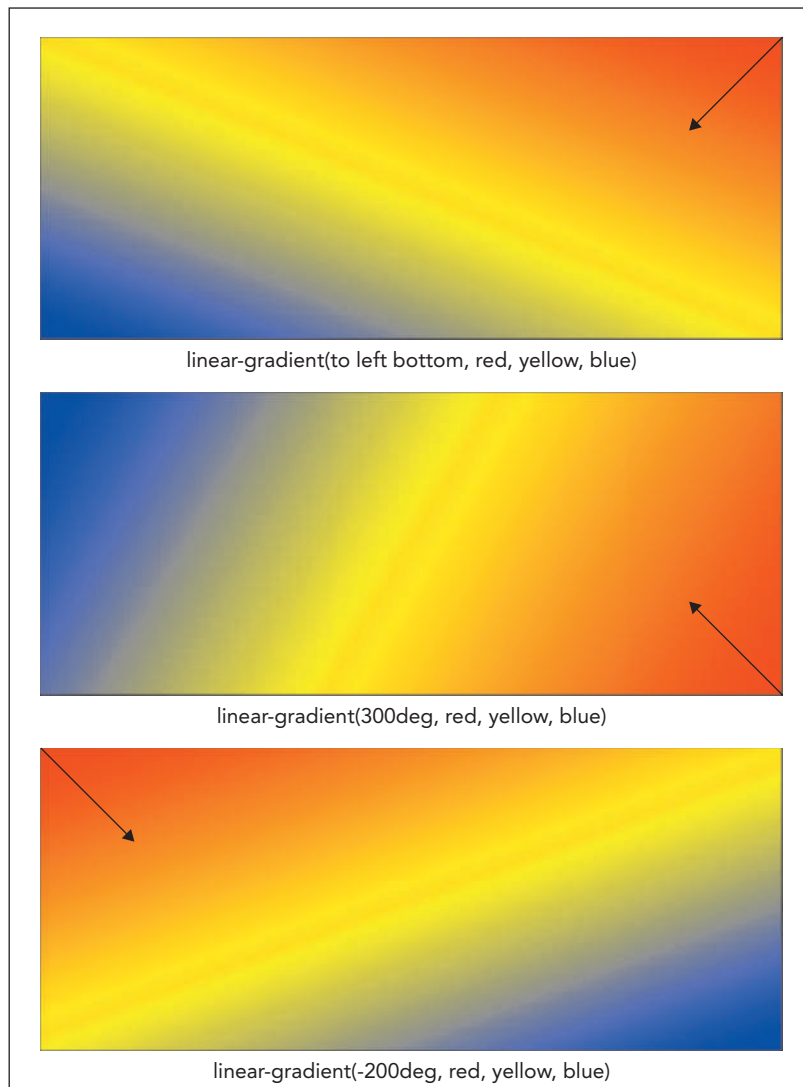
You can also use gradients to create background images that appear to fade by using multiple backgrounds in which the gradient appears on top of an image. For example, the following background style creates a fading background using the back.png image file:

```
background: linear-gradient(rgb(255, 255, 255, 0), rgb(255, 255, 255, 1)), url(back.png);
```

When rendered by the browser, the background image will start as solid but gradually fade to white as the linear gradient proceeds through the element background.

Figure 4–34

Directions of linear gradients





Note that the degree values can be negative in which case the direction is pointed counter-clockwise around the circle shown in Figure 4–33. A negative angle of  $-45\text{deg}$ , for example, would be equivalent to a positive angle of  $315\text{deg}$ , an angle of  $-200\text{deg}$  would be equal to  $160\text{deg}$ , and so forth.

## Gradients and Color Stops

The colors specified in a gradient are evenly distributed so that the following gradient starts with a solid red, solid green appears halfway through the gradient, and finishes with solid blue:

```
background: linear-gradient(red, green, blue);
```

To change how the colors are distributed, you define color stops, which represent the point at which the specified color stops and the transition to the next color begins. The `linear-gradient` function using color stops has the general form

```
linear-gradient(direction, color-stop1, color-stop2, ...)
```

where `color-stop1`, `color-stop2`, and so on are the colors and their stopping positions within the gradient. Stopping positions can be entered using any of the CSS units of measurement. For example, the following gradient starts with solid red up until 50 pixels from the starting point, red blends to solid green stopping at 60 pixels from the starting point and then blends into solid blue 80 pixels from the start. After 80 pixels, the gradient will remain solid blue to the end of the background.

```
linear-gradient(red 50px, green 60px, blue 80px)
```

### TRY IT

You can create your own linear gradients using the `demo_linear.html` file from the `html04` ► `demo` folder.

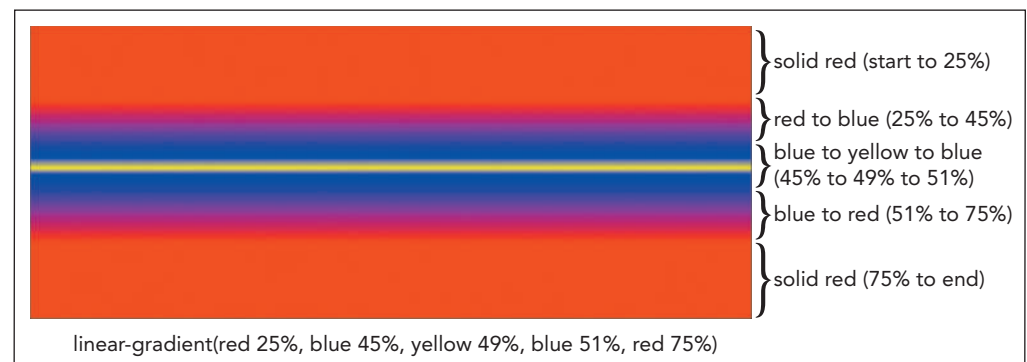
Similarly, the following style rule sets the color stops using percentages with solid red for the first 25% of the background, transitioning to solid green from 25% to 75% of the background, and then transitioning to solid blue from 75% to 95% of the background size. From that point to the end, the background remains solid blue.

```
linear-gradient(red 25%, green 75%, blue 95%)
```

Figure 4–35 shows an example of a linear gradient in which color stops are used to create a narrow strip of yellow within a background of red blended into blue.

Figure 4–35

Linear gradient color stops



Kevin suggests you use a linear gradient that transitions from light green to dark green as the background for the page footer.

### To apply a linear gradient:

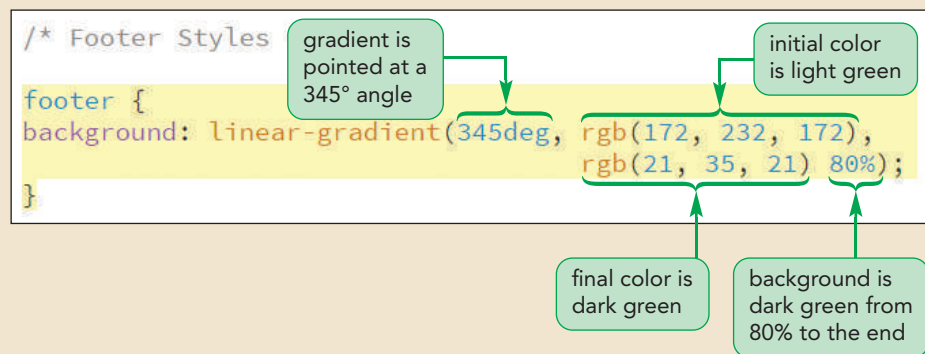
1. Return to the `tb_visual1.css` file in your editor and go to the Footer Styles section.
2. Insert the following style rule for the `footer` element:

```
footer {
    background: linear-gradient(345deg, rgb(172, 232, 172),
                                rgb(21, 35, 21) 80%);
}
```

Figure 4–36 highlights the style to create the linear gradient.

Figure 4–36

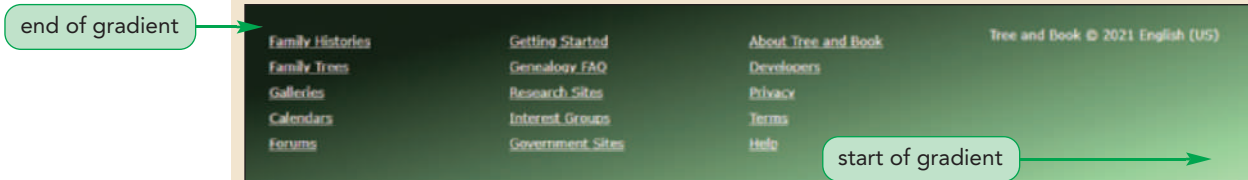
### Applying a linear gradient



3. Save your changes and reload `tb_komatsu.html` in your browser. Figure 4–37 shows the revised appearance of the page footer with a linear gradient.

Figure 4–37

### Page footer with linear gradient background



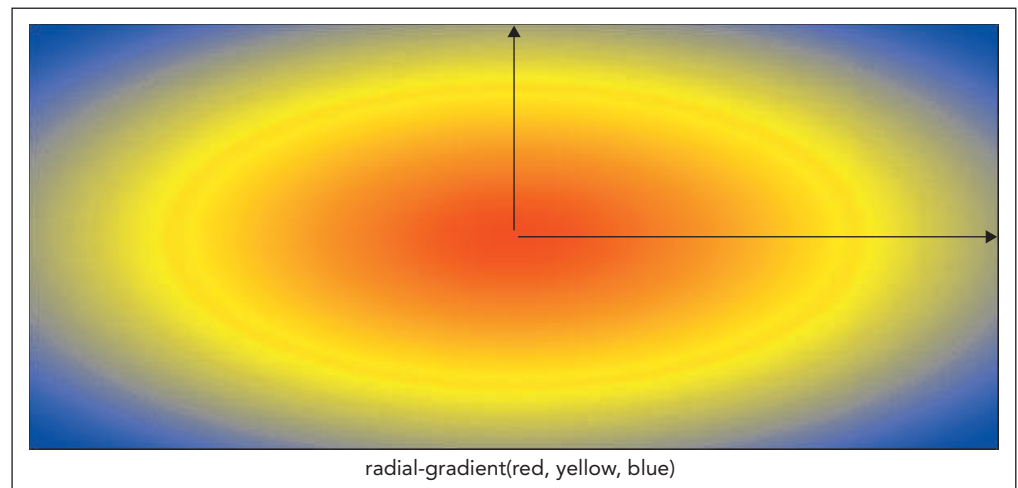
The other color gradient supported in CSS is a radial gradient. You will explore how to create radial gradients now.

## Creating a Radial Gradient

A **radial gradient** is a color gradient that starts from a central point and proceeds outward in a series of concentric circles or ellipses. Figure 4–38 shows an example of a radial gradient consisting of a series of concentric ellipses radiating from a central red color to an ending blue color.

Figure 4–38

## A radial gradient of three colors



Radial gradients are created using the following `radial-gradient` function.

```
radial-gradient(shape size at position, color-stop1,
               color-stop2, ...)
```

The `shape` value defines the shape of the gradient and is either `ellipse` (the default) or `circle`. The `size` value defines the extent of the gradient as it radiates outward and can be expressed with a CSS unit of measure, a percentage of the background's width and height, or with one of the following keywords:

- `farthest-corner` (the default) Gradient extends to the background corner farthest from the gradient's center.
- `farthest-side` Gradient extends to background side farthest from the gradient's center.
- `closest-corner` Gradient extends to the nearest background corner.
- `closest-side` Gradient extends to the background side closest to the gradient's center.

The `position` defines where the gradient radiates from and can be expressed in coordinates using pixels, percentages of the element's width and height, or with the keywords: `left`, `center`, `right`, `top`, and `bottom`. The default is to place the gradient within the center of the background.

Finally the `color-stop1`, `color-stop2` ... values are the colors and their stopping positions within the gradient and have the same interpretation used for linear gradients except they mark stopping points as the gradient radiates outward. Note that the color stops are optional, just as they are in linear gradients. For example the following function defines a circular gradient radiating from the horizontal and vertical center of the background through the colors red, yellow, and blue:

```
radial-gradient(circle closest-corner at center center,
               red, yellow, blue)
```

The gradient ends when it reaches the closest background corner. Anything outside of the gradient will be a solid blue.

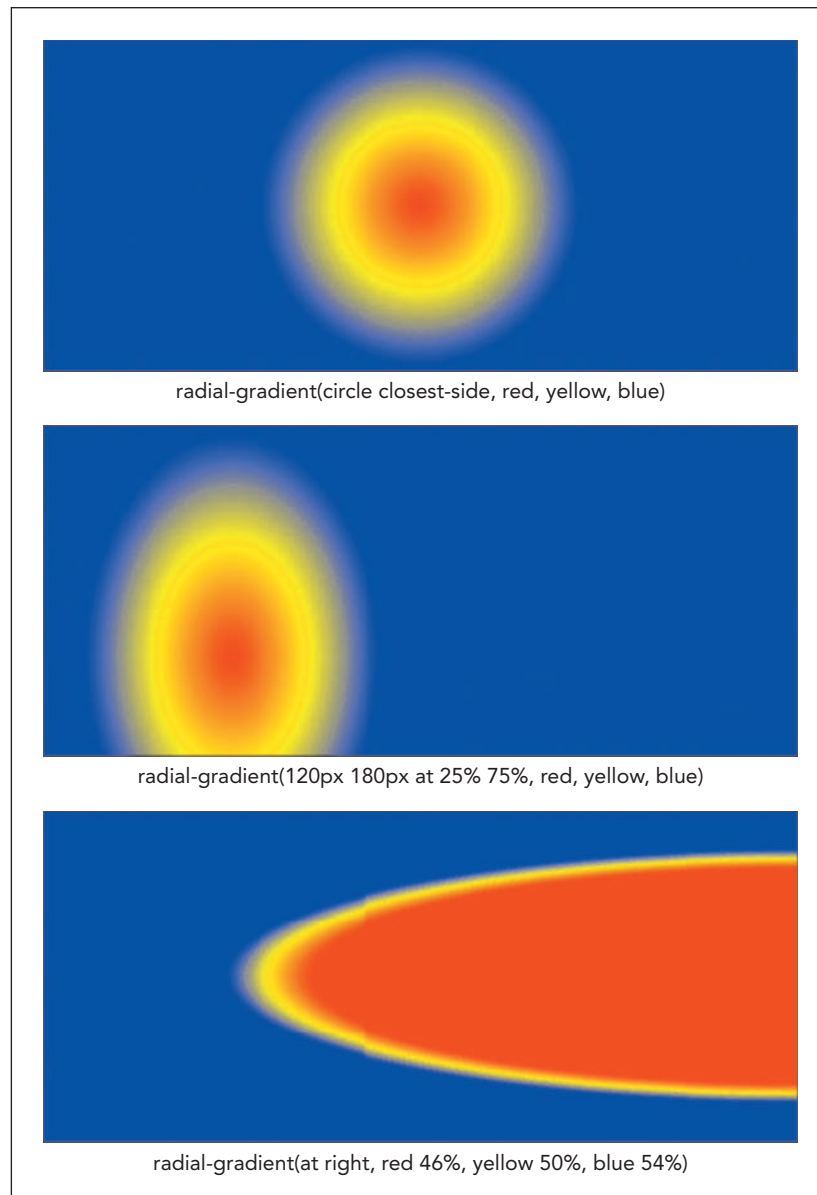
Figure 4–39 shows other examples of the different effects that can be accomplished using the `radial-gradient` function. Note that when parameters of the `radial-gradient` function are omitted they take their default values.

**TRY IT**

You can explore how to create your own radial gradients using the `demo_radial.html` file from the `html04` ► `demo` folder.

Figure 4–39

## Examples of radial gradients



Kevin would like you to apply a radial gradient to the background of the `aside` element. The gradient will start from a white center blending into to a medium green and then into a darker shade of green.

**To apply a radial gradient:**

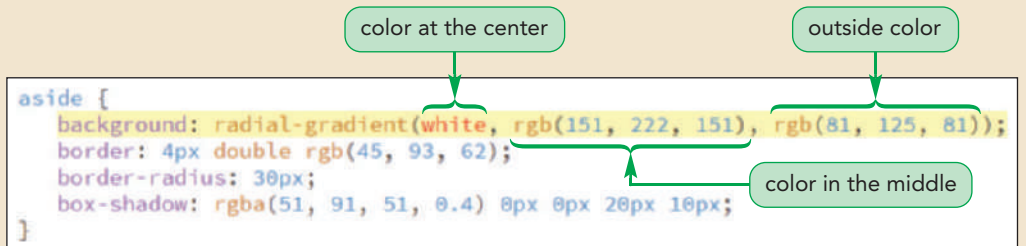
1. Return to the `tb_visual1.css` file in your editor and go to the `Aside Styles` section.
2. Add the following style to the style rule for the `aside` element:

```
background:
radial-gradient(white, rgb(151, 222, 151),
                rgb(81, 125, 81));
```

Note that this style supersedes the previous background style created in the `tb_styles1.css` style sheet. Figure 4–40 highlights the code to create the radial gradient.

Figure 4–40

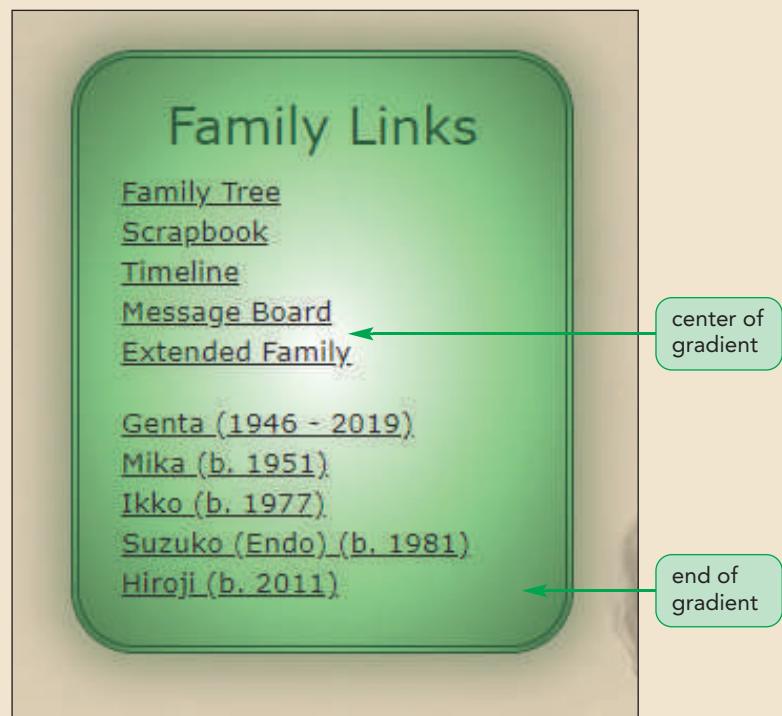
## Applying a radial gradient



3. Save your changes and reload `tb_komatsu.html` in your browser. Figure 4–41 shows the radial gradient within the `aside` element.

Figure 4–41

## Aside element with radial gradient background



© imtmphoto/Shutterstock.com

Kevin likes the effect of the radial gradient on the `aside` element and feels that it works well with the glowing effect you added earlier.

### Gradients and Browser Extensions

The gradient functions were heavily revised as they went from being browser-specific properties to the final syntax approved by the W3C. If you work with older browsers, you may need to accommodate their versions of these gradient functions. For example, the following linear gradient that blends red to blue going in the direction to the right edge of the background

```
linear-gradient(to right, red, blue)
```

would be expressed using the old WebKit gradient function as:

```
-webkit-gradient(linear, left, right, from(red), to(blue))
```

Other older versions of browsers such as Mozilla, Internet Explorer, and Opera have their own gradient functions with different syntax. You can study these functions using the online support at the browser websites or doing a search on the Web for CSS gradient functions.

Note that not all browser extensions support the same types of gradients, which means that it is difficult and sometimes impossible to duplicate a particular gradient background for every browser. Thus, you should not make gradients an essential feature of your design if you want to be compatible with older browsers.

## Repeating a Gradient

As you add more color stops, the gradient function can become unwieldy and overly complicated. One alternative is to repeat the gradient design. You can repeat linear and radial gradients using the functions

```
repeating-linear-gradient(params)  
repeating-radial-gradient(params)
```

### TRY IT

You can create your own repeating gradients using the `demo_repeat_linear.html` and `demo_repeat_radial.html` files from the `html04` ▶ `demo` folder.

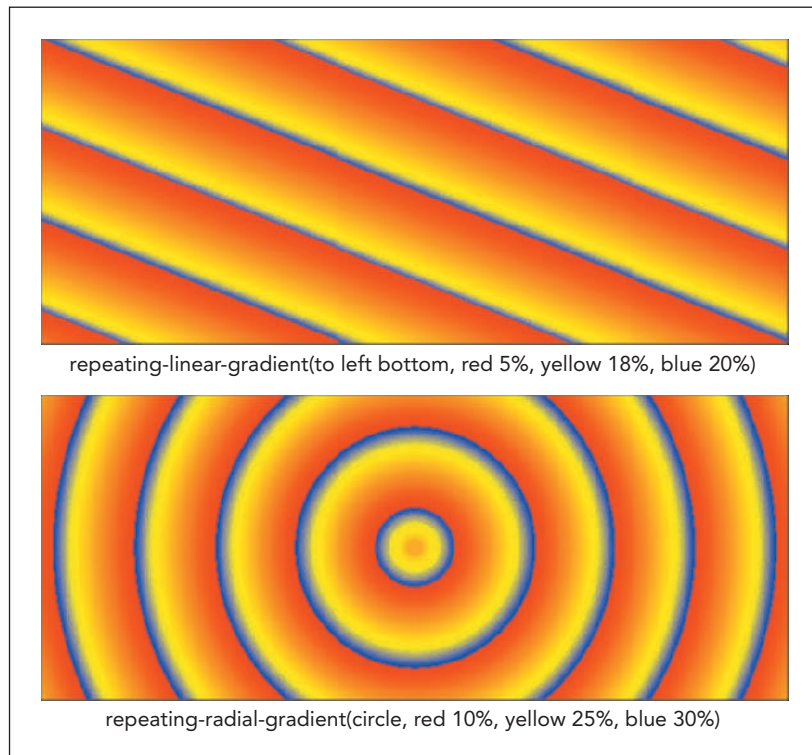
where *params* are the parameters of the `linear-gradient` or the `radial-gradient` functions already discussed. The only requirement for a repeating gradient is that a stopping position is required for the last color in the list that is less than the size of the object background. Once the last color in the color list is reached, the gradient starts over again. For example, the following function repeats a vertical gradient starting with white transitioning to black, transitioning back to white at 10% of the height of the object, and then repeating that pattern each time it reaches the next 10% of the height of the object:

```
repeating-linear-gradient(white, black 10%)
```

Figure 4-42 shows some other examples of repeating linear and radial gradients.

Figure 4–42

## Repeating a gradient



## REFERENCE

**Creating a Gradient**

- To create a linear gradient, use the function

```
linear-gradient(direction, color-stop1, color-stop2, ...)
```

where *direction* is the direction of the gradient and *color-stop1*, *color-stop2*, and so on are the colors and their stopping positions within the gradient.

- To create a radial gradient, use the function

```
radial-gradient(shape size at position, color-stop1,  
color-stop2, ...)
```

where *shape* defines the shape of the gradient, *size* sets the gradient size, *position* places the center of the gradient, and *color-stop1*, *color-stop2*, and so on are the colors and their stopping positions within the gradient.

- To repeat a gradient, use the functions

```
repeating-linear-gradient(params)  
repeating-radial-gradient(params)
```

where *params* are the parameters of the linear-gradient or the radial-gradient functions.

The last visual effect that Kevin wants you to add to the Komatsu Family page is to make the figure box semi-transparent so that it blends in better with its background.



## INSIGHT

### Gradients as Images

Gradients can be treated as images and thus can be tiled within a background or used with the `border-image` style. For example, the following style rule creates a background of radial gradients repeated in the horizontal and vertical directions:

```
background-size: 50% 50%;
background-image-repeat: repeat;
background-image: radial-gradient(circle, yellow, blue);
```

Note that setting the background size to 50% 50% sets the width and height of each radial gradient to half of the width and height of the object. By setting the `background-image-repeat` style to `repeat`, the entire background is filled with radial gradients, resulting in two rows of two gradient images.

The following code shows how to use a linear gradient as a border image:

```
border: 30px solid transparent;
border-image: (linear-gradient(yellow, blue)), 15);
```

with the image file replaced by the `linear-gradient()` function. The gradient is generated for the entire object but is trimmed with a slice width of 15px to form the gradient border.

## TRY IT

You can explore gradients as backgrounds in the `demo_linear_image.html` and `demo_radial_image.html` files in the `html04` ▶ demo folder

## TRY IT

You can explore gradient borders using the `demo_gradient_border.html` file in the `html04` ▶ demo folder.

## Creating Semi-Transparent Objects

In Tutorial 2, you learned that you could create semi-transparent colors that blend with the background color. You can also create whole page objects that are semi-transparent using the following `opacity` property:

```
opacity: value;
```

where *value* ranges from 0 (completely transparent) up to 1 (completely opaque). For example, the following style rule makes the page body 70% opaque, allowing a bit of the browser window background to filter through

```
body {
  opacity: 0.7;
}
```

## REFERENCE

### Making a Semi-transparent Object

- To make a page object semi-transparent, use the property

```
opacity: value;
```

where *value* ranges from 0 (completely transparent) up to 1 (completely opaque).

Kevin suggests that you set the opacity of the figure box to 55% in order to blend the figure box with the paper texture background you added to the `article` element.

### To create a semi-transparent object:

1. Return to the `tb_visual1.css` file in your editor and scroll up to the Figure Box Styles section.
2. Within the style rule for the `figure` element, insert the following style:

```
opacity: 0.55;
```

Figure 4–43 highlights the code to make the figure box semi-transparent.

Figure 4–43

### Creating a semi-transparent object

```
figure {  
  border-style: solid;  
  border-width: 25px;  
  border-image: url(tb_border.png) 50 repeat;  
  margin: 20px auto 0px;  
  opacity: 0.55;  
  width: 80%;  
}
```

sets the opacity of the figure box to 55%

3. Save your changes and reload `tb_komatsu.html` in your browser. Figure 4–44 displays the semi-transparent figure box with part of the background paper texture showing through.

Figure 4–44

### Changing the opacity of the figure box



part of the background page texture shows through in the figure box



## PROSKILLS

### Written Communication: How to Use Visual Effects

The CSS visual styles can add striking effects to your website, but they might not be supported by older browsers. This leaves you with the dilemma of when and how to use these styles. Here are some tips to keep in mind when applying visual effects to your website:

- Because not every user will be able to see a particular visual effect, design your page so that it is still readable to users with or without the effect.
- Be aware that some visual effects that flicker or produce strobe-like effects can cause discomfort and even photo-epileptic seizures in susceptible individuals. Avoid clashing color combinations and optical illusions that can cause these conditions.
- If you need to create a cross-browser solution, use browser extensions and be aware that the browser extension syntax might not match the syntax of the CSS standard.
- Consider using graphic images to create your visual effects. For example, rather than using the CSS gradient functions, create a background image file containing the gradient effect of your choice.

No matter how you employ visual effects on your website, remember that the most important part of your site is its content. Do not let visual effects distract from your content and message.

At this point you've completed your work on the design of the Komatsu Family page. In the next session, you will learn how to use CSS to apply transformations and filters. You will also learn how to work with image maps to create linkable images. Close any open files now.

## REVIEW

### Session 4.2 Quick Check

1. Provide a style rule to create a red text shadow that is 5 pixels to the right and 10 pixels up from the text with a blur of 15 pixels.
  - a. `text-shadow: red 5px 10px 15px;`
  - b. `text-shadow: red -5px 10px 15px;`
  - c. `text-shadow: red 5px -10px 15px;`
  - d. `text-shadow: red -5px -10px 15px;`
2. Provide a style rule to add a blue drop shadow to a page object that is 2 pixels to the left, 5 pixels up and with a blur radius of 8 pixels.
  - a. `box-shadow: blue 2px 5px 8px;`
  - b. `box-shadow: blue -2px 5px 8px;`
  - c. `box-shadow: blue 2px -5px 8px;`
  - d. `box-shadow: blue -2px -5px 8px;`
3. Provide a style to add a green interior drop shadow that is 2 pixels to the left, 5 pixels up and a blur radius of 8 pixels.
  - a. `box-shadow: green 2px 5px 8px;`
  - b. `box-shadow: green 2px 5px 8px inset;`
  - c. `box-shadow: green -2px -5px 8px;`
  - d. `box-shadow: green -2px -5px 8px inset;`

4. Provide a style rule to create a red halo effect with no shadow offset, a blur of 15 pixels and a shadow size that is 10 pixels larger than the element.
  - a. `box-shadow: red 0px 0px 15px 10px;`
  - b. `box-shadow: red 0px 0px 10px 15px;`
  - c. `box-shadow: red 15px 10px 0px 0px;`
  - d. `box-shadow: red 10px 15px 0px 0px;`
5. Provide code for a linear gradient that moves in the direction of the lower-left corner of the element through the colors: orange, yellow, and green.
  - a. `linear-gradient(left bottom, orange, yellow, green)`
  - b. `linear-gradient(bottom left, orange, yellow, green)`
  - c. `linear-gradient(to left bottom, orange, yellow, green)`
  - d. `linear-gradient(from right top, orange, yellow, green)`
6. Create a linear gradient that moves at a 15 degree angle with the color orange stopping at 10% of the background, yellow stopping at 50%, and green stopping at 55%.
  - a. `linear-gradient(15deg, 10% orange, 50% yellow, 55% green)`
  - b. `linear-gradient(15deg, orange 10%, yellow 50%, green 55%)`
  - c. `linear-gradient(15deg, orange 10% yellow 50% green 55%)`
  - d. `linear-gradient(195deg, 10% orange, 50% yellow, 55% green)`
7. Create a radial gradient that extends to the farthest background corner, going through the colors orange, yellow, and green.
  - a. `radial(farthest-corner, orange, yellow, green)`
  - b. `radial-gradient(from farthest-corner, orange, yellow, green)`
  - c. `radial-gradient(farthest-corner, orange, yellow, green)`
  - d. `radial(farthest-corner, orange, yellow, green)`
8. Create a repeating circular gradient of orange, yellow, and green bands centered at the right edge of the element with the colors stopped at 10%, 20%, and 30% respectively.
  - a. `radial-gradient(circle at right center, orange 10%, yellow 20%, green 30%)`
  - b. `radial-gradient-repeat(circle at right center, orange 10%, yellow 20%, green 30%)`
  - c. `radial-gradient-repeat(circle at right center, orange 10% 10%, yellow 20% 20%, green 30% 30%)`
  - d. `repeating-radial-gradient(circle at right center, orange 10%, yellow 20%, green 30%)`
9. Create a style rule to set the opacity to 75%.
  - a. `transparency: 25%;`
  - b. `transparency: 0.25;`
  - c. `opacity: 75%;`
  - d. `transform: opacity(0.75);`

# Session 4.3 Visual Overview:

**Perspective** is used in 3D transformations to measure how rapidly objects appear to recede from or approach the viewer.

The **transform** property is used to rotate, rescale, skew, or shift a page object.

The **filter** property is used to modify an object's color, brightness, contrast, or general appearance.

The grayscale function displays the object in grayscale.

The saturate and contrast functions increase the color saturation by 50% and increase the color contrast by 20%.

```

/* Transformation and Filter Styles */
article {
  perspective: 600px;
}

figure#figure1 {
  transform: rotateX(30deg) translateZ(50px);
  filter: sepia(0.8);
}

figure#figure2 {
  transform: rotate(-40deg) scale(0.8, 0.8) translate(20px, -100px)
  rotateZ(30deg) rotateY(60deg);
  filter: grayscale(1);
}

figure#figure3 {
  transform: rotate(10deg) scale(0.9, 0.9)
  translateY(-120px) rotateY(-70deg) translateZ(-20px);
  filter: saturate(1.5) contrast(1.2);
}

```

The rotateX and translateZ functions rotate the object 30° around the x-axis and move it 50 pixels toward the viewer.

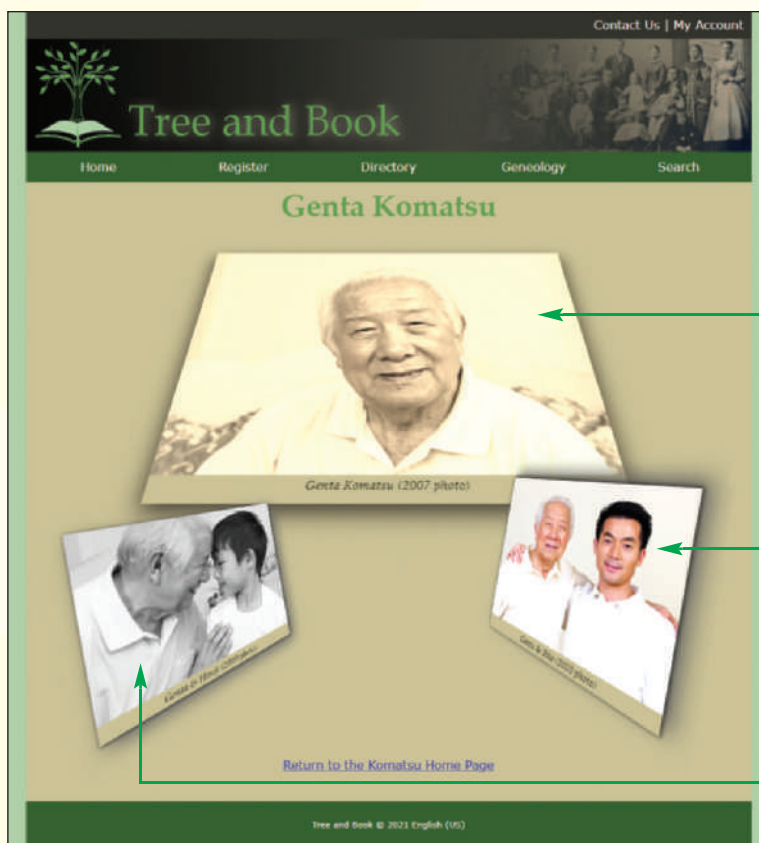
The rotateZ and rotateY functions rotate the object 30° around the z-axis and 60° around the y-axis.

The sepia function displays the object in a sepia tone.

The scale function reduces the object to 90% of its default size.

Logo Design Studio Pro; Source: wiki Media;  
© imtmphoto/Shutterstock.com

# Transformations and Filters



The image uses a sepia tone and is rotated around the x-axis.

This shows a rescaled image with increased color saturation and contrast; it is rotated around the y-axis.

This shows the image in grayscale and rotated around the z and y axes.

Source: Design Studio Pro; Source: Wikimedia Commons; imtmphoto/Shutterstock.com

## Transforming Page Objects

In this session, you will examine some CSS styles that can be used to transform the appearance of page objects through rotation, rescaling, and translation in space. To accomplish these transformations, you'll use the following `transform` property:

```
transform: effect(params);
```

where *effect* is a transformation function that will be applied to the page object and *params* are any parameters required by the function. Figure 4–45 describes some of the CSS transformation functions.

Figure 4–45

CSS 2D transformation functions

Function	Description
<code>translate(offX, offY)</code>	Moves the object <i>offX</i> pixels to the right and <i>offY</i> pixels down; negative values move the object to the left and up
<code>translateX(offX)</code>	Moves the object <i>offX</i> pixels to the right; negative values move the object to the left
<code>translateY(offY)</code>	Moves the object <i>offY</i> pixels down; negative values move the object up
<code>scale(x, y)</code>	Resizes the object by a factor of <i>x</i> horizontally and a factor of <i>y</i> vertically
<code>scaleX(x)</code>	Resizes the object by a factor of <i>x</i> horizontally
<code>scaleY(y)</code>	Resizes the object by a factor of <i>y</i> vertically
<code>skew(angleX, angleY)</code>	Skews the object by <i>angleX</i> degrees horizontally and <i>angleY</i> degrees vertically
<code>skewX(angleX)</code>	Skews the object by <i>angleX</i> degrees horizontally
<code>skewY(angleY)</code>	Skews the object by <i>angleY</i> degrees vertically
<code>rotate(angle)</code>	Rotates the object by <i>angle</i> degrees clockwise; negative values rotate the object counter-clockwise
<code>matrix(n, n, n, n, n, n)</code>	Applies a 2D transformation based on a matrix of six values

For example, to rotate an object 30° clockwise, you would apply the following style using the `rotate` function:

```
transform: rotate(30deg);
```

To rotate an object counter-clockwise, you would use a negative value for the angle of rotation. Thus, the following style rotates an object 60° counter-clockwise:

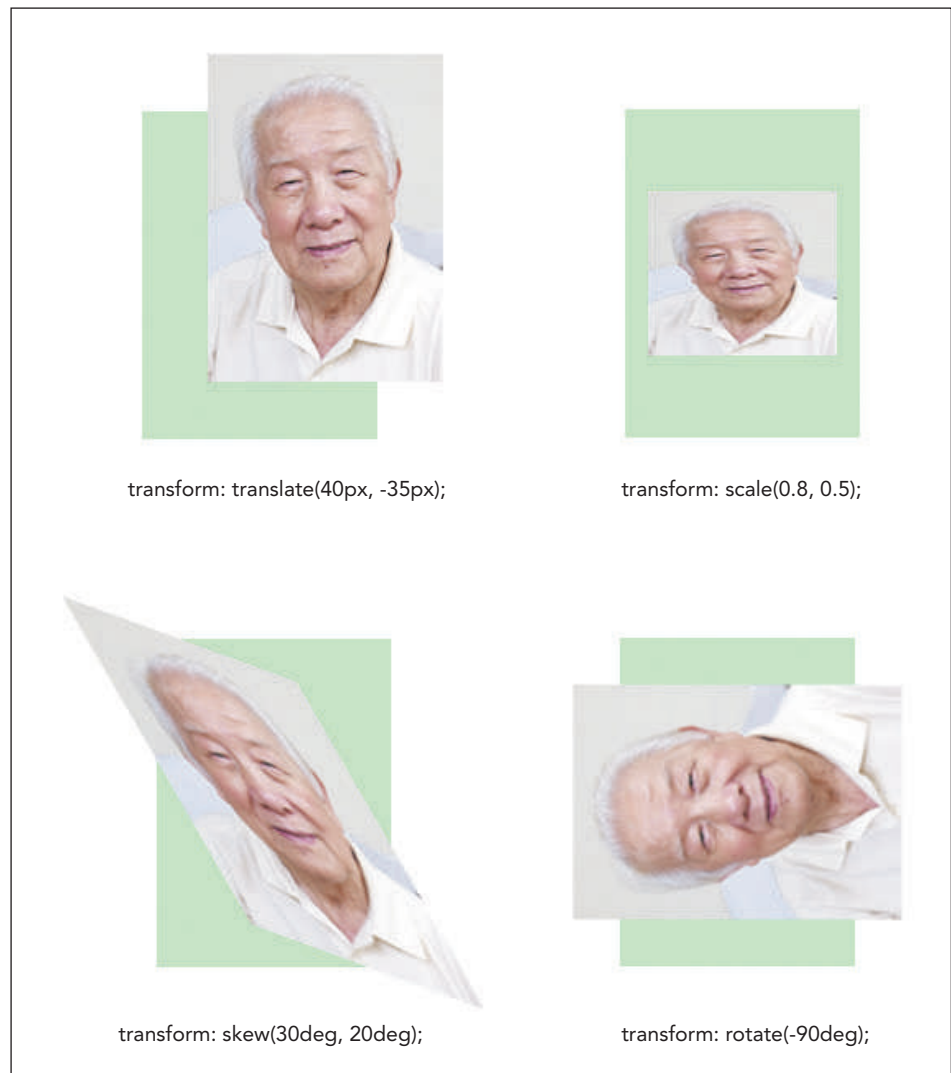
```
transform: rotate(-60deg);
```

Figure 4–46 displays the effects of other transformation functions on a sample page image.



Figure 4–46

## Examples of CSS Transformations



© imtmphoto/Shutterstock.com

**TRY IT**

You can explore different 2D CSS transformations using the demo pages `demo_transform2d.html` and `demo_transform2dm.html` from the `html04` ► demo folder.

Transforming an object has no impact on the page layout. All of the other page objects will retain their original positions.

You can apply multiple transformations by placing the effect functions in a space-separated list. In this situation, transformations are applied in the order listed. For example, the following style first rotates the object 30° clockwise and then shifts it 20 pixels to the right.

```
transform: rotate(30deg) translateX(20px);
```

**REFERENCE****Applying a CSS Transformation**

- To apply a transformation to a page object, use the property

```
transform: effect(params);
```

where *effect* is a transformation function that will be applied to the page object and *params* are any parameters required by the function.

The website has pages with photos for each individual in the Komatsu family. Kevin wants you to work on transforming the photos on Genta Komatsu's page. Kevin has already created the page content and a layout and typographical style sheet but wants you to work on the style sheet containing the visual effects. Open the Genta Komatsu page now.

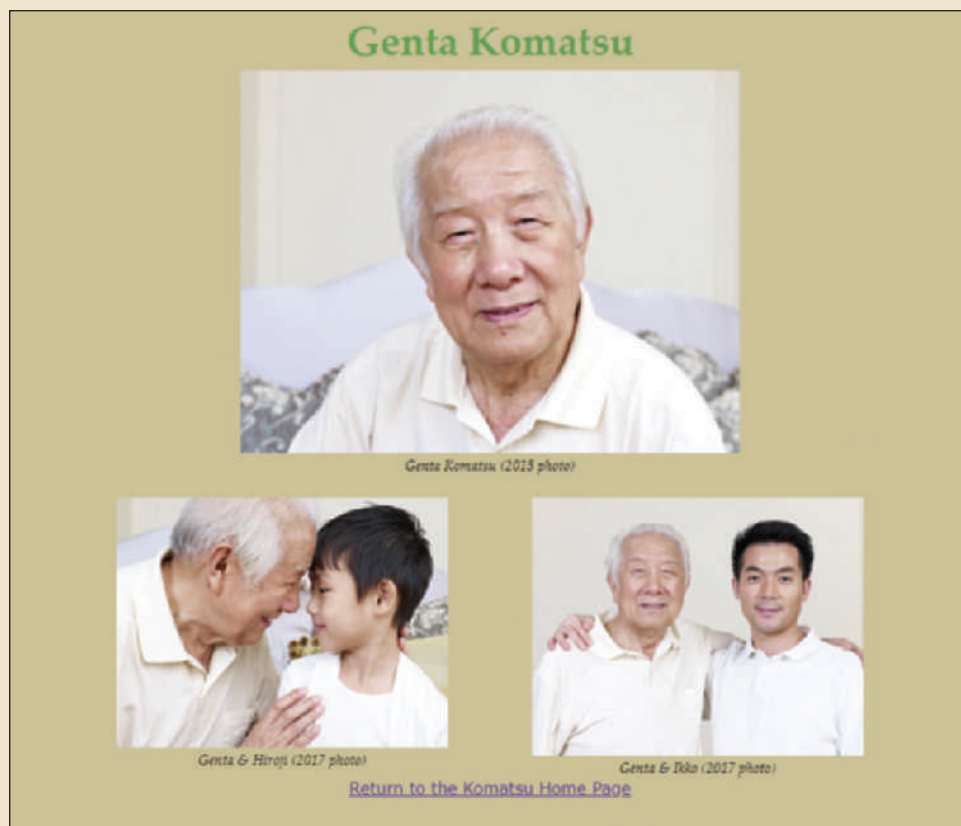
### To open the Genta Komatsu page:

1. Use your editor to open the **tb\_genta\_txt.html** and **tb\_visual2\_txt.css** files from the html04 ► tutorial folder. Enter **your name** and **the date** in the comment section of both files and save them as **tb\_genta.html** and **tb\_visual2.css** respectively.
2. Return to the **tb\_genta.html** file in your editor. Within the document head, insert the following `link` elements to link the page to the `tb_reset.css`, `tb_styles2.css`, and `tb_visual2.css` style sheet files.

```
<link href="tb_reset.css" rel="stylesheet" />
<link href="tb_styles2.css" rel="stylesheet" />
<link href="tb_visual2.css" rel="stylesheet" />
```
3. Take some time to scroll through the contents of the file. Note that the document content consists mainly of three figure boxes each containing a different photo of Genta Komatsu.
4. Close the file, saving your changes.
5. Open the **tb\_genta.html** file in your browser. Figure 4–47 shows the initial layout and design of the page content.

Figure 4–47

Initial design of the Genta Komatsu page



Kevin feels that the page lacks visual interest. He suggests you transform the bottom row of photos by rotating them and shifting them upward to partially cover the main photo, creating a collage-style layout. Apply the `transform` property now to make these changes.

### To apply the transform style:

1. Go to the `tb_visual2.css` file in your editor and scroll as needed to the Transformation Styles section.
2. Insert the following style rule to rotate the figure2 figure box 40° counter-clockwise, reduce it to 80% of its former size, and shift it 20 pixels to the right and 100 pixels up. Also, add a style to create a drop shadow using the code that follows:

```
figure#figure2 {
    transform: rotate(-40deg) scale(0.8, 0.8)
              translate(20px, -100px);
    box-shadow: rgb(101, 101, 101) 10px 10px 25px;
}
```

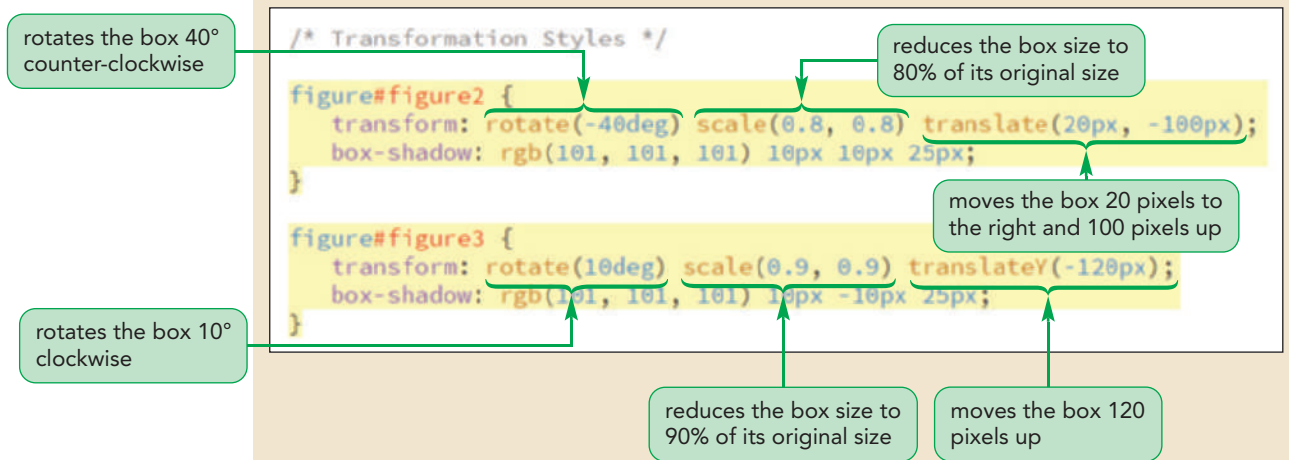
3. Add the following style rule to rotate the figure3 figure box 10° clockwise, resize it to 90% of its current size, and shift it 120 pixels upward. Also add a drop shadow to the figure box using the following style rule:

```
figure#figure3 {
    transform: rotate(10deg) scale(0.9, 0.9)
              translateY(-120px);
    box-shadow: rgb(101, 101, 101) 10px -10px 25px;
}
```

Figure 4–48 describes the newly added style rules.

Figure 4–48

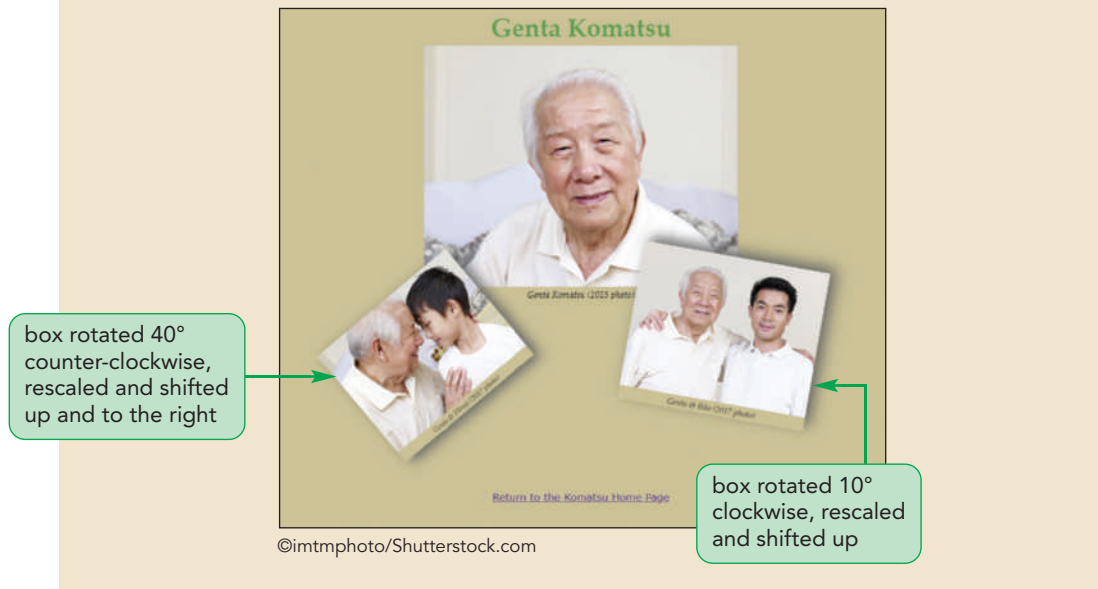
### Transforming the figure boxes



4. Save your changes to the file and then reload `tb_genta.html` in your browser. Figure 4–49 shows the revised design of the page's content.

Figure 4-49

Viewing the transformed figure boxes



The transformations you applied rotated the figure boxes along a two-dimensional or 2D space that consisted of a horizontal and vertical axis. CSS also supports transformations that operate in a three-dimensional or 3D space.

### Setting the Transformation Origin

#### INSIGHT

By default, transformations originate in the center of the page object. When an object is rotated, for example, it rotates the specified number of degrees around its horizontal and vertical center. If you wish to rotate around a different point, such as the object's left edge or bottom-right corner, you can modify the transformation origin using the following `transform-origin` property:

```
transform-origin: horizontal vertical;
```

where *horizontal* and *vertical* specify the location of the origin of the transformation. For example, the following set of style rules used in conjunction will rotate an object 30 degrees clockwise around its bottom-right corner:

```
transform: rotate(30deg);
transform-origin: right bottom;
```

#### TRY IT

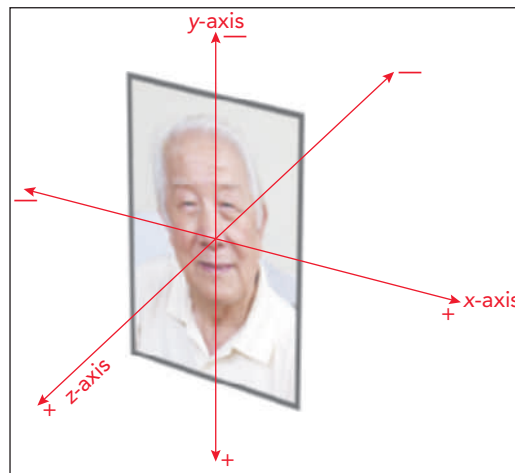
You can explore the `transform-origin` property in the `demo_transform2d.html` and `demo_transform2dm.html` files from the `html04` demo folder.

## Transformations in Three Dimensions

A **3D transformation** is a change that involves three spatial axes: an *x*-axis that runs horizontally across the page, a *y*-axis that runs vertically, and a *z*-axis that comes straight out of the page toward and away from the viewer. Positive values along the axes are to the right, down, and toward the reader; negative values are to the left, up, and away from the reader (see Figure 4-50.)

Figure 4–50

## A page object viewed in 3D



© imtmphoto/Shutterstock.com

With the addition of a third spatial axis, you can create effects in which an object appears to zoom toward and away from users, or to rotate in three dimensional space. Figure 4–51 describes the 3D transformations supported by CSS.

Figure 4–51

## CSS 3D transformation functions

Function	Description
<code>translate3d(offX, offY, offZ)</code>	Shifts the object <i>offX</i> pixels horizontally, <i>offY</i> pixels vertically, and <i>offZ</i> pixels along the z-axis
<code>translateX(offX)</code> <code>translateY(offY)</code> <code>translateZ(offZ)</code>	Shifts the object <i>offX</i> , <i>offY</i> , or <i>offZ</i> pixels along the specified axis
<code>rotate3d(x, y, z, angle)</code>	Rotates the object around the three-dimensional vector ( <i>x</i> , <i>y</i> , <i>z</i> ) at a direction of <i>angle</i>
<code>rotateX(angle)</code> <code>rotateY(angle)</code> <code>rotateZ(angle)</code>	Rotates the object around the specified axis at a direction of <i>angle</i>
<code>scale3d(x, y, z)</code>	Resizes the object by a factor of <i>x</i> horizontally, a factor of <i>y</i> vertically, and a factor of <i>z</i> along the z-axis
<code>scaleX(x)</code> <code>scaleY(y)</code> <code>scaleZ(z)</code>	Resizes the object by a factor of <i>x</i> , <i>y</i> , or <i>z</i> along the specified axis
<code>perspective(p)</code>	Sets the size of the perspective effect to <i>p</i>
<code>matrix3d(n, n, ..., n)</code>	Applies a 3D transformation based on a matrix of 16 values

For example the following style rotates the object 60° around the x-axis, making it appear as if the top of the object is farther from the viewer and the bottom is closer to the viewer.

```
transform: rotateX(60deg);
```

To truly create the illusion of 3D space however, you also need to set the perspective of that space.

**TRY IT**

View 3D rotations and perspective values with the `demo_3dview.html` file in the `html04 ▶ demo` folder.

**Understanding Perspective**

Perspective is a measure of how rapidly objects appear to recede from the viewer in a 3D space. You can think of perspective in terms of a pair of railroad tracks that appear to converge at a point, known as the **vanishing point**. A smaller perspective

value causes the tracks to converge over an apparently shorter distance while a larger perspective value causes the tracks to appear to go farther before converging.

You define the perspective of a 3D space using the `perspective` property

```
perspective: value;
```

where `value` is a positive value that measures the strength of the perspective effect with lower values resulting in more extreme distortion. For example, the following style rule sets the perspective of the space within the `div` element to 400 pixels.

```
div {
  perspective: 400px;
}
```

Any 3D transformations applied to children of that `div` element will assume a perspective value of 400 pixels. Perspective can also be set for individual transformations using the following `perspective` function:

```
transform: perspective(value);
```

Thus, the following style rule sets the perspective only for the `figure1` figure box within the `div` element as the figure box is rotated 60° around the x-axis.

```
div figure#figure1 {
  transform: perspective(400px) rotateX(60deg);
}
```

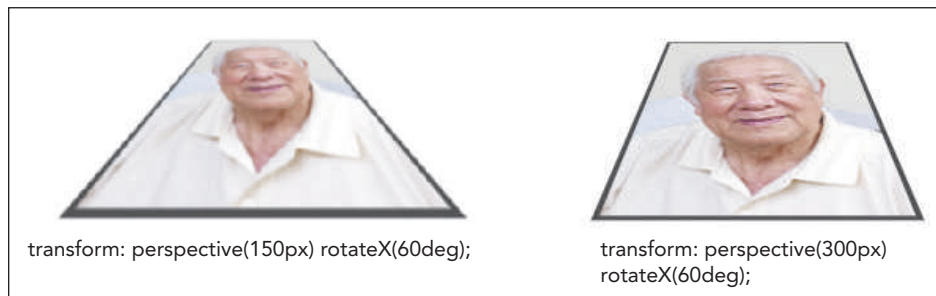
You use the `perspective` property when you have several transformed objects within a container that all need to appear within the same 3D space with a common perspective. You use the `perspective` function when you have only one object that needs to be transformed in the 3D space. Figure 4–52 compares two different perspective values for an object rotated 60° around the x-axis in 3D space.

Figure 4–52

## Transformations in three dimensions

## TRY IT

Explore multiple 3D transformations with the `demo_transform3dm.html` file in the `html04 ▶ demo` folder.



© imtmphoto/Shutterstock.com

Note that the smaller perspective value results in a more extreme distortion as the top of the object appears to more quickly recede from the viewer while the bottom appears to approach the viewer more rapidly.

## Setting Perspective in 3D

## REFERENCE

- To set the perspective for a container and the objects it contains, use the property `perspective: value;`

where `value` is a positive value that measures the strength of the perspective effect with lower values resulting in more extreme distortion.

- To set the perspective of a single object or to set the perspective individually of objects within a group of objects, use the `perspective` function

```
transform: perspective(value);
```

Add a 3D transformation to each of the three figure boxes in the Genta Komatsu page, making it appear that they have been rotated in three dimensional space along the x-, y-, and z-axes, setting the perspective value to 600 pixels for all of the objects in the page article.

### To apply the 3D transformations:

1. Return to the `tb_visual2.css` file in your editor.
2. Directly after the Transformation Styles comment, insert the following style rule to set the perspective of the 3D space of the `article` element.
 

```
article {
    perspective: 600px;
}
```
3. Next, insert the following style rule for the `figure1` figure box to rotate it 30° around the x-axis, shift it 50 pixels along the z-axis, and add a drop shadow.
 

```
figure#figure1 {
    transform: rotateX(30deg) translateZ(50px);
    box-shadow: rgb(51, 51, 51) 0px 10px 25px;
}
```
4. Add the following functions to the `transform` property for the `figure2` figure box to rotate the box 30° around the z-axis and 60° around the y-axis:
 

```
rotateZ(30deg) rotateY(60deg)
```
5. Add the following functions to the `transform` property for the `figure3` figure box to rotate the box counter-clockwise 70° around the y-axis and shift it 20 pixels away from the user along the z-axis:
 

```
rotateY(-70deg) translateZ(-20px)
```

Figure 4–53 highlights the 3D transformations styles in the style sheet.

Figure 4–53 Applying 3D transformations

The figure shows a code snippet for CSS 3D transformations. The code is as follows:

```
/* Transformation Styles */
article {
    perspective: 600px;
}
figure#figure1 {
    transform: rotateX(30deg) translateZ(50px);
    box-shadow: rgb(51, 51, 51) 0px 10px 25px;
}
figure#figure2 {
    transform: rotate(-40deg) scale(0.8, 0.8)
        translate(20px, -100px)
        rotateZ(30deg) rotateY(60deg);
    box-shadow: rgb(101, 101, 101) 10px 10px 25px;
}
figure#figure3 {
    transform: rotate(10deg) scale(0.9, 0.9)
        translateY(-120px)
        rotateY(-70deg) translateZ(-20px);
    box-shadow: rgb(101, 101, 101) 10px -10px 25px;
}
```

Callouts explain the following parts of the code:

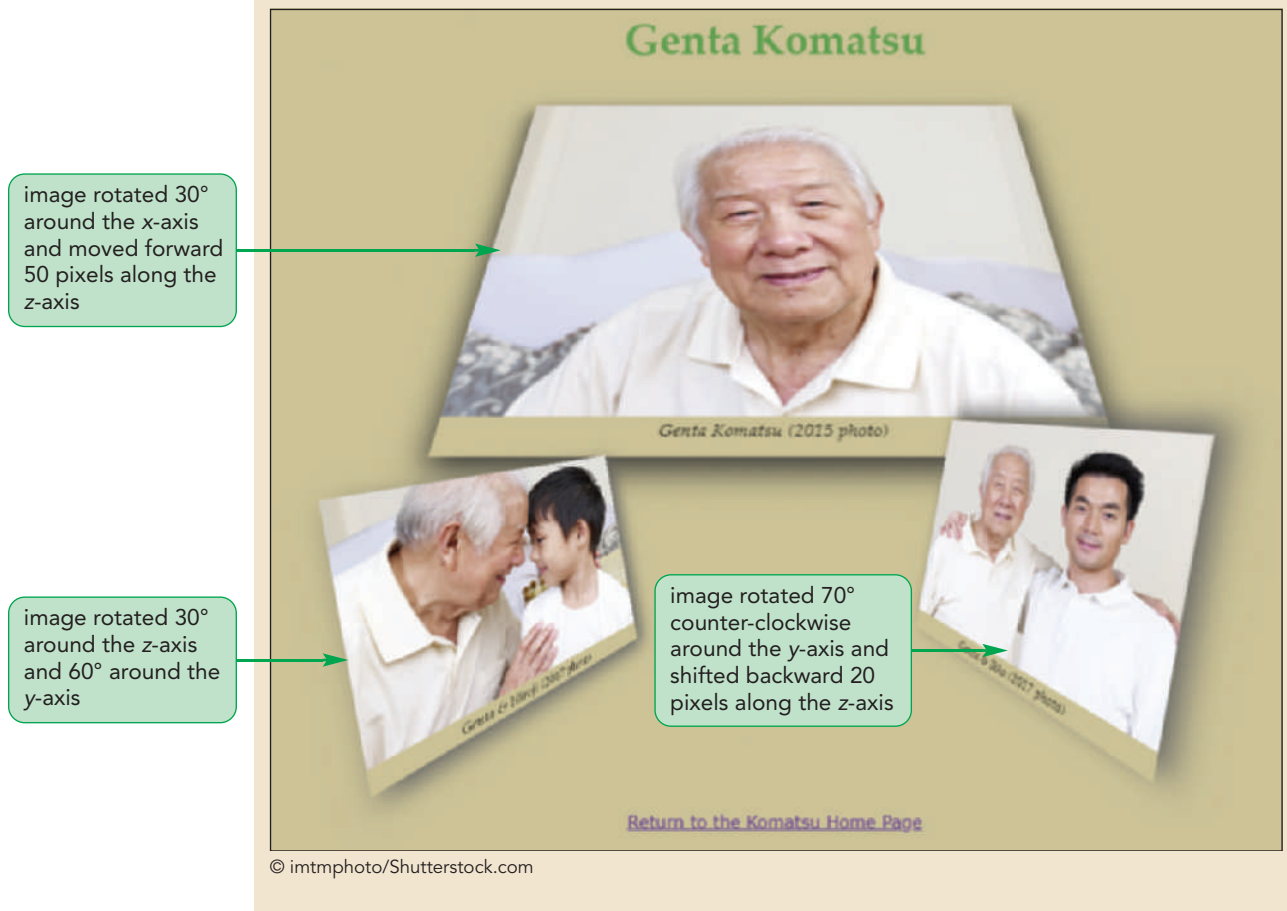
- `perspective: 600px;` sets the perspective of the article space to 600 pixels.
- `transform: rotateX(30deg) translateZ(50px);` rotates the box 30° around the x-axis and shifts it forward 50 pixels along the z-axis.
- `box-shadow: rgb(51, 51, 51) 0px 10px 25px;` adds a box shadow on the box's bottom border.
- `transform: rotate(-40deg) scale(0.8, 0.8) translate(20px, -100px) rotateZ(30deg) rotateY(60deg);` rotates the box 30° around the z-axis and 60° around the y-axis.
- `transform: rotate(10deg) scale(0.9, 0.9) translateY(-120px) rotateY(-70deg) translateZ(-20px);` rotates the box 70° counter-clockwise around the y-axis and shifts it backward 20 pixels along the z-axis.



6. Save your changes to the file and then reload `tb_genta.html` in your browser. Figure 4–54 shows the result of applying 3D transformations to each of the figure boxes on the page.

Figure 4–54

Figure boxes in 3D space



You have only scratched the surface of what can be done using transformations. For example, you can create a mirror image of an object by rotating it 180° around the y-axis. You can create virtual 3D objects like cubes that can be viewed from any angle or spun. You are only limited by your imagination.

### Managing a 3D Space

You might want to have several objects that coexist within the same 3D space. You can do this by creating a container for all those objects, allowing them to share a common 3D perspective using the following `transform-style` property:

```
transform-style: type;
```

where `type` is either `preserve-3d` to preserve the 3D space for all nested elements or `flat` to allow the nested elements to exist within their own separate 3D spaces. For example, the following style rules will pass the same 3D space to all elements nested within the container, including any value assigned to the `perspective` property.

```
#container {  
    transform-style: preserve-3d;  
}
```

#### TRY IT

Explore managing multiple objects within a 3D space in the `demo_preserve3d.html`, `demo_cards.html`, and `demo_cube.html` files from the `html04` ► `demo` folder.

An object in a 3D space is considered to have a front and a back. The default behavior is to allow any text or images on the front to “bleed through” to the back (thus appearing in reverse when the object is rotated around the x or y axes.) You can turn off this feature setting the `backface-visibility` property to `hidden`, which prevents text and images on the front face of the object from appearing on the back face. Setting `backface-visibility` to `visible` restores the default.

## Exploring CSS Filters

A final way to alter an object is through a CSS filter. Filters adjust how the browser renders an image, a background, or a border by modifying the object’s color, brightness, contrast, or general appearance. For example, a filter can be used to change a color image to grayscale, increase the image’s color saturation, or add a blurring effect. Filters are applied using the `filter` property

```
filter: effect(params);
```

where `effect` is a filter function and `params` are the parameters of the function. Figure 4–55 describes the different filter functions supported by most current browsers.

Figure 4–55

## CSS filter functions

Function	Description
<code>blur(<i>length</i>)</code>	Applies a blur to the image where <i>length</i> defines the size of blur in pixels
<code>brightness(<i>value</i>)</code>	Adjusts the brightness where values from 0 to 1 decrease the brightness and values greater than 1 increase the brightness
<code>contrast(<i>value</i>)</code>	Adjusts the contrast where values from 0 to 1 decrease the contrast and values greater than 1 increase the contrast
<code>drop-shadow(<i>offsetX</i> <i>offsetY</i> <i>blur</i> <i>color</i>)</code>	Adds a drop shadow to the image where <i>offsetX</i> and <i>offsetY</i> are horizontal and vertical distances of the shadow, <i>blur</i> is the shadow blurring, and <i>color</i> is the shadow color
<code>grayscale(<i>value</i>)</code>	Displays the image in grayscale from 0, leaving the image unchanged, up to 1, displaying the image in complete grayscale
<code>hue-rotate(<i>angle</i>)</code>	Adjusts the hue by <i>angle</i> in the color wheel where 0deg leaves the hue unchanged, 180deg displays the complimentary colors and 360deg again leaves the hue unchanged
<code>invert(<i>value</i>)</code>	Inverts the color from 0 (leaving the image unchanged), up to 1 (completely inverting the colors)
<code>opacity(<i>value</i>)</code>	Applies transparency to the image from 0 (making the image transparent), up to 1 (leaving the image opaque)
<code>saturate(<i>value</i>)</code>	Adjusts the color saturation where values from 0 to 1 decrease the saturation and values greater than 1 increase the saturation
<code>sepia(<i>value</i>)</code>	Displays the color in a sepia tone from 0 (leaving the image unchanged), up to 1 (image completely in sepia)
<code>url(<i>url</i>)</code>	Loads an SVG filter file from <i>url</i>

Figure 4–56 shows the impact of some of the filter functions on a sample image.

Figure 4–56

## CSS filter examples



**TRY IT**

Explore the CSS filter styles with the `demo_filter.html` file in the `html04 ▶ demo` folder.

Filter functions can be combined in a space-separated list to create new effects. For example, the following style reduces the object's color contrast and applies a sepia tone.

```
filter: contrast(75%) sepia(100%);
```

With multiple filter effects, the effects are applied in the order they are listed. Thus, a style in which the sepia effect is applied first followed by the contrast effect will result in a different image than if the order is reversed.

**REFERENCE****Applying a CSS Filter**

- To apply a CSS filter to a page object, use the property

```
filter: effect(params);
```

where *effect* is a filter function and *params* are the parameters of the function.

Kevin wants you to apply filters to the photos in the Genta Komatsu page. He wants a sepia tone applied to the first photo, a grayscale filter applied to the second photo, and a color enhancement applied to the third photo.

**To apply the CSS filters:**

1. Return the `tb_visual2.css` file in your editor and go down to the Filter Styles section.
2. Change the `figure1` figure box to a sepia tone by adding the following style rule:
 

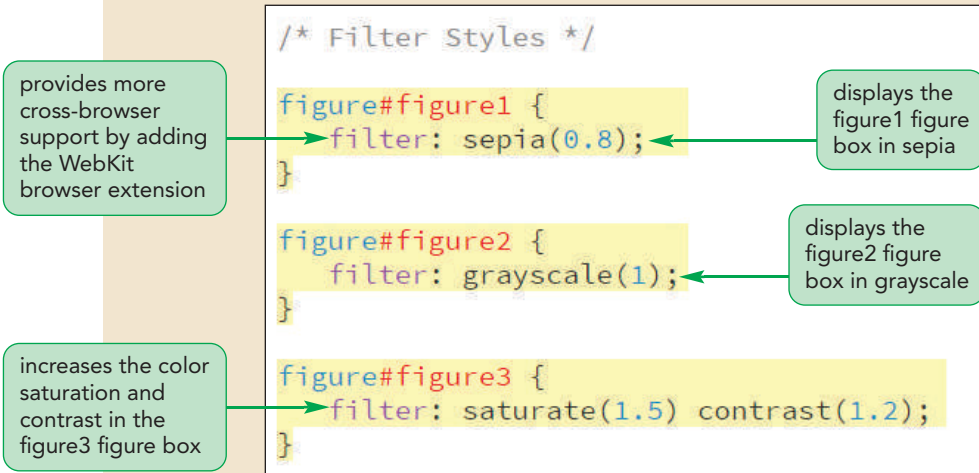
```
figure#figure1 {
    filter: sepia(0.8);
}
```
3. Change the `figure2` figure box to grayscale by adding the style rule:
 

```
figure#figure2 {
    filter: grayscale(1);
}
```
4. Increase the saturation and contrast for the `figure3` figure box with the style rule:
 

```
figure#figure3 {
    filter: saturate(1.5) contrast(1.2);
}
```

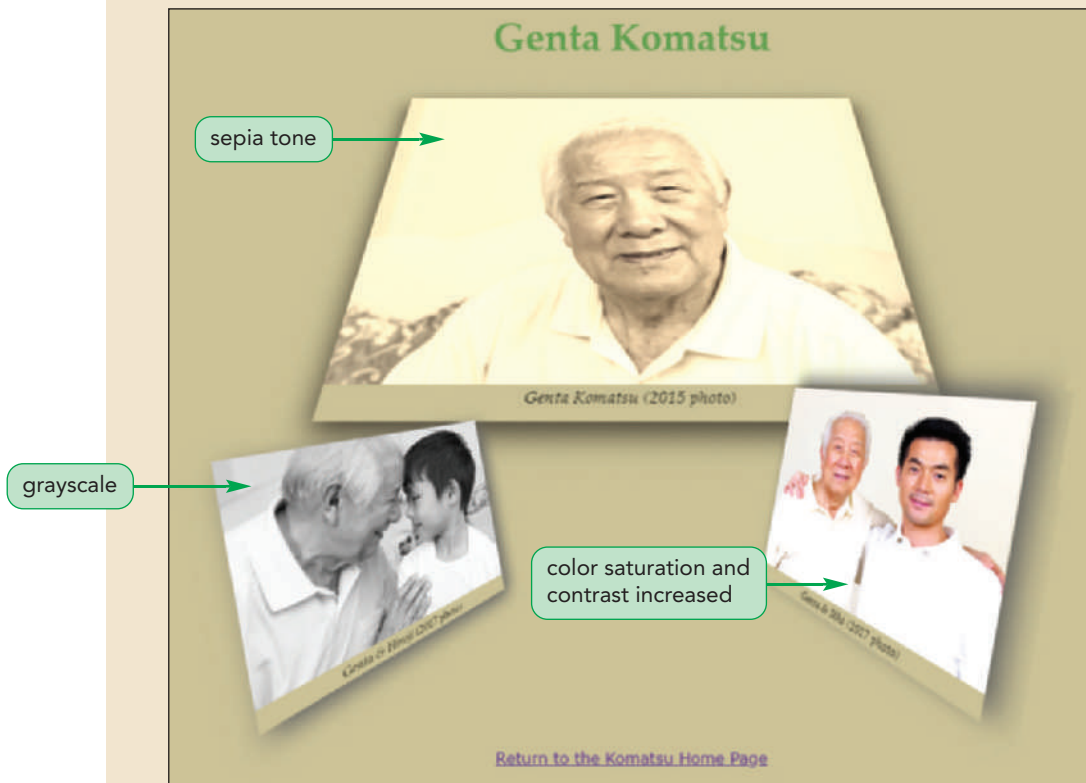
Figure 4–57 highlights the CSS filters added to the style sheet.

Figure 4-57 Applying the filter property



5. Save your changes to the file and then reload `tb_genta.html` in your browser. Figure 4-58 shows the final design of the Genta Komatsu page.

Figure 4-58 Filters applied to the web page photos



## Box Shadows and Drop Shadows

You may wonder why you need a `drop-shadow` filter if you already have the `box-shadow` property. While they both can be used to add shadowing to a page object, one important difference is that the `drop-shadow` filter creates a shadow that traces the shape of the object, while the `box-shadow` property always applies a rectangular shadow. Another important difference is that you can only change the size of a shadow using the `box-shadow` property. Thus, if you want to apply a drop shadow around objects such as text or a circular shape, use the `drop-shadow` filter. However, if you need to create an internal shadow or change the size of the drop shadow shadow, use the `box-shadow` property.

You've completed your redesign of the Genta Komatsu page by adding transformation and filter effects to make a more visually striking page. Kevin now wants to return to the page for the Komatsu family. He wants you to edit the family portrait on the page so that individual pages like the Genta Komatsu page can be accessed by clicking the person's face on the family portrait. You can create this effect using an image map.

## Working with Image Maps

When you mark an inline image as a hyperlink, the entire image is linked to the same file; however, HTML also allows you to divide an image into different zones, or **hotspots**, which can then be linked to different URLs through information provided in an **image map**. HTML supports two kinds of image maps: client-side image maps and server-side image maps. A **client-side image map** is an image map that is defined within the web page and handled entirely by the web browser, while a **server-side image map** relies on a program running on the web server to create and administer the map. Generally client-side maps are easier to create and do not rely on a connection to the server in order to run.

## Defining a Client-Side Image Map

Client-side image maps are defined with the following `map` element

```
<map name="text">
  hotspots
</map>
```

where `text` is the name of the image map and `hotspots` are defined regions within an image that are linked to different URLs. Client-side image maps can be placed anywhere within the body of a web page because they are not actually displayed by browsers but are simply used as references for mapping the locations of the hotspots within the image. The most common practice is to place a `map` element below the corresponding inline image.

Each hotspot within the `map` element is defined using the following `area` element:

```
<area shape="shape" coords="coordinates"
      href="url" alt="text" />
```

where `shape` is the shape of the hotspot region, `coordinates` are the list of points that define the boundaries of that region, `url` is the URL of the hypertext link, and `text` is alternate text displayed for non-graphical browsers.



**TIP**

Do not overlap the hotspots to avoid confusing the user about which hotspot is associated with which URL.

Hotspots can be created as rectangles, circles, or polygons (multisided figures) using *shape* values of *rect*, *circle*, and *poly* respectively. A fourth possible *shape* value, *default*, represents the remaining area of the inline image not covered by any hotspots. There is no limit to the number of hotspots you can add to an image map.

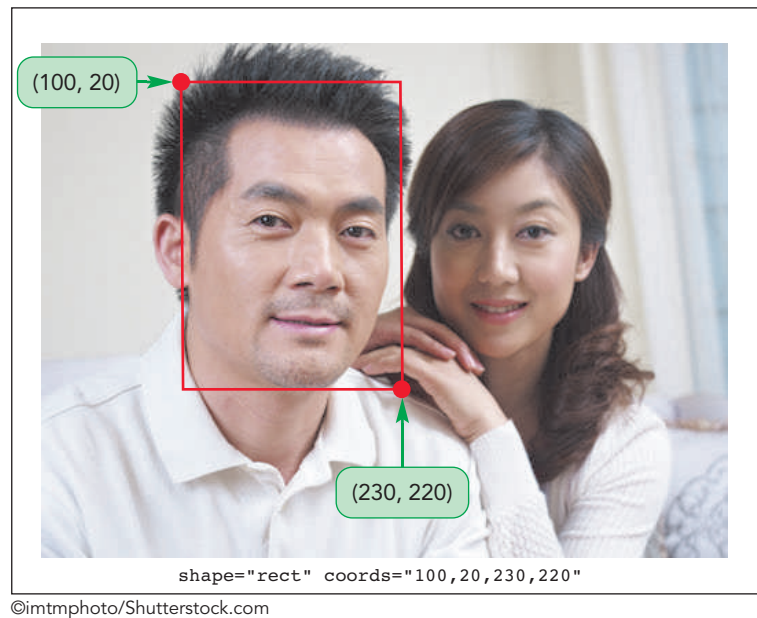
For rectangular hotspots, the *shape* and *coords* attributes have the general form:

```
shape="rect" coords="left,top,right,bottom"
```

where *left*, *top* are the coordinates of the top-left corner of the rectangle and *right*, *bottom* are the coordinates of the bottom-right corner. Coordinates for hotspot shapes are measured in pixels and thus, the following attributes define a rectangular hotspot with the left-top corner at the coordinates (100, 20) and the right-bottom corner at (230, 220):

```
shape="rect" coords="100,20,230,220"
```

To determine the coordinates of a hotspot, you can use either a graphics program such as Adobe Photoshop or image map software that automatically generates the HTML code for the hotspots you define. Note that coordinates are always expressed relative to the top-left corner of the image, regardless of the position of the image on the page. For example, in Figure 4–59, the top-left corner of this rectangular hotspot is 100 pixels right of the image's left border and 20 pixels down from the top border.

**Figure 4–59****Defining a rectangular hotspot**

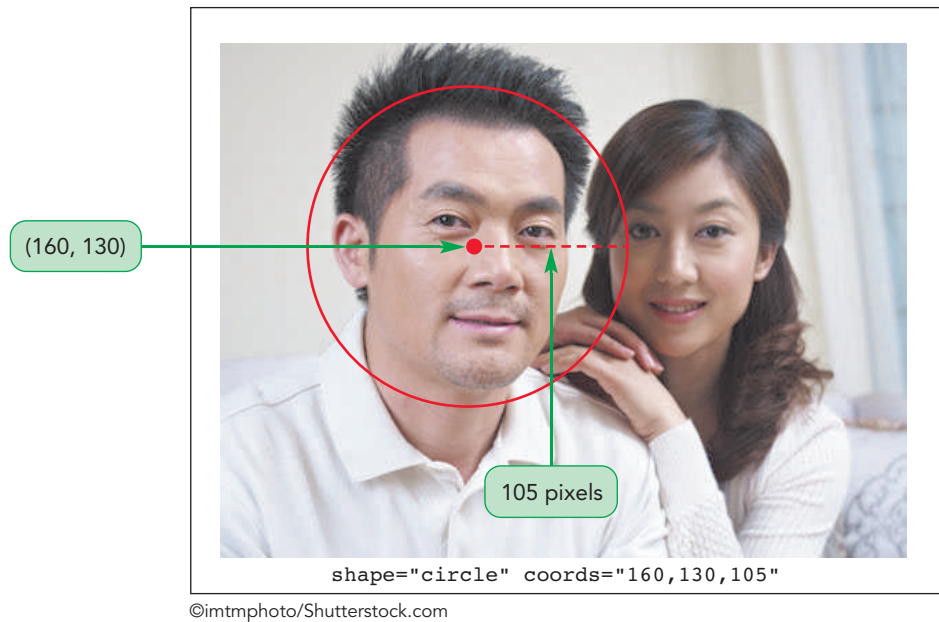
Circular hotspots are defined using the attributes

```
shape="circle" coords="x,y,radius"
```

where *x* and *y* are the coordinates of the center of the circle and *radius* is the circle's radius. Figure 4–60 shows the coordinates for a circular hotspot where the center of the circle is located at the coordinates (160, 130) with a radius of 105 pixels.



Figure 4–60 Defining a circular hotspot

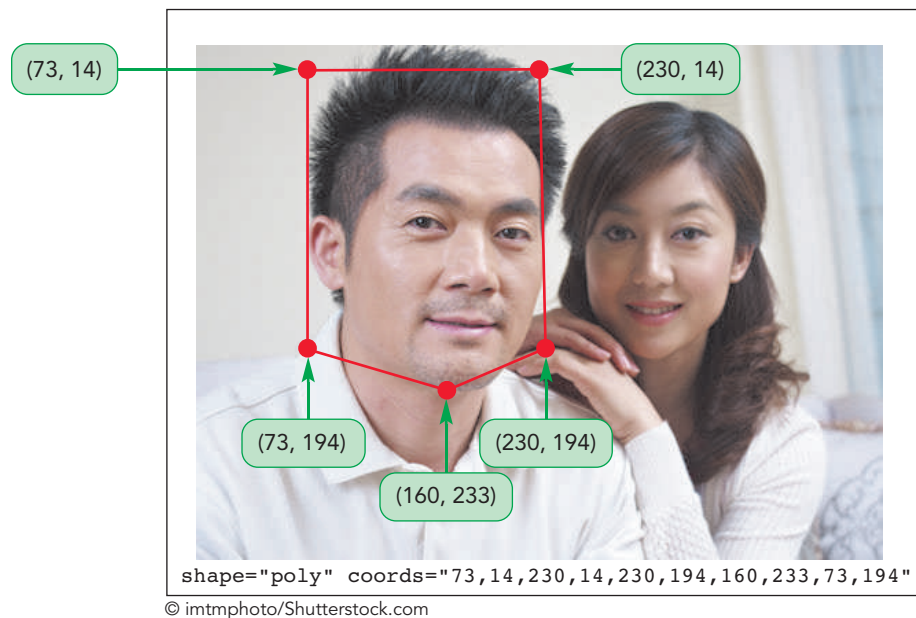


Polygonal hotspots have the attributes

```
shape="poly" coords="x1, y1, x2, y2, ..."
```

where  $(x1, y1)$ ,  $(x2, y2)$ , ... set the coordinates of each vertex in the shape. Figure 4–61 shows the coordinates for a 5-sided polygon.

Figure 4–61 Defining a polygonal hotspot

**TIP**

Default hotspots should always be listed last.

To define the default hotspot for an image, create the following hotspot:

```
shape="default" coords="0,0,width,height"
```

where *width* is the width of the image in pixels and *height* is the image's height. Any region in the image that is not covered by another hotspot activates the default hotspot link.

### Creating an Image Map

- To create an image map, use

```
<map name="text">
  hotspots
</map>
```

where *text* is the name of the image map and *hotspots* are the hotspots within the image.

- To define each hotspot, use

```
<area shape="shape" coords="coordinates" href="url" alt="text" />
```

where *shape* is the shape of the hotspot region, *coordinates* list the points defining the boundaries of the region, *url* is the URL of the hypertext link, and *text* is alternate text that is displayed for non-graphical browsers.

- To define a rectangular hotspot, use the *shape* and *attribute* values

```
shape="rect" coords="left,top,right,bottom"
```

where *left*, *top* are the coordinates of the top-left corner of the rectangle and *right*, *bottom* are the coordinates of the bottom-right corner.

- To define a circular hotspot, use

```
shape="circle" coords="x,y,radius"
```

where *x* and *y* are the coordinates of the center of the circle and *radius* is the circle's radius.

- To define a polygonal hotspot, use

```
shape="poly" coords="x1,y1,x2,y2,..."
```

where (*x1*, *y1*), (*x2*, *y2*), and so on provide the coordinates of each vertex in the multisided shape.

- To define the default hotspot link, use

```
shape="default" coords="0,0,width,height"
```

where *width* and *height* is the width and height of the image.

Kevin has provided you with the coordinates for five rectangular hotspots to cover the five faces on the Komatsu family portrait. Add an image map named "family\_map" to the `tb_komatsu.html` page with rectangular hotspots for each of the faces in the family portrait.

#### To create an image map:

1. Open or return to the `tb_komatsu.html` file in your editor.
2. Directly below the figure box, insert the following HTML code:

```
<map name="family_map">
  <area shape="rect" coords="74,74,123,141"
    href="tb_ikko.html" alt="Ikko Komatsu" />
  <area shape="rect" coords="126,109,177,172"
    href="tb_mika.html" alt="Mika Komatsu" />
  <area shape="rect" coords="180,157,230,214"
    href="tb_hiroji.html" alt="Hiroji Komatsu" />
```

```

<area shape="rect" coords="258,96,312,165"
      href="tb_genta.html" alt="Genta Komatsu" />
<area shape="rect" coords="342,86,398,162"
      href="tb_suzuko.html" alt="Suzuko Komatsu" />
</map>

```

Figure 4–62 highlights the HTML code for the image map and hotspots.

Figure 4–62

### Inserting an image map

```

<figure>
  
  <figcaption>(L-R): Ikko, Mika, Hiroji, Genta, Suzuko</figcaption>
</figure>
<map name="family_map">
  <area shape="rect" coords="74,74,123,141" href="tb_ikko.html" alt="Ikko Komatsu" />
  <area shape="rect" coords="126,109,177,172" href="tb_mika.html" alt="Mika Komatsu" />
  <area shape="rect" coords="180,157,230,214" href="tb_hiroji.html" alt="Hiroji Komatsu" />
  <area shape="rect" coords="258,96,312,165" href="tb_genta.html" alt="Genta Komatsu" />
  <area shape="rect" coords="342,86,398,162" href="tb_suzuko.html" alt="Suzuko Komatsu" />
</map>

```

Annotations in the diagram:

- name of the image map (points to `name="family_map"`)
- shape of the hotspot (points to `shape="rect"`)
- coordinates of the rectangular hotspot (points to `coords="x1,y1,x2,y2"`)
- URL of the hotspot link (points to `href="url"`)
- alternate text for the hotspot (points to `alt="text"`)

3. Save your changes to the file.

With the image map defined, your next task is to apply that map to the image in the figure box.

## Applying an Image Map

To apply an image map to an image, you add the following `usemap` attribute to the `img` element

```

```

where `map` is the name assigned to the image map within the current HTML file.

### REFERENCE

#### Applying an Image Map

- To apply an image map to an image, add the `usemap` attribute to the `img` element

```

```

where `map` is the name assigned to the image map.

Apply the `family_map` image map to the figure box and then test it in your web browser.

### To apply an image map:

1. Add the attribute `usemap="#family_map"` to the `img` element for the family portrait.

Figure 4–63 highlights the code to apply the image map.

Figure 4–63

### Applying an image map

Applies the family\_map image map to the image

```
<figure>
  
  <figcaption>(L-R): Ikko, Mika, Hiroji, Genta, Suzuko</figcaption>
</figure>
```

2. Save your changes to the file and then reload `tb_komatsu.html` in your browser.
3. Click the five faces in the family portrait and verify each face is linked to a separate HTML file devoted to that individual. Use the link under the image of each individual to return to the home page.

Kevin likes the addition of the image map and plans to use it on other photos in the website.



PROSKILLS

### Problem Solving: Image Maps with Flexible Layouts

Image maps are not easily applied to flexible layouts in which the size of the image can change based on the size of the browser window. The problem is that, because hotspot coordinates are expressed in pixels, they don't resize and will not point to the correct region of the image if the image is resized.

One way to deal with flexible layouts is to create hotspots using hypertext links that are sized and positioned using relative units. The image and the hypertext links would then be nested within a `figure` element as follows:

```
<figure class="map">
  
  <a href="url" id="hotspot1"></a>
  <a href="url" id="hotspot2"></a>
  ...
</figure>
```

The figure box itself needs to be placed using relative or absolute positioning and the image should occupy the entire figure box. Each hypertext link should be displayed as a block with width and height defined using percentages instead of pixels and positioned absolutely within the figure box, also using percentages for the coordinates. As the figure box is resized under the flexible layout, the hotspots marked with the hypertext links will automatically be resized and moved to match. The opacity of the hotspot links should be set to 0 so that the links do not obscure the underlying image file. Even though the hotspots will be transparent to the user, they will still act as hypertext links.

This approach is limited to rectangular hotspots. To create a flexible layout for other shapes, you need to use a third-party add-in that automatically resizes the shape based on the current size of the image.

You've completed your work on the Komatsu Family pages for *Tree and Book*. Kevin will incorporate your work and ideas with other family pages as he continues on the site redesign. He'll get back to you with more projects in the future. For now you can close any open files or applications.

## REVIEW

### Session 4.3 Quick Check

1. Provide the transformation to shift a page object 5 pixels to the right and 10 pixels up.
  - a. `transform: translate(5px, 10px);`
  - b. `transform: translate(5px, -10px);`
  - c. `transform: translate(-5px, -10px);`
  - d. `translate: -5px 10px;`
2. Provide the transformation to reduce the horizontal and vertical size of an object by 50%.
  - a. `scale: 0.5, 0.5;`
  - b. `transform: scale(0.5, 0.5);`
  - c. `transform: rescale(0.5, 0.5);`
  - d. `transform: resize(0.5, 0.5);`
3. Provide the transformation to rotate an object 30° counter-clockwise around the x-axis.
  - a. `transform: rotate(-30deg);`
  - b. `transform: rotate(30deg);`
  - c. `transform: rotateX(30deg);`
  - d. `transform: rotateX(-30deg);`
4. Provide the filter to increase the brightness of an object by 20%.
  - a. `filter: brightness(20%);`
  - b. `filter: brightness(0.2);`
  - c. `filter: brightness(1.2);`
  - d. `brightness: 0.2;`
5. Provide the filter to decrease the contrast of an object by 30%.
  - a. `filter: contrast(0.3);`
  - b. `filter: contrast(0.7);`
  - c. `filter: contrast(30%);`
  - d. `filter: contrast(-0.3);`
6. Provide the code to create a triangular hotspot with vertices at (200, 5), (300, 125), and (100, 125), linked to the info.html file.
  - a. `<area type="poly" coords="200, 5, 300, 125, 100, 125" href="info.html" />`
  - b. `<area type="triangle" coords="200, 5, 300, 125, 100, 125" href="info.html" />`
  - c. `<area type="draw" coords="200, 5, 300, 125, 100, 125" href="info.html" />`
  - d. `<area type="poly" coords="5, 200, 125, 300, 125, 100" href="info.html" />`
7. Provide code to attach the logo.png image to the mapsites image map:
  - a. ``
  - b. ``
  - c. ``
  - d. ``
8. Provide a style rule to transfer 3D space from a container element to its nested elements.
  - a. `transform-style: perspective;`
  - b. `preserve-3d: true;`
  - c. `transfer-style: preserve-3d;`
  - d. `transform-style: preserve-3d;`

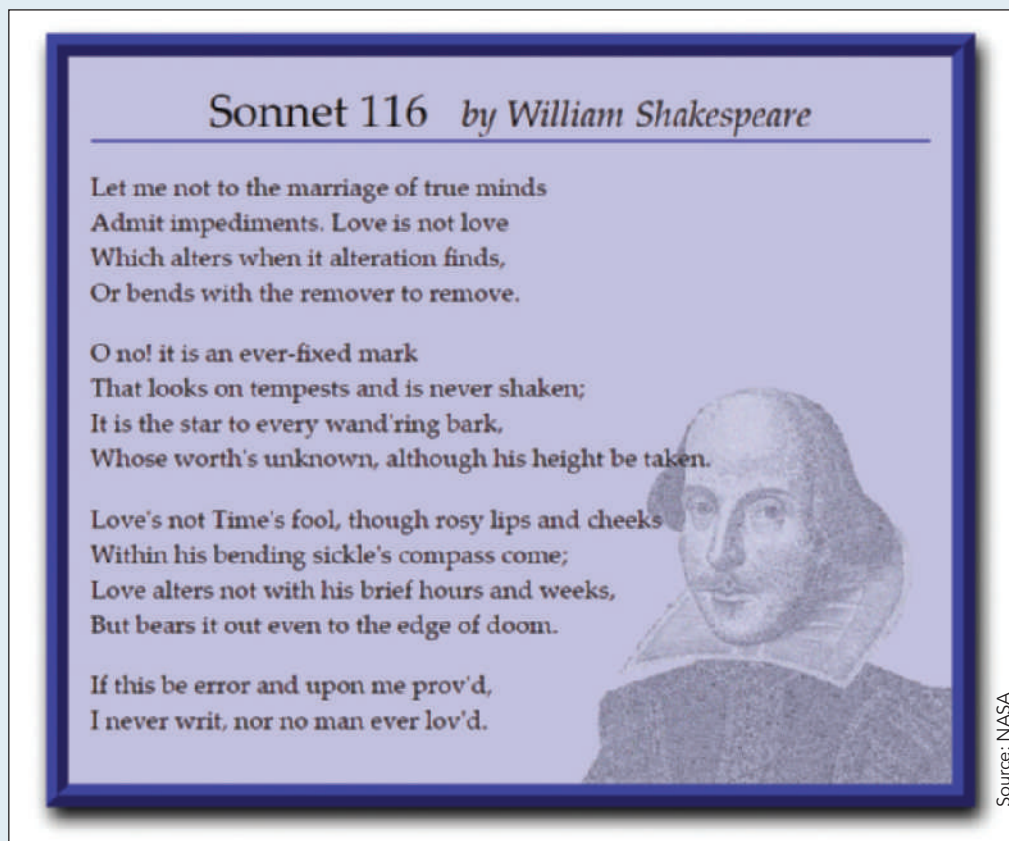
## Coding Challenge 1

Data Files needed for this Coding Challenge: `code4-1_txt.html`, `code4-1_back_txt.css`, `code4-1.css`, `ws.png`

Figure 4–64 shows a web page containing text from a Shakespearean sonnet. In this Coding Challenge you will augment the text of the poem with background colors and images and add a graphic border.

Figure 4–64

Coding Challenge 4-1 example page



Do the following:

1. Open the `code4-1_txt.html` and `code4-1_back_txt.css` files from the `html04 ▶ code1` folder. Enter **your name** and **the date** in each document and save the files as `code4-1.html` and `code4-1_back.css` respectively.
2. Go to the `code4-1.html` file in your editor. Within the head section insert a link element linking the page to the `code4-1_back.css` style sheet file.
3. Enclose the content of the sonnet within a `figure` element. At the top of the `figure` element insert a figure caption containing the HTML code `Sonnet 116 <cite>by William Shakespeare</cite>`.
4. Save your changes to the file and return to the `code4-1_back.css` file in your editor.
5. Create a style rule for the `figure` element that:
  - a. Sets the padding space to 20 pixels.
  - b. Adds a 20-pixel border in the ridge style with the color value `rgb(52, 52, 180)`.
  - c. Has a background consisting of the image file `ws.png` placed in the bottom right corner of the figure box and set to 45% of the width of the figure box with no tiling. Be sure to separate the



- position of the image and its size with the / character. Add a second background containing the color `rgba(52, 52, 180, 0.3)`. Enter both background properties within a single style rule separated by a comma.
- d. Has a black box shadow that is 5 pixels to the right, 10 pixels down with a blur size of 15 pixels.
  6. Create a style rule for the figure caption that:
    - a. Sets the font size to 1.8em.
    - b. Centers the text of the caption.
    - c. Adds a 2-pixel bottom solid bottom border of the color value `rgb(52, 52, 180)`.
  7. Save your changes to the style sheet.
  8. Open the page in your browser and verify that the design resembles that shown in Figure 4–64.
  9. Submit the completed file to your instructor.

## CODE

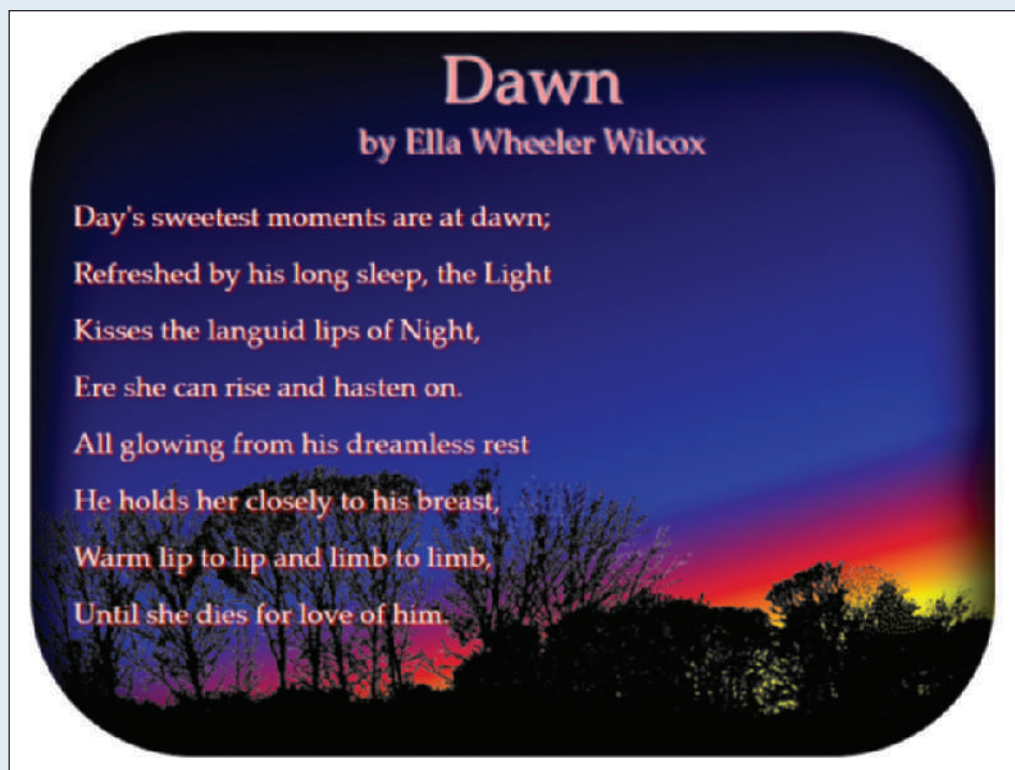
## Coding Challenge 2

Data Files needed for this Coding Challenge: `code4-2_txt.html`, `code4-2_grad_txt.css`, `code4-2.css`, `landscape.png`

Figure 4–65 shows a web page containing text of a poem by Ella Wheeler Wilcox entitled “Dawn.” To augment the poem, a background image containing a linear gradient has been added to the web page. In addition, text shadows have been added to bring the text of the poem out of the page.

Figure 4–65

Coding Challenge 4-2 example page





Complete the following:

1. Open the `code4-2_txt.html` and `code4-2_grad_txt.css` files from the `html04 ▶ code2` folder. Enter *your name* and *the date* in each document and save the files as `code4-2.html` and `code4-2_grads.css` respectively.
2. Go to the `code4-2.html` file in your editor. Within the head section insert a `link` element linking the page to the `code4-2_grad.css` file. Save your changes to the file.
3. Go to the `code4-2_grad.css` file. Create a style rule for `h1` and `h2` elements that adds a white text shadow 2 pixels above and to the left of the text with a blur radius of 3 pixels.
4. Create a style rule for paragraphs that adds a red text shadow 2 pixels down and to the right of the text with a blur radius of 3 pixels.
5. Create a style rule for the `article` element that adds a black inset box shadow with a horizontal and vertical offset of 0 pixels, a blur radius of 50 pixels and a size of 20 pixels.
6. Create a style rule for the `article` element that sets the radius of the border corners to 150 pixels.
7. Create a style rule for the `article` element that adds the following multiple backgrounds:
  - a. A background containing the image file `landscape.png` placed with no tiling at the bottom right corner of the element with a size of 100%.
  - b. A linear gradient at an angle of 165 degrees that goes from black to the color value `rgb(0, 0, 200)` with a color stop of 65%, to `rgb(211, 0, 55)` with a color stop of 75%, to orange with a color stop of 80%, and finally to yellow with a color stop of 82%.
8. Save your changes to the style sheet.
9. Open the page in your browser and verify that the design resembles that shown in Figure 4–65.
10. Submit the completed file to your instructor.

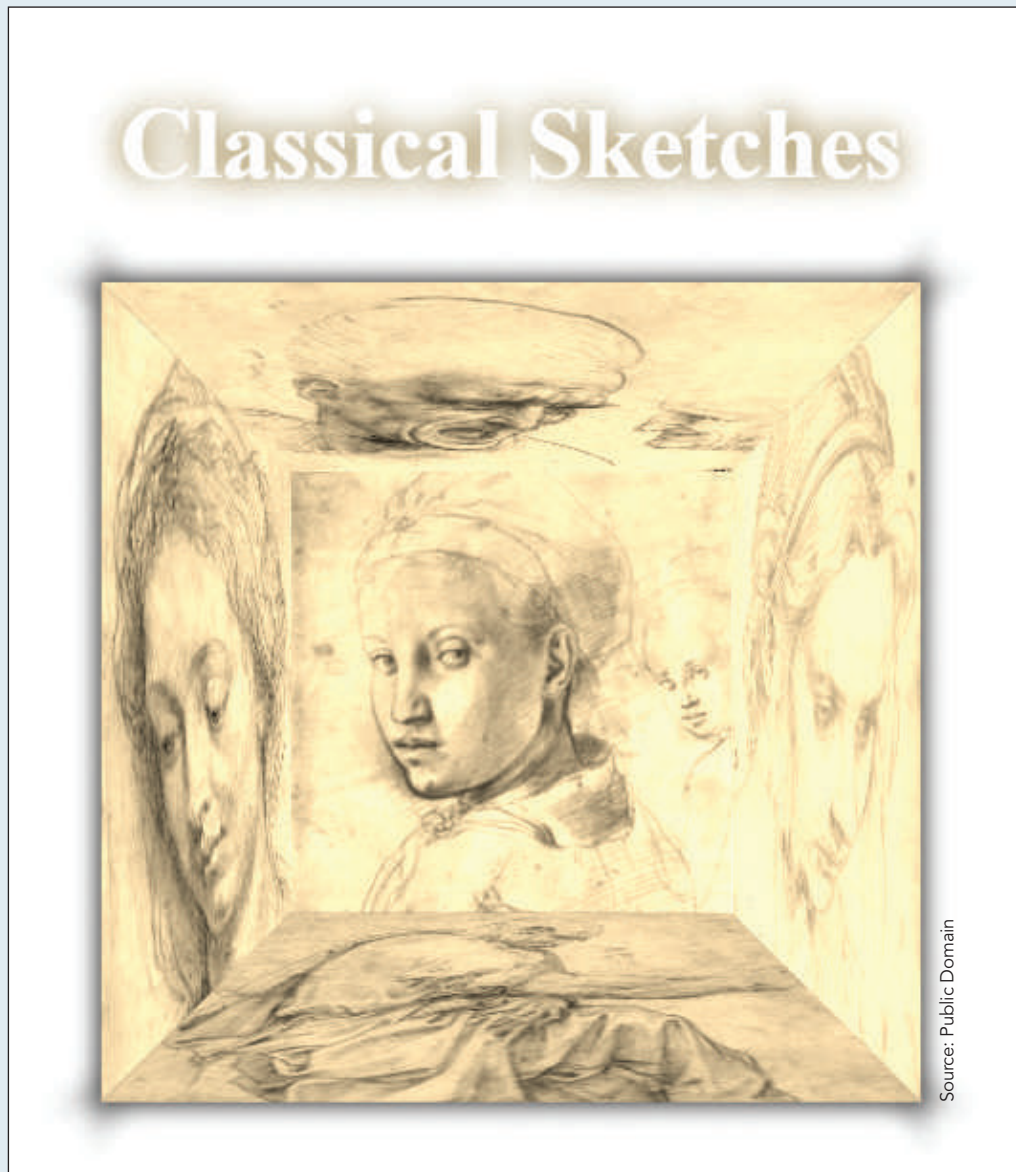
### Coding Challenge 3

Data Files needed for this Coding Challenge: `code4-3_txt.html`, `code4-3_cube_txt.css`, `code4-3.css`, `image01.png` - `image05.png`

Figure 4–66 shows a web page in which five faces of the cube are displayed in a 3D view. You can create this effect using the CSS 3D transformation styles. The page also contains CSS styles for box shadows and text shadows that you will have to add.

Figure 4–66

Coding Challenge 4-3 example page



Complete the following to create the web page:

1. Open the **code4-3\_txt.html** and **code4-3\_cube\_txt.css** files from the html04 ► code3 folder. Enter **your name** and **the date** in each document and save the files as **code4-3.html** and **code4-3\_cube.css** respectively.
2. Go to the **code4-3.html** file in your editor. Within the head section insert a `link` element that links the page to the `code4-3_cube.css` style sheet file. Note that within the web page the five images are contained with a `div` element with the ID value "cube". The images are given ID values of `img1` through `img5`. Save your changes to the file.
3. Go to the **code4-3\_cube.css** file in your editor. Create a style rule for the `h1` element that changes the font color to white and adds a text shadow with horizontal and vertical offsets of 0 pixels, a blur radius of 20 pixels and a shadow color value of `rgb(120, 85, 0)`.
4. Create a style rule for a `div` element with the id "cube" that sets the perspective size of the 3D space to 500 pixels. Use the `transform-style` property to preserve the 3D setting for the children of this element so that the cube and its children exist in the same 3D space.

5. For all `img` elements create a style rule that applies a sepia filter with a value of 1. Add a black box shadow with horizontal and vertical offsets of 0 pixels and a blur radius of 20 pixels.
6. Create the following style rules for the five image elements:
  - a. For the `img1` image, translate the image -150 pixels along the z-axis.
  - b. For the `img2` image, rotate the image 90 degrees around the x-axis and translate the image -150 pixels along the z-axis.
  - c. For the `img3` image, rotate the image -90 degrees around the y-axis and translate the image 150 pixels along the z-axis.
  - d. For the `img4` image, rotate the image 90 degrees around the y-axis and translate the image 150 pixels along the z-axis.
  - e. For the `img5` image, rotate the image -90 degrees around the x-axis and translate the image -150 pixels along the z-axis.
7. Save your changes to the style sheet.
8. Open the page in your browser and verify that the design resembles that shown in Figure 4–66.
9. Submit the completed file to your instructor.

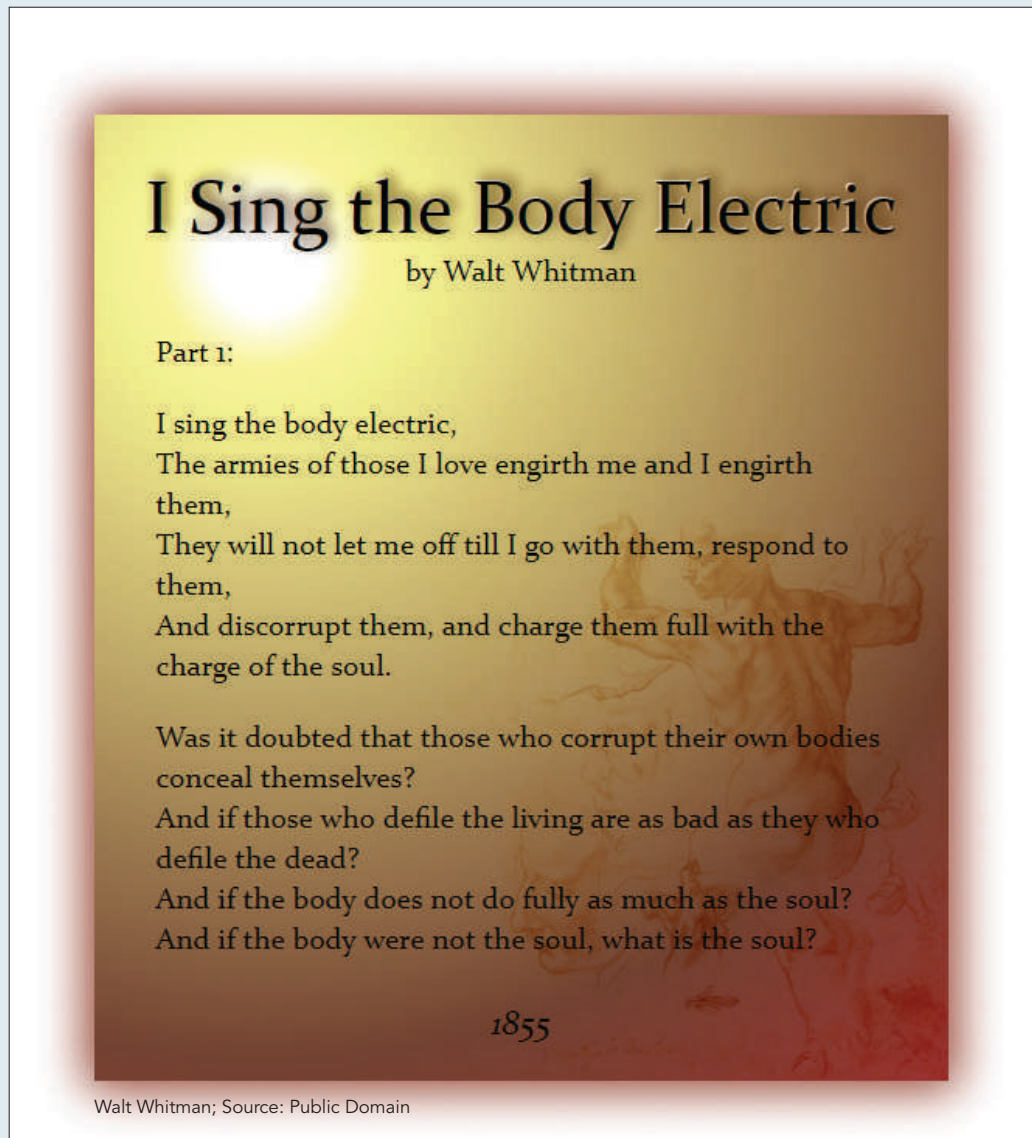
### Coding Challenge 4

**Data Files needed for this Coding Challenge:** `code4-4_txt.html`, `debug4-4_txt.css`, `code4-4_.css`, `Michelangelo.png`

Figure 4–67 shows a completed web page that uses CSS design elements to enhance the appearance of a poem by Walt Whitman. You’ve been given a copy of the files for this web page, but there are several syntax errors in the CSS stylesheet. Use your knowledge of CSS to fix the stylesheet code and complete the page.

Figure 4–67

Coding Challenge 4-4 example page



Do the following:

1. Open the **code4-4\_txt.html** and **debug4-4\_txt.css** files from the `html04 > code4` folder. Enter **your name** and **the date** in each document and save the files as **code4-4.html** and **debug4-4.css** respectively.
2. Go to the **code4-4.html** file in your editor. Within the head section insert a `link` element that links the page to the `code4-4_debug.css` style sheet file. Study the contents of the file and then save your changes.
3. Go to the **debug4-4.css** file in your browser. The first style rule adds two text shadows to the `h1` heading: a dark brown shadow and a white highlight. The shadows are not appearing in the web page. Locate and fix the syntax error in this style rule.
4. The next style rule was written to add a box shadow to the `article` element that has an offset of 0 pixels in the horizontal and vertical directions, blur radius of 30 pixels and size value of 5 pixels. Fix the syntax errors in this style rule.
5. The next style rule creates a border image for the `article` element using a linear gradient for the image. Fix the syntax error in this style rule.
6. The final style rule defines the background for the `article` element consisting of:
  - a. A radial gradient going from white circle located near the top left corner of the background, to semi-transparent yellow, semi-transparent brown, and semi-transparent ochre,

- b. A sketch by Michelangelo located in the lower right corner sized at 75% of the width of the element, and
  - c. An ivory-colored background fill. There are several syntax errors in the code. Locate and fix all of the errors.
7. Save your changes and open the **code4-4.html** file in your browser. Verify that design of the page resembles that shown in Figure 4–67.
  8. Submit the completed file to your instructor.

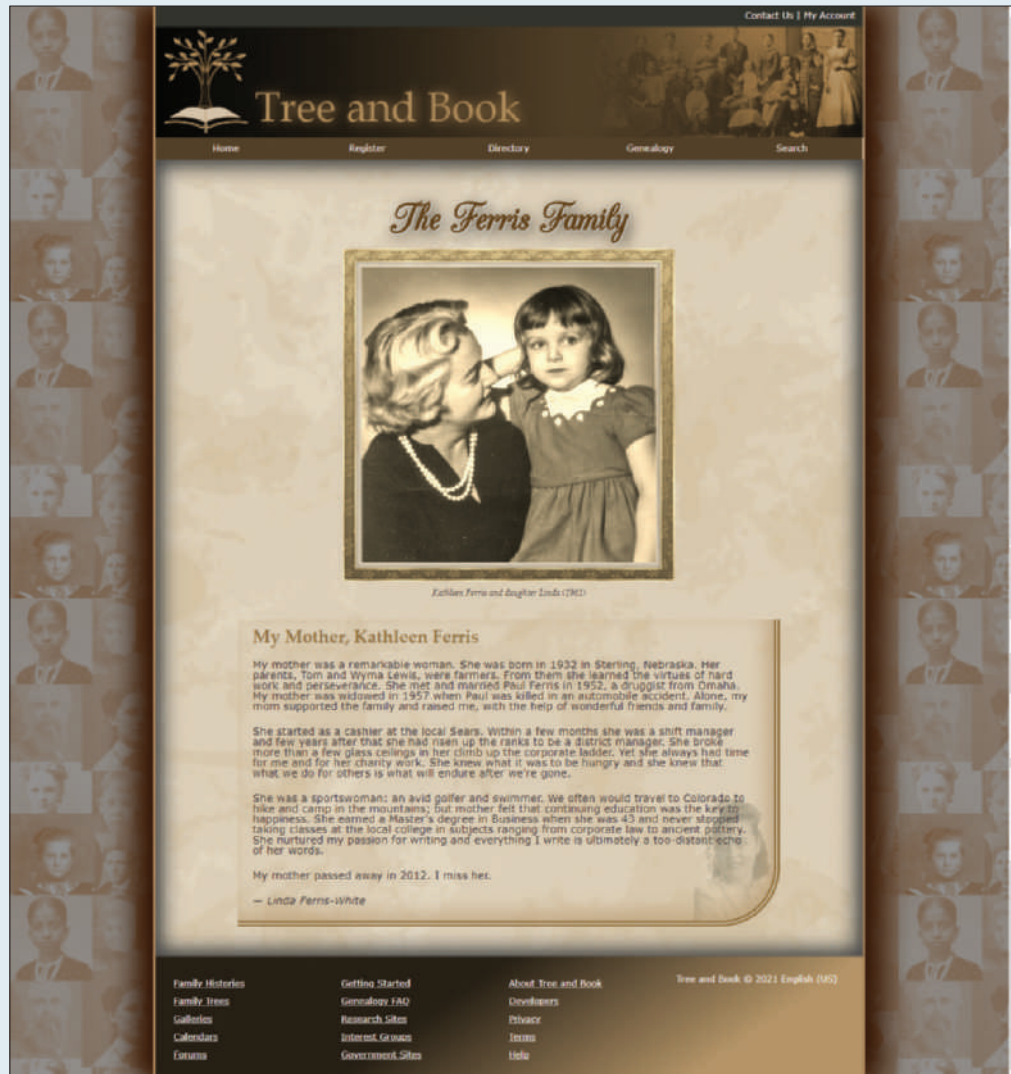
## Review Assignments

**Data Files needed for the Review Assignments: tb\_ferris\_txt.html, tb\_kathleen\_txt.html, tb\_visual3\_txt.css, tb\_visual4\_txt.css, 3 CSS files, 1 HTML file, 10 PNG files, 1 TTF file, 1 WOFF file**

Kevin wants you to work on another family page for the Tree and Book website. The page was created for the Ferris family with content provided by Linda Ferris-White. Kevin is examining a new color scheme and design style for the page. A preview of the design you'll create is shown in Figure 4–68.

Figure 4–68

Ferris Family page



Source: Design Studio Pro; Source: Wikimedia Commons;  
© Elzbieta Sekowska/Shutterstock.com



All of the HTML content and the typographical and layout styles have already been created for you. Your task will be to complete the work by writing the visual style sheet to incorporate Kevin's suggestions.

Complete the following:

1. Use your HTML editor to open the **tb\_visual3\_txt.css**, **tb\_visual4\_txt.css**, **tb\_ferris\_txt.html** and **tb\_kathleen\_txt.html** files from the html04 ► review folder. Enter *your name* and *the date* in the comment section of each file, and save them as **tb\_visual3.css**, **tb\_visual4.css**, **tb\_ferris.html**, and **tb\_kathleen.html** respectively.
2. Go to the **tb\_ferris.html** file in your editor. Add links to the **tb\_base.css**, **tb\_styles3.css**, and **tb\_visual3.css** style sheets in the order listed.
3. Scroll down and, within the `main` element header and after the `h1` heading, insert a figure box containing: a) the **tb\_ferris.png** inline image with the alternate text *Ferris Family* using the image map named *portrait\_map* and b) a figure caption with the text *Kathleen Ferris and daughter Linda (1961)*.
4. Directly below the figure box, create the *portrait\_map* image map containing the following hotspots: a) a rectangular hotspot pointing to the **tb\_kathleen.html** file with the left-top coordinate (10, 50) and the right-bottom coordinate (192, 223) and alternate text, "Kathleen Ferris" and b) a circular hotspot pointing to the **tb\_linda.html** file with a center point at (264, 108) and a radius of 80 pixels and the alternate text, *Linda Ferris-White*.
5. Take some time to study the rest of the page content and structure and then save your changes to the file.
6. Go to the **tb\_visual3.css** file in your editor. In this file, you'll create the graphic design styles for the page.
7. Go to the HTML Styles section and create a style rule for the `html` element to use the image file **tb\_back5.png** as the background.
8. Go to the Page Body Styles section and create a style rule for the `body` element that: a) adds a left and right 3-pixel solid border with color value `rgb(169, 130, 88)`, b) adds a box shadow to the right border with a horizontal offset of 25 pixels, a vertical offset of 0 pixels and a 35-pixel blur and a color value of `rgb(53, 21, 0)`, and then adds the mirror images of this shadow to the left border.
9. Go to the Main Styles section. Create a style rule for the `main` element that: a) applies the **tb\_back7.png** file as a background image with a size of 100% covering the entire background with no tiling and positioned with respect to the padding box and b) adds two inset box shadows, each with a 25-pixel blur and a color value of `rgb(71, 71, 71)`, and then one with offsets of -10 pixels in the horizontal and vertical direction and the other with horizontal and vertical offsets of 10 pixels.
10. Create a style rule for the `h1` heading within the main header that adds the following two text shadows: a) a shadow with the color value `rgb(221, 221, 221)` and offsets of 1 pixels and no blurring and b) a shadow with the color value `rgba(41, 41, 41, 0.9)` and offsets of 5 pixels and a 20-pixel blur.
11. Go to the Figure Box Styles section. Create a style rule for the `figure` element that sets the top/bottom margin to 10 pixels and the left/right margin to `auto`. Set the width of the element to 70%.
12. Next, you'll modify the appearance of the figure box image. Create a style rule for the image within the figure box that: a) sets the border width to 25 pixels, b) sets the border style to solid, c) applies the **tb\_frame.png** file as a border image with a slice size of 60 pixels stretched across the sides, d) displays the image as a block with a width of 100%, and e) applies a sepia tone to the image with a value of 80% (include the WebKit browser extension in your style sheet).
13. Create a style rule for the figure caption that: a) displays the text using the font stack 'Palatino Linotype', Palatino, 'Times New Roman', serif, b) sets the style to italic, c) sets the top/bottom padding to 10 pixels and the left/right padding to 0 pixels, and d) centers the text.
14. Go to the Article Styles section. Here you'll create borders and backgrounds for the article that Linda Ferris-White wrote about her mother. Create a style rule for the `article` element that: a) displays the background image file **tb\_back6.png** placed at the bottom-right corner of the element with a size of 15% and no tiling, b) adds an 8-pixel double border with color value `rgb(147, 116, 68)` to the right and bottom sides of the `article` element, c) creates a curved bottom-right corner with a

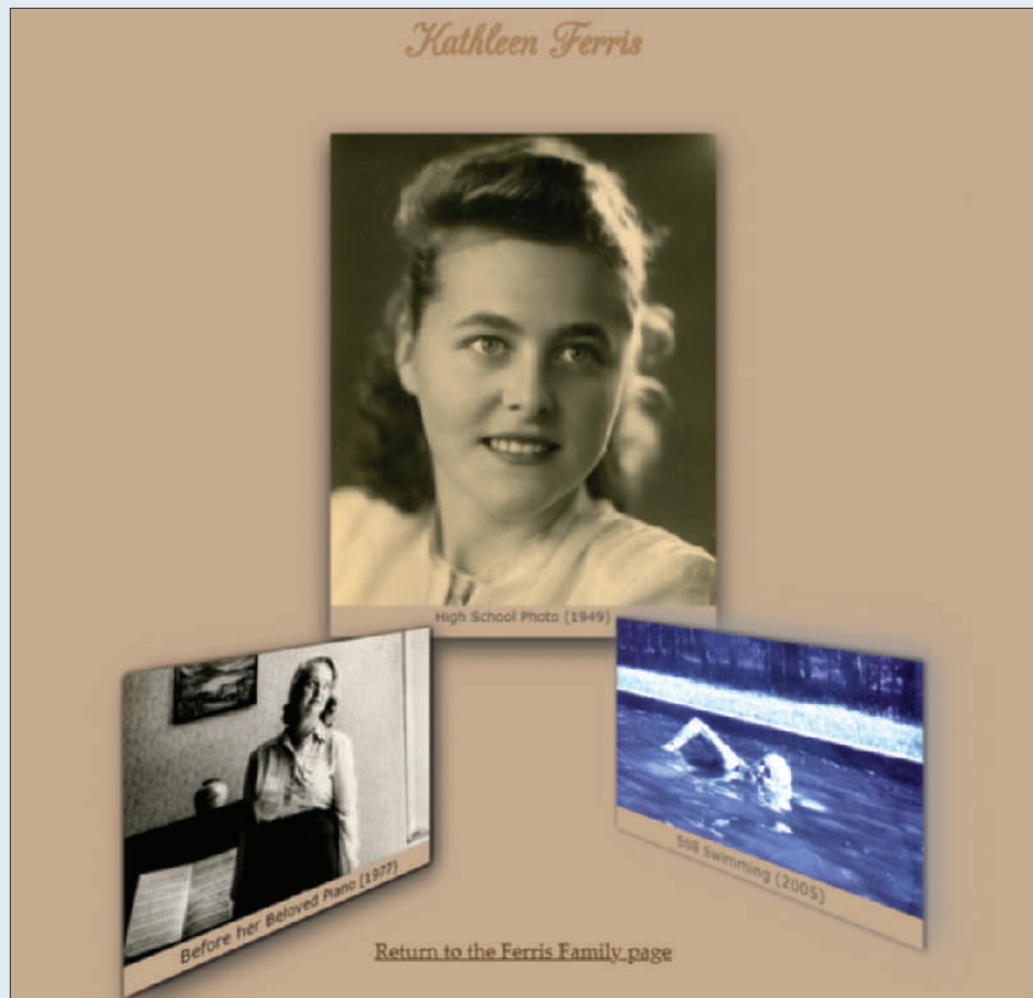
radius of 80 pixels, and d) adds an interior shadow with horizontal and vertical offsets of  $-10$  pixels, a 25-pixel blur, and a color value of `rgba(184, 154, 112, 0.7)`.

15. Kevin wants a gradient background for the page footer. Go to the Footer Styles section and create a style rule for the footer that adds a linear gradient background with an angle of  $325^\circ$ , going from the color value `rgb(180, 148, 104)` with a color stop at 20% of the gradient length to the value `rgb(40, 33, 23)` with a color stop at 60%.
16. Save your changes to the style sheet and then open `tb_ferris.html` in your browser. Verify that the colors and designs resemble that shown in Figure 4–68.

Next, you will create the design styles for individual pages about Kathleen Ferris and Linda Ferris-White. A preview of the content of the Kathleen Ferris page is shown in Figure 4–69.

Figure 4–69

Kathleen Ferris page



© Elzbieta Sekowska/Shutterstock.com

17. Go to the `tb_kathleen.html` file in your editor and create links to the `tb_base.css`, `tb_styles4.css`, and `tb_visual4.css` files. Study the contents of the file and then close it, saving your changes.
18. Go to the `tb_visual4.css` file in your editor. Scroll down to the Transformation Styles section and add a style rule for the `article` element to set the size of the perspective space to 800 pixels.
19. Create a style rule for the `figure1` figure box to translate it  $-120$  pixels along the  $z$ -axis.
20. Create a style rule for the `figure2` figure box to translate it  $-20$  pixels along the  $y$ -axis and rotate it  $50^\circ$  around the  $y$ -axis.
21. Create a style rule for the `figure3` figure box to translate it  $-30$  pixels along the  $y$ -axis and rotate it  $-50^\circ$  around the  $y$ -axis.



22. Go to the Filter Styles section to apply CSS filters to the page elements. Create a style rule for the figure1 figure box that applies a saturation filter with a value of 1.3.
23. Create a style rule for the figure2 figure box that sets the brightness to 0.8 and the contrast to 1.5.
24. Create a style rule for the figure3 figure box that sets the hue rotation to 170°, the saturation to 3, and the brightness to 1.5.
25. Save your changes to the file and then return to the **tb\_ferris.html** file in your browser. Verify that you can display the individual pages for Kathleen Ferris and Linda Ferris-White by clicking on their faces in the family portrait. Further verify that the appearance of the Kathleen Ferris page resembles that shown in Figure 4–69. (Note: Use the link under the pictures to return to the home page.)

## Case Problem 1

**Data Files needed for this Case Problem:** `sf_torte_txt.html`, `sf_effects_txt.css`, 2 CSS files, 9 PNG files

**Save your Fork** Amy Wu has asked for your help in redesigning her website, *Save your Fork*, a baking site for people who want to share dessert recipes and learn about baking in general. She has prepared a page containing a sample dessert recipe and links to other pages on the website. A preview of the page you'll create is shown in Figure 4–70.

Figure 4–70 Save your Fork sample recipe page

The screenshot shows the 'Save your Fork' website interface. At the top, there's a teal header with the site logo and navigation links: HOME, MY ACCOUNT, SEARCH. Below the header is a large image of various desserts. A teal navigation bar contains links for RECIPES, MARKS, HOLIDAYS, MARKET, SHOPPING, COLUMN, and MESSAGE. On the left, there's a 'Categories' sidebar with a list of dessert types. The main content area features a recipe for 'Apple Bavarian Torte' by Lemking, including a photo of the dish, a description, ingredients, and directions. To the right of the recipe box is a 'Recipe Box' with several action buttons. Further right are three user reviews with star ratings and profile pictures. At the bottom of the page, there's a 'Directions' section with a numbered list of steps.

Amy has already created a style sheet for the page layout and typography, so your work will be focused on enhancing the page with graphic design styles.

Complete the following:

1. Using your editor, open the `sf_torte_txt.html` and `sf_effects_txt.css` files from the `html04 ▶ case1` folder. Enter *your name* and *the date* in the comment section of each file, and save them as `sf_torte.html` and `sf_effects.css` respectively.
2. Go to the `sf_torte.html` file in your editor. Within the document head create links to the `sf_base.css`, `sf_layout.css`, and `sf_effects.css` style sheet files in that order. Take some time to study the structure of the document and then close the document, saving your changes.
3. Go to the `sf_effects.css` file in your editor. Within the Body Header Styles section, create a style rule for the `body` element to add drop shadows to the left and right border of the page body with an offset of 10 pixels, a blur of 50 pixels, and the color `rgb(51, 51, 51)`. Note that the right border is a mirror image of the left border.
4. Go to the Navigation Tabs List Styles section. Amy has created a navigation list with the class name `tabs` that appears at the top of the page with the body header. Create a style rule for the `body > header nav.tabs` selector that changes the background to the image file `sf_back1.png` with no tiling, centered horizontally and vertically within the element and sized to cover the entire navigation list.
5. Amy wants the individual list items in the tabs navigation list to appear as tabs in a recipe box. She wants each of these “tabs” to be trapezoidal in shape. To create this effect, you’ll create a style rule for the `body > header nav.tabs li` selector that transforms the list item by setting the perspective of its 3D space to 50 pixels and rotating it 20° around the x-axis.
6. As users hover the mouse pointer over the navigation tabs, Amy wants a rollover effect in which the tabs appear to come to the front. Create a style rule for the `body > header nav.tabs li` selector that uses the pseudo-element `hover` that changes the background color to `rgb(231, 231, 231)`.
7. Go to the Left Section Styles section. Referring to Figure 4–70, notice that in the left section of the page, Amy has placed two vertical navigation lists. She wants these navigation lists to have rounded borders. For the vertical navigation lists in the left section, create a style rule for the `section#left nav.vertical` selector that adds a 1-pixel solid border with color value `rgb(20, 167, 170)` and has a radius of 25 pixels at each corner.
8. The rounded corner also has to apply to the `h1` heading within each navigation list. Create a style rule for `h1` elements nested within the left section vertical navigation list that sets the top-left and top-right corner radii to 25 pixels.
9. Go to the Center Article Styles section. The `article` element contains an image and brief description of the Apple Bavarian Torte, which is the subject of this sample page. Create a style rule for the `section#center article` selector that adds the following: a) a radial gradient to the background with a white center with a color stop of 30% transitioning to `rgb(151, 151, 151)`, b) a 1-pixel solid border with color value `rgb(151, 151, 151)` and a radius of 50 pixels, and c) a box shadow with horizontal and vertical offsets of 10 pixels with a 20-pixel blur and a color of `rgb(51, 51, 51)`.
10. Go to the Blockquote Styles section. Amy has included three sample reviews from users of the Save your Fork website. Amy wants the text of these reviews to appear within the image of a speech bubble. For every `blockquote` element, create a style rule that does the following: a) sets the background image to the `sf_speech.png` file with no tiling and a horizontal and vertical size of 100% to cover the entire block quote, and b) uses the `drop-shadow` filter to add a drop shadow around the speech bubble with horizontal and vertical offsets of 5 pixels, a blur of 10 pixels and the color `rgb(51, 51, 51)`.
11. Amy has included the photo of each reviewer registered on the site within the citation for each review. She wants these images to appear as circles rather than squares. To do this, create a style rule for the selector `cite img` that sets the border radius to 50%.
12. Save your changes to the style sheet file and then open `sf_torte.html` in your browser. Verify that the design of your page matches that shown in Figure 4–70. Confirm that when you hover the mouse over the navigation tabs the background color changes to match the page color.

## Case Problem 2

Data Files needed for this Case Problem: [cf\\_home\\_txt.html](#), [cf\\_effects\\_txt.css](#), 2 CSS files, 7 PNG files

**Chupacabra Music Festival** Debra Kelly is the director of the website for the *Chupacabra Music Festival*, which takes place every summer in San Antonio, Texas. Work is already underway on the website design for the 15<sup>th</sup> annual festival and Debra has approached you to work on the design of the home page.

Debra envisions a page that uses semi-transparent colors and 3D transformations to make an attractive and eye-catching page. A preview of her completed design proposal is shown in Figure 4–71.

Figure 4–71 Chupacabra 15 home page



© Memo Angeles/Shutterstock.com; © Ivan Galashchuk/Shutterstock.com; © Andrey Armyagov/Shutterstock.com; © Away/Shutterstock.com; Source: Facebook; Source: Twitter, Inc.

Debra has provided you with the HTML code and the layout and reset style sheets. Your job will be to finish her work by inserting the graphic design styles.

Complete the following:

1. Using your editor, open the [cf\\_home\\_txt.html](#) and [cf\\_effects\\_txt.css](#) files from the `html04 ► case2` folder. Enter **your name** and **the date** in the comment section of each file, and save them as [cf\\_home.html](#) and [cf\\_effects.css](#) respectively.
2. Go to the [cf\\_home.html](#) file in your HTML editor. Within the document head, create a link to the [cf\\_reset.css](#), [cf\\_layout.css](#), and [cf\\_effects.css](#) style sheets. Take some time to study the content and structure of the document. Pay special note to the nested `div` elements in the center section of the page; you will use these to create a 3D cube design. Close the file, saving your changes.
3. Return to the [cf\\_effects.css](#) file in your editor and go to the HTML Styles section. Debra wants a background displaying a scene from last year's festival. Add a style rule for the `html` element that displays the `cf_back1.png` as a fixed background, centered horizontally and vertically in the browser window and covering the entire window.

4. Go to the Body Styles section and set the background color of the page body to `rgba(255, 255, 255, 0.3)`.
5. Go to the Body Header Styles section and change the background color of the body header to `rgba(51, 51, 51, 0.5)`.
6. Debra has placed useful information for the festival in `aside` elements placed within the left and right `section` elements. Go to the Aside Styles section and create a style rule for the `section aside` selector that adds a 10-pixel double border with color `rgba(92, 42, 8, 0.3)` and a border radius of 30 pixels.
7. Debra wants a curved border for every `h1` heading within an `aside` element. For the selector `section aside h1`, create a style rule that sets the border radius of the top-left and top-right corners to 30 pixels.
8. Define the perspective of the 3D space for the left and right sections by creating a style rule for those two sections that sets their perspective value to 450 pixels.
9. Create a style rule that rotates the `aside` elements within the left section  $25^\circ$  around the  $y$ -axis. Create another style rule that rotates the `aside` elements within the right section  $-25^\circ$  around the  $y$ -axis.
- ✚ **Explore** 10. Go to the Cube Styles section. Here you'll create the receding cube effect that appears in the center of the page. The cube has been constructed by creating a `div` element with the id `cube` containing five `div` elements belonging to the `cube_face` class with the ids `cube_bottom`, `cube_top`, `cube_left`, `cube_right`, and `cube_front`. (There will be no back face for this cube.) Currently the five faces are superimposed upon each other. To create the cube you have to shift and rotate each face in 3D space so that they form the five faces of the cube. First, position the cube on the page by creating a style rule for the `div#cube` selector containing the following styles:
  - a. Place the element using relative positioning.
  - b. Set the top margin to 180 pixels, the bottom margin to 150 pixels, and the left/right margins to `auto`.
  - c. Set the width and height to 400 pixels.
  - d. Set the perspective of the space to 450 pixels.
11. For each `div` element of the `cube_face` class, create a style rule that places the faces with absolute positioning and sets their width and height to 400 pixels.
- ✚ **Explore** 12. Finally, you'll construct the cube by positioning each of the five faces in 3D space so that they form the shape of a cube. Add the following style rules for each of the five faces to transform their appearance.
  - a. Translate the `cube_front` `div` element  $-50$  pixels along the  $z$ -axis.
  - b. Translate the `cube_left` `div` element  $-200$  pixels along the  $x$ -axis and rotate it  $90^\circ$  around the  $y$ -axis.
  - c. Translate the `cube_right` `div` element 200 pixels along the  $x$ -axis and rotate it  $90^\circ$  counter-clockwise around the  $y$ -axis.
  - d. Translate the `cube_top` `div` element  $-200$  pixels along the  $y$ -axis and rotate it  $90^\circ$  counter-clockwise around the  $x$ -axis.
  - e. Translate the `cube_bottom` `div` element 200 pixels along the  $y$ -axis and rotate it  $90^\circ$  around the  $x$ -axis.
13. Save your changes to style sheet file and open `cf_home.html` in your browser. Verify that the layout of your page matches Figure 4–71 including the center cube with the five faces of photos and text.



## OBJECTIVES

**Session 5.1**

- Create a media query
- Work with the browser viewport
- Apply a responsive design
- Create a pulldown menu with CSS

**Session 5.2**

- Create a flexbox
- Work with flex sizes
- Explore flexbox layouts

**Session 5.3**

- Create a print style sheet
- Work with page sizes
- Add and remove page breaks

# Designing for the Mobile Web

## Creating a Mobile Website for a Daycare Center

### Case | *Trusted Friends Daycare*

Marjorie Kostas is the owner of *Trusted Friends Daycare*, an early childhood education and care center located in Carmel, Indiana. You've been hired to help work on the redesign of the company's website. Because many of her clients access the website from their mobile phones, Marjorie is interested in improving the site's appearance on mobile devices. However, your design still has to be compatible with tablet devices and desktop computers. Finally, the site contains several pages that her clients will want to print, so your design needs to meet the needs of printed media.

## STARTING DATA FILES



# Session 5.1 Visual Overview:

The **viewport meta** tag is used to set the properties of the layout viewport.

This sets the width of the layout viewport equal to the width of the visual viewport.

This sets the initial scale of the viewport to 1.0.

```
<meta name="viewport" content="width=device-width, initial-scale=1" />
```

Responsive designs should start with base styles that apply to all devices, followed by mobile styles, tablet styles, and then desktop styles.

A **media query** is used to apply specified style rules to a device based on the device type and the device features.

```

/* =====
Base Styles
=====
*/
style rules

/* =====
Mobile Styles: 0 to 480 pixels
=====
*/
@media only screen and (max-width: 480px) {
  style rules
}

/* =====
Tablet Styles: 481px and greater
=====
*/
@media only screen and (min-width: 481px) {
  style rules
}

/* =====
Desktop Styles: 769px and greater
=====
*/
@media only screen and (min-width: 769px) {
  style rules
}

```

This media query matches screens with a maximum width of 480 pixels.

Within a media query are style rules that are only applied to devices that match the query.

This media query matches screens with a minimum width of 481 pixels.

This media query matches screens with a minimum width of 769 pixels.

# Media Queries

The image illustrates three different screen sizes and their corresponding website layouts for 'trusted friends daycare':

- Mobile (Smartphone):** The layout is a single column with a vertical navigation menu on the left. A callout box states: "Mobile styles are applied when the screen width is 0 to 480 pixels."
- Tablet:** The layout is a wider single column with a horizontal navigation bar at the top. A callout box states: "Tablet styles are applied once the screen width exceeds 480 pixels."
- Desktop (Monitor):** The layout is a multi-column design with a wide header, a navigation bar, and a main content area with two columns of text and images. A callout box states: "Desktop styles are applied once the screen width is 769 pixels and greater."



## Introducing Responsive Design

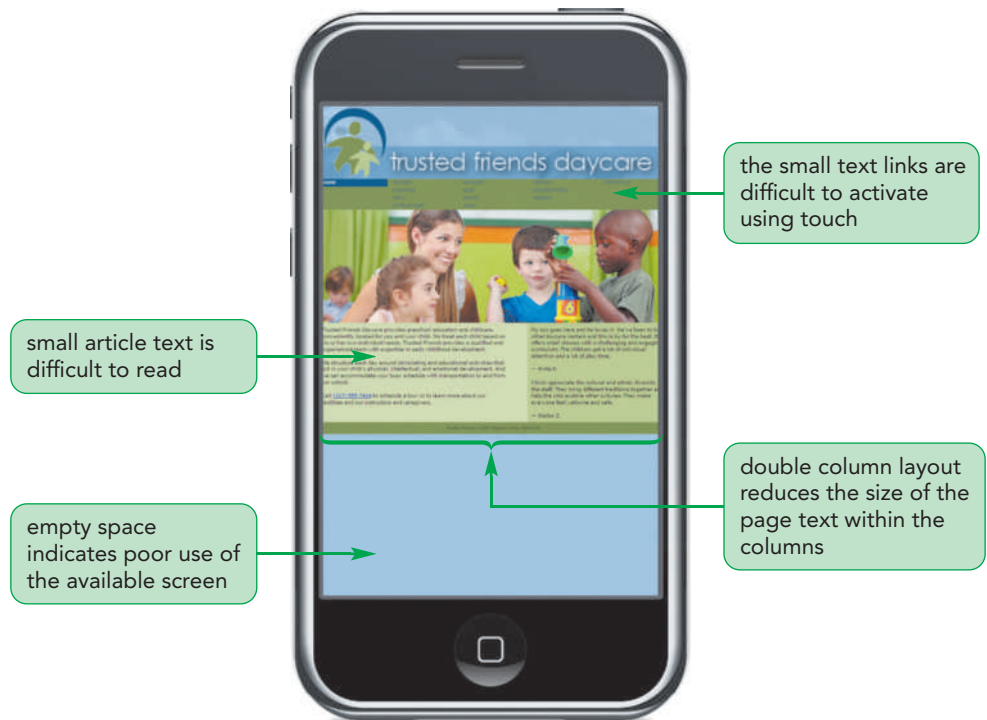
In the first four tutorials, you created a single set of layout and design styles for your websites without considering what type of device would be rendering the site. However, this is not always a practical approach and with many users increasingly accessing the web through mobile devices, a web designer must take into consideration the needs of those devices. Figure 5–1 presents some of the important ways in which designing for the mobile experience differs from designing for the desktop experience.

**Figure 5–1** Designing for mobile and desktop devices

User Experience	Mobile	Desktop
Page Content	Content should be short and to the point.	Content can be extensive, giving readers the opportunity to explore all facets of the topic.
Page Layout	Content should be laid out within a single column with no horizontal scrolling.	With a wider screen size, content can be more easily laid out in multiple columns.
Hypertext Links	Links need to be easily accessed via a touch interface.	Links can be activated more precisely using a cursor or mouse pointer.
Network Bandwidth	Sites tend to take longer to load over cellular networks and thus overall file size should be kept small.	Sites are quickly accessed over high-speed networks, which can more easily handle large file sizes.
Lighting	Pages need to be easily visible in outdoor lighting through the use of contrasting colors.	Pages are typically viewed in an office setting, allowing a broader color palette.
Device Tools	Mobile sites often need access to devices such as phone dialing, messaging, mapping, and built-in cameras and video.	Sites rarely have need to access desktop devices.

Viewing a web page on a mobile device is a fundamentally different experience than viewing the same web page on a desktop computer. As a result, these differences need to be taken into account when designing a website. Figure 5–2 shows the current home page of the Trusted Friends website as it appears on a mobile device.

Figure 5–2 Trusted Friends home page displayed on a mobile device



© Robert Kneschke/Shutterstock.com; BenBois/openclipart

Notice that the mobile device has automatically zoomed out to display the complete page width resulting in text that is difficult to read and small hypertext links that are practically unusable with a touch interface. While the design might be fine for a desktop monitor in landscape orientation, it's clear that it is ill-suited to a mobile device.

### TIP

For more information on the development of responsive design, refer to *Responsive Web Design* by Ethan Marcotte (<http://alistapart.com/article/responsive-web-design>).

What this website requires is a design that is not only specifically tailored to the needs of her mobile users but also is easily revised for tablet and desktop devices. This can be accomplished with responsive design in which the design of the document changes in response to the device rendering the page. An important leader in the development of responsive design is Ethan Marcotte, who identified three primary components of responsive design theory:

- **flexible layout** so that the page layout automatically adjusts to screens of different widths
- **responsive images** that rescale based on the size of the viewing device
- **media queries** that determine the properties of the device rendering the page so that appropriate designs can be delivered to specific devices

In the preceding tutorials, you've seen how to create grid-based fluid layouts and you've used images that scaled based on the width of the browser window and web page. In this session, you'll learn how to work with media queries in order to create a truly responsive website design.

## Introducing Media Queries

Media queries are used to associate a style sheet or style rule with a specific device or list of device features. To create a media query within an HTML file, add the following `media` attribute to either the `link` or `style` element in the document head

```
media="devices"
```

where *devices* is a comma-separated list of supported media types associated with a specified style sheet. For example, the following `link` element accesses the `output.css` style sheet file but only when the device is a printer or projection device:

```
<link href="output.css" media="print, projection" />
```

If any other device accesses this web page, it will not load the `output.css` style sheet file. Figure 5–3 lists other possible media type values for the `media` attribute.

Figure 5–3

Media types

Media Type	Used For
all	All output devices (the default)
braille	Braille tactile feedback devices
embossed	Paged Braille printers
handheld	Mobile devices with small screens and limited bandwidth
print	Printers
projection	Projectors
screen	Computer screens
speech	Speech and sound synthesizers, and aural browsers
tty	Fixed-width devices such as teletype machines and terminals
tv	Television-type devices with low resolution, color, and limited scrollability

When no `media` attribute is used, the style sheet is assumed to apply to all devices accessing the web page.

## The @media Rule

Media queries can also be used to associate specific style rules with specific devices by including the following `@media` rule in a CSS style sheet file

```
@media devices {  
    style rules  
}
```

where *devices* are supported media types and *style rules* are the style rules associated with those devices. For example, the following style sheet is broken into three sections: an initial style rule that sets the font color of all `h1` headings regardless of device, a second section that sets the font size for `h1` headings on screen or television devices, and a third section that sets the font size for `h1` headings that are printed:

```
h1 {  
    color: red;  
}  
@media screen, tv {  
    h1 {font-size: 2em;}  
}  
@media print {  
    h1 {font-size: 16pt;}  
}
```

Note that in this style sheet, the font size for screen and television devices is expressed using the relative `em` unit but the font size for print devices is expressed using points, which is a more appropriate sizing unit for that medium.

Finally, you can specify media devices when importing one style sheet into another by adding the media type to the `@import` rule. Thus, the following CSS rule imports the `screen.css` file only when a screen or projection device is being used:

```
@import url("screen.css") screen, projection;
```

The initial hope was that media queries could target mobile devices using the `handheld` device type; however, as screen resolutions improved to the point where the cutoff between mobile, tablet, laptop, and desktop was no longer clear, media queries began to be based on what features a device supported and not on what the device was called.

## Media Queries and Device Features

To target a device based on its features, you add the feature and its value to the `media` attribute using the syntax:

```
media="devices and | or (feature:value)"
```

where *feature* is the name of a media feature and *value* is the feature's value. The `and` and `or` keywords are used to create media queries that involve different devices or different features, or combinations of both.

The `@media` and `@import` rules employ similar syntax:

```
@media devices and|or (feature:value) {
    style rules
}
```

and

```
@import url(url) devices and|or (feature:value);
```

For example, the following media query applies the style rules only for screen devices with a width of 320 pixels.

```
@media screen and (device-width: 320px) {
    style rules
}
```

Figure 5-4 provides a list of the device features supported by HTML and CSS.

Figure 5-4

Media features

Feature	Description
<code>aspect-ratio</code>	The ratio of the width of the display area to its height
<code>color</code>	The number of bits per color component of the output device; if the device does not support color, the value is 0
<code>color-index</code>	The number of colors supported by the output device
<code>device-aspect-ratio</code>	The ratio of the <code>device-width</code> value to the <code>device-height</code> value
<code>device-height</code>	The height of the rendering surface of the output device
<code>device-width</code>	The width of the rendering surface of the output device
<code>height</code>	The height of the display area of the output device
<code>monochrome</code>	The number of bits per pixel in the device's monochrome frame buffer
<code>orientation</code>	The general description of the aspect ratio: equal to <code>portrait</code> when the height of the display area is greater than the width; equal to <code>landscape</code> otherwise
<code>resolution</code>	The resolution of the output device in pixels, expressed in either <code>dpi</code> (dots per inch) or <code>dpcm</code> (dots per centimeter)
<code>width</code>	The width of the display area of the output device

All of the media features in Figure 5–4, with the exception of `orientation`, also accept `min-` and `max-` prefixes, where `min-` provides a minimum value for the specified feature, and `max-` provides the feature's maximum value. Thus, the following media query applies style rules only for screen devices whose width is at most 700 pixels:

```
@media screen and (max-width: 700px) {  
    style rules  
}
```

Similarly, the following media query applies style rules only to screens that are at least 400 pixels wide:

```
@media screen and (min-width: 400px) {  
    style rules  
}
```

You can combine multiple media features using logical operators such as `and`, `not`, and `or`. The following query applies the enclosed styles to all media types but only when the width of the output devices is between 320 and 480 pixels (inclusive):

```
@media all and (min-width: 320px) and (max-width: 480px) {  
    style rules  
}
```

Some media features are directed toward devices that do not have a particular property or characteristic. This is done by applying the `not` operator, which negates any features found in the expression. For example, the following query applies only to media devices that are not screen or do not have a maximum width of 480 pixels:

```
@media not screen and (max-width: 480px) {  
    style rules  
}
```

### TIP

If you specify a feature without specifying a device, the media query will apply to all devices.

For some features, you do not have to specify a value but merely indicate the existence of the feature. The following query matches any screen device that also supports color:

```
@media screen and (color) {  
    style rules  
}
```

Finally, for older browsers that do not support media queries, CSS provides the `only` keyword to hide style sheets from those browsers. In the following code, older browsers will interpret `only` as an unsupported device name and so will not apply the enclosed style rules, while newer browsers will recognize the keyword and continue to apply the style rules.

```
@media only screen and (color) {  
    style rules  
}
```

All current browsers support media queries, but you will still see the `only` keyword used in many website style sheets.

## REFERENCE

### Creating a Media Query

- To create a media query that matches a device in a `link` or `style` element within an HTML file, use the following media attribute

```
media="devices and|or (feature:value)"
```

where *devices* is a comma-separated list of media types, *feature* is the name of a media feature, and *value* is the feature's value

- To create a media query, create the following `@media` rule within a CSS style sheet

```
@media devices and|or (feature:value) {
  style rules
}
```

where *style rules* are the style rules applied for the specified device and feature.

- To import a style sheet based on a media query, apply the following `@import` rule within a CSS style sheet

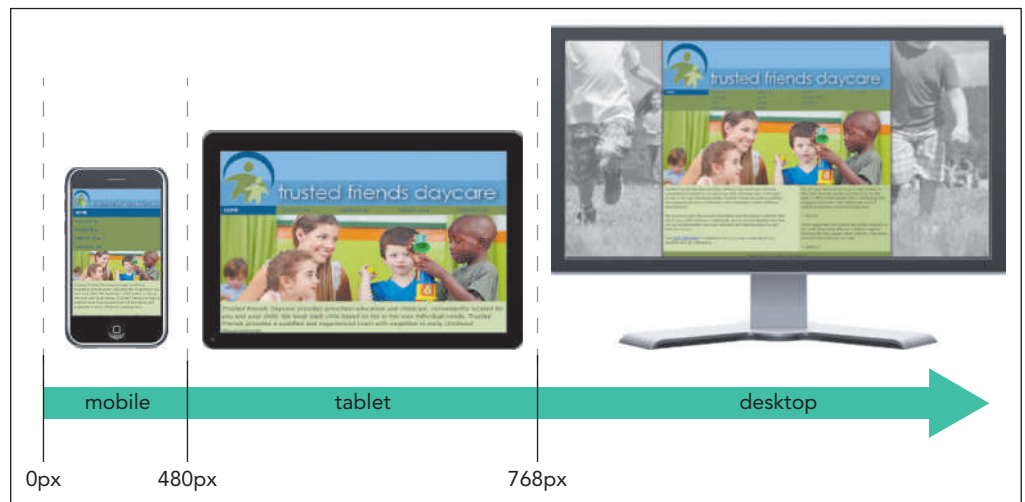
```
@import url(url) devices and|or (feature:value);
```

## Applying Media Queries to a Style Sheet

You meet with Marjorie to discuss her plans for the home page redesign. She envisions three designs: one for mobile devices, a different design for tablets, and finally a design for desktop devices based on the current appearance of the site's home page (see Figure 5–5).

Figure 5–5

Trusted Friends home page for different screen widths



© Robert Kneschke/Shutterstock.com; © dotshock/Shutterstock.com; BenBois/openclicart; JMLevick/openclicart; Easy/openclicart

The mobile design will be used for screen widths up to 480 pixels, the tablet design will be used for widths ranging from 481 pixels to 768 pixels, and the desktop design will be used for screen widths exceeding 768 pixels. To apply this approach, you'll create a style sheet having the following structure:

```
/* Base Styles */
  style rules

/* Mobile Styles */
@media only screen and (max-width: 480px) {
  style rules
}

/* Tablet Styles */
@media only screen and (min-width: 481px) {
  style rules
}

/* Desktop Styles */
@media only screen and (min-width: 769px) {
  style rules
}
```

Note that this style sheet applies the principle **mobile first** in which the overall page design starts with base styles that apply to all devices followed by style rules specific to mobile devices. Tablet styles are applied when the screen width is 481 pixels or greater, and desktop styles build upon the tablet styles when the screen width exceeds 768 pixels. Thus, as your screen width increases, you add on more features or replace features found in smaller devices. In general, with responsive design, it is easier to add new styles through progressive enhancement than to replace styles.

Marjorie has supplied you with the HTML code and initial styles for her website's home page. Open her HTML file now.

### To open the site's home page:

1. Use your editor to open the **tf\_home\_txt.html** and **tf\_styles1\_txt.css** files from the **html05** ► tutorial folder. Enter **your name** and **the date** in the comment section of each file and save them as **tf\_home.html** and **tf\_styles1.css** respectively.
2. Return to the **tf\_home.html** file in your editor and, within the document head, create links to the **tf\_reset.css** and **tf\_styles1.css** style sheet files.
3. Take some time to scroll through the contents of the document to become familiar with its contents and structure and then save your changes to the file, but do not close it.

Next, you'll insert the structure for the responsive design styles in the **tf\_styles1.css** style sheet, adding sections for mobile, tablet, and desktop devices.

### To add media queries to a style sheet:

1. Return to the **tf\_styles1.css** file in your editor.
2. Marjorie has already inserted the base styles that will apply to all devices at the top of the style sheet file. Take time to review those styles.



3. Scroll to the bottom of the document and add the following code and comments after the New Styles Added Below comment.

```
/* =====  
   Mobile Styles: 0px to 480px  
   =====  
*/  
@media only screen and (max-width: 480px) {  
  
}  
  
/* =====  
   Tablet Styles: 481px and greater  
   =====  
*/  
@media only screen and (min-width: 481px) {  
  
}  
  
/* =====  
   Desktop Styles: 769px and greater  
   =====  
*/  
@media only screen and (min-width: 769px) {  
  
}
```

Figure 5–6 highlights the media queries in the style sheet file.

Figure 5–6

### Creating media queries for different screen widths

media query matching  
screen devices with a  
maximum width of 480 pixels

media query matching  
screen devices with a  
minimum width of 481 pixels

media query matching  
screen devices with a  
minimum width 769 pixels

```
/* New Styles Added Below */  
  
/* =====  
   Mobile Styles: 0px to 480px  
   =====  
*/  
@media only screen and (max-width: 480px) {  
  
}  
  
/* =====  
   Tablet Styles: 481px and greater  
   =====  
*/  
@media only screen and (min-width: 481px) {  
  
}  
  
/* =====  
   Desktop Styles: 769px and greater  
   =====  
*/  
@media only screen and (min-width: 769px) {  
  
}
```

4. Save your changes to the file.

The media queries you've written are based on the screen width. However, before you can begin writing styles for each media query, you have to understand how those width values are interpreted by your browser.

## Exploring Viewports and Device Width

Web pages are viewed within a window called the viewport. For desktop computers, the viewport is the same as the browser window; however, this is not the case with mobile devices. Mobile devices have two types of viewports: a **visual viewport** displaying the web page content that fits within a mobile screen and a **layout viewport** containing the entire content of the page, some of which may be hidden from the user.

The two viewports exist in order to accommodate websites that have been written with desktop computers in mind. A mobile device will automatically zoom out of a page in order to give users the complete view of the page's contents, but as shown earlier in Figure 5–2, this often results in a view that is too small to be usable. While the user can manually zoom into a page to make it readable within the visual viewport, this is done at the expense of hiding content, as shown in Figure 5–7.

Figure 5–7 Comparing the visual and layout viewports



© Robert Kneschke/Shutterstock.com; BenBois/openclipart

Notice in the figure how the home page of the Trusted Friends website has been zoomed in on a mobile device so that only part of the page is displayed within the visual viewport and the rest of the page, which is hidden from the user, extends into the layout viewport.

Widths in media queries are based on the width of the layout viewport, not the visual viewport. Thus, depending on how the page is scaled, a width of 980 pixels might match the physical width of the device as shown in Figure 5–2 or it might extend beyond it as shown in Figure 5–7. In order to correctly base a media query on the

physical width of the device, you have to tell the browser that you want the width of the layout viewport matched to the device width by adding the following `meta` element to the HTML file:

```
<meta name="viewport" content="properties" />
```

where *properties* is a comma-separated list of viewport properties and their values, as seen in the example that follows:

```
<meta name="viewport"
  content="width=device-width, initial-scale=1" />
```

In this `meta` element, the `device-width` keyword is used to set the width of the layout viewport to the physical width of the device's screen. For a mobile device, this command sets the width of the layout viewport to the width of the device. The line `initial-scale=1` is added so that the browser doesn't automatically zoom out of the web page to fit the page content within the width of the screen. We want the viewport to match the device width, which is what the above `meta` element tells the browser to do.

**REFERENCE**

### Configuring the Layout Viewport

- To configure the properties of the layout viewport for use with media queries, add the following `meta` element to the HTML file

```
<meta name="viewport" content="properties" />
```

where *properties* is a comma-separated list of viewport properties and their values.

- To size the layout viewport so that it matches the width of the device without rescaling, use the following viewport `meta` element

```
<meta name="viewport"
  content="width=device-width, initial-scale=1" />
```

Add the viewport `meta` element to the `tf_home.html` file now, setting the width of the layout viewport to match the device width and the initial scale to 1.

### To define the visual viewport:

1. Return to the `tf_home.html` file in your editor.
2. Below the `meta` element that defines the character set, insert the following HTML tag:

```
<meta name="viewport"
  content="width=device-width, initial-scale=1" />
```

Figure 5–8 highlights the code for the viewport `meta` element.

Figure 5–8

## Setting the properties of the viewport

page does not automatically zoom out when the page is initially opened by the browser

sets the width of the layout viewport to the width of the device

```
<title>Trusted Friends Daycare</title>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1" />
<link href="tf_reset.css" rel="stylesheet" />
<link href="tf_styles1.css" rel="stylesheet" />
</head>
```

4. Save your changes to the file.
5. Open the **tf\_home.html** file in your browser. Figure 5–9 shows the initial design of the page.

Figure 5–9

## Mobile layout of the Trusted Friends home page

The screenshot shows the mobile layout of the Trusted Friends daycare home page. The layout is organized into several sections:

- company logo:** Located at the top left, featuring a stylized logo of a person and a child.
- navigation menu:** A vertical list of menu items on the left side, including HOME, CLASSES (with a submenu), INFANTS, TODDLERS, PRE-K, AFTER SCHOOL, PARENTS (with a submenu), ARTICLES, BLOG, FORMS, LEGAL, ABOUT US (with a submenu), HISTORY, ACCREDITATION, REVIEWS, and CONTACT US.
- main article:** A large text block in the center containing a paragraph about childhood education and daycare, followed by a call to action: "Call (317) 555-7414 to schedule a tour or to learn more about our facilities and our instructors."
- aside comments:** A section on the right side containing two testimonials from Anita K. and Stefan Z.
- footer:** Located at the bottom of the page, containing the text "Trusted Friends • 3430 Regency Lane, Carmel, IN".

Now that you've set up the media queries and configured the viewport, you can work on the design of the home page. You'll start by designing for mobile devices.

**INSIGHT**

### Not All Pixels Are Equal

While pixels are a basic unit of measurement in web design, there are actually two types of pixels to consider as you design a website. One is a **device pixel**, which refers to the actual physical pixel on a screen. The other is a **CSS pixel**, which is the fundamental unit in CSS measurements. The difference between device pixels and CSS pixels is easiest to understand when you zoom into and out of a web page. For example, the following style creates an `aside` element that is 300 CSS pixels wide:

```
aside: {width: 300px;}
```

However, the element is not necessarily 300 device pixels. If the user zooms into the web page, the apparent size of the article increases as measured by device pixels but remains 300 CSS pixels wide, resulting in 1 CSS pixel being represented by several device pixels.

The number of device pixels matched to a single CSS pixel is known as the **device-pixel ratio**. When a page is zoomed at a factor of 2x, the device-pixel ratio is 2, with a single CSS pixel represented by a 2x2 square of device pixels.

One area where the difference between device pixels and CSS pixels becomes important is in the development of websites optimized for displays with high device-pixel ratios. Some mobile devices are capable of displaying images with a device pixel ratio of 3, resulting in free crisp and clear images. Designers can optimize their websites for these devices by creating one set of style sheets for low-resolution displays and another for high-resolution displays. The high-resolution style sheet would load extremely detailed, high-resolution images, while the low-resolution style sheet would load lower resolution images better suited to devices that are limited to smaller device-pixel ratios. For example, the following media query

```
<link href="retina.css" rel="stylesheet"
media="only screen and (-webkit-min-device-pixel-ratio: 2) " />
```

loads the `retina.css` style sheet file for high-resolution screen devices that have device-pixel ratios of at least 2. Note that currently the `device-pixel-ratio` feature is a browser-specific extension supported only by WebKit.

## Creating a Mobile Design

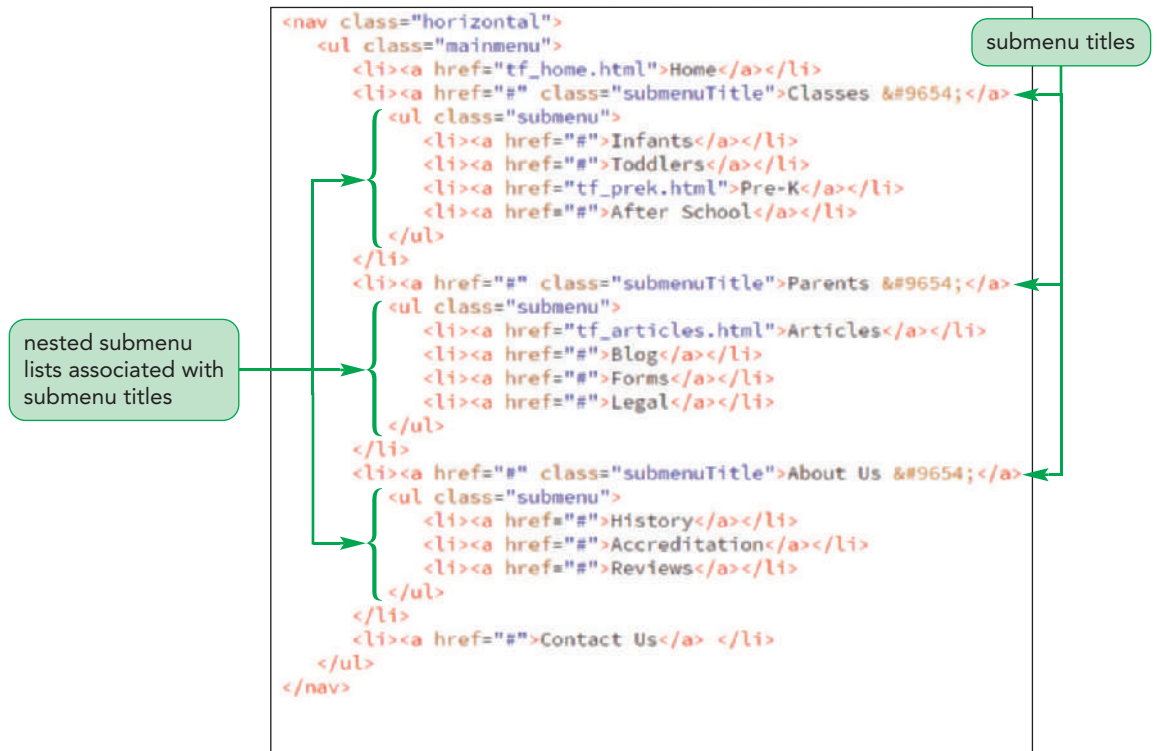
A mobile website design should reflect how users interact with their mobile devices. Because your users will be working with a small handheld touchscreen device, one key component in your design is to have the most important information up-front and easily accessible, which means your home page on a mobile device needs to be free of unnecessary clutter. Another important principle of designing for mobile devices is that you should limit the choices you offer to your users. Ideally, there should only be a few navigation links on the screen at any one time.

With these principles in mind, consider the current layout of the Trusted Friends home page shown in Figure 5–9. The content is arranged within a single column providing the maximum width for the text and images, but an area of concern for Marjorie is the long list of hypertext links, which forces the user to scroll vertically down the page to view information about the center. Most mobile websites deal with this issue by hiding extensive lists of links in pulldown menus, appearing only in response to a tap of a major heading in the navigation list. You'll use this technique for the Trusted Friends home page.

## Creating a Pulldown Menu with CSS

Marjorie has already laid the foundation for creating a pulldown menu in her HTML code. Figure 5–10 shows the code used to mark the contents of the navigation list in the body header.

Figure 5–10 Submenus in the navigation list



Marjorie has created a navigation bar that includes topical areas named Classes, Parents, and About Us. Within each of these topical areas are nested lists containing links to specific pages on the Trusted Friends website. Marjorie has put each of these nested lists within a class named *submenu*. So, first you'll hide each of these submenus to reduce the length of the navigation list as it is rendered within the user's browser. You'll place this style rule in the section for Base Styles because it will be used by both mobile and tablet devices (but not by desktop devices as you'll see later).

### To hide a submenu:

1. Return to the **tf\_styles1.css** file in your editor.
2. Scroll to the Pulldown Menu Styles section and add the following style rule:

```

ul.submenu {
  display: none;
}

```

Figure 5–11 highlights the styles to hide the navigation list submenus.



Figure 5–11

## Hiding the navigation list submenus

prevents the submenu unordered lists from being displayed

```
/* Pulldown Menu Styles */
ul.submenu {
  display: none;
}
```

3. Save your changes to the file and then reload the `tf_home.html` file in your browser. Verify that the navigation list no longer shows the contents of the submenus but only the Home, Classes, Parents, About Us, and Contact Us links. See Figure 5–12.

Figure 5–12

## Navigation list with hidden submenus

submenu lists are hidden



© Robert Kneschke/Shutterstock.com

Next, you want to display a nested submenu only when the user hovers the mouse pointer over its associated submenu title, which for this page are the Classes, Parents, and About Us titles. Because the submenu follows the submenu title in the HTML file (see Figure 5–10), you can use the following selector to select the submenu that is immediately preceded by a hovered submenu title:

```
a.submenuTitle:hover+ul.submenu
```

However, this selector is not enough because you want the submenu to remain visible as the pointer moves away from the title and hovers over the now-visible submenu. So, you need to add `ul.submenu:hover` to the selector:

```
a.submenuTitle:hover+ul.submenu, ul.submenu:hover
```



To make the submenu visible, you change its display property back to `block`, resulting in the following style rule:

```
a.submenuTitle:hover+ul.submenu, ul.submenu:hover {
    display: block;
}
```

You may wonder why you don't use only the `ul.submenu:hover` selector. The reason is that you can't hover over the submenu until it's visible and it won't be visible until you first hover over the submenu title. Add this rule now to the `tf_styles1.css` style sheet and test it.

### To redisplay the navigation submenus:

1. Return to the `tf_styles1.css` file in your editor.
2. Add the following style rule to the Pulldown Menu Styles section:

```
a.submenuTitle:hover+ul.submenu, ul.submenu:hover {
    display: block;
}
```

Figure 5–13 highlights the styles to display the navigation list submenus.

Figure 5–13

### Displaying the hidden submenus

The diagram shows a code editor window with the following CSS code:

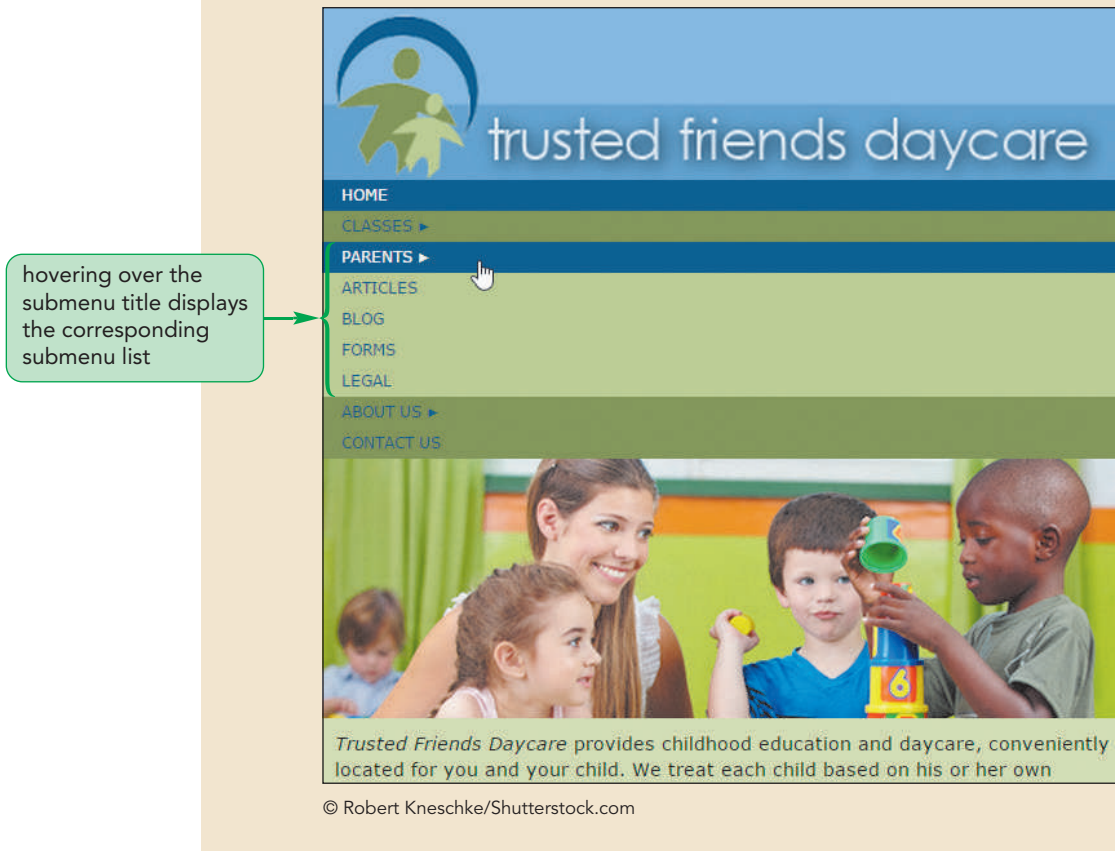
```
/* Pulldown Menu Styles */
ul.submenu {
    display: none;
}
a.submenuTitle:hover+ul.submenu, ul.submenu:hover {
    display: block;
}
```

Annotations explain the code:

- selects the submenu that is preceded by a hovered-over submenu title**: Points to the `a.submenuTitle:hover+ul.submenu` selector.
- selects the now-visible submenu as it's being hovered over**: Points to the `ul.submenu:hover` selector.
- makes the submenu visible by changing the display property to block**: Points to the `display: block;` property in the second rule.

3. Save your changes to the file and then reload the `tf_home.html` file in your browser. Hover your mouse pointer over each of the submenu titles and verify that the corresponding submenu becomes visible and remains visible as you move the mouse pointer over its contents.

Figure 5–14 shows the revised appearance of the navigation list using the pulldown menus.

**Figure 5–14** Displaying the contents of a pulldown menu

The hover event is used with mouse pointers on desktop computers, but it has a different interpretation when applied to mobile devices. Because almost all mobile devices operate via a touch interface, there is no hovering. A mobile browser will interpret a hover event as a tap event in which the user taps the page object. When the hover event is used to hide an object or display it (as we did with the submenus), mobile browsers employ a double-tap event in which the first tap displays the page object and a second tap, immediately after the first, activates any hypertext links associated with the object. To display the Trusted Friends submenus, the user would tap the submenu title and to hide the submenus the user would tap elsewhere on the page.

To test the hover action, you need to view the Trusted Friends page on a mobile device or a mobile emulator.

## Testing Your Mobile Website

The best way to test a mobile interface is to view it directly on a mobile device. However, given the large number of mobile devices and device versions, it's usually not practical to do direct testing on all devices. An alternative to having the physical device is to emulate it through a software program or an online testing service. Almost every mobile phone company provides a software development kit or SDK that developers can use to test their programs and websites. Figure 5–15 lists some of the many **mobile device emulators** available on the web at the time of this writing.

Figure 5–15

## Popular device emulators

Mobile Emulator	Description
Android SDK	Software development kit for Android developers ( <a href="http://developer.android.com/sdk">developer.android.com/sdk</a> )
iOS SDK	Software development kit for iPhone, iPad, and other iOS devices ( <a href="http://developer.apple.com">developer.apple.com</a> )
Mobile Phone Emulator	Online emulation for a variety of mobile devices ( <a href="http://www.mobilephoneemulator.com">www.mobilephoneemulator.com</a> )
Mobile Test Me	Online emulation for a variety of mobile devices ( <a href="http://mobiletest.me">mobiletest.me</a> )
Opera Mobile SDK	Developer tools for the Opera Mobile browser ( <a href="http://www.opera.com/developer">www.opera.com/developer</a> )

Browsers are also starting to include device emulators as part of their developer tools. You will examine the device emulator that is supplied with the Google Chrome browser and use it to view the Trusted Friends home page under a device of your choosing. If you don't have access to the Google Chrome browser, review the steps that follow and apply them to the emulator of your choice.

### Viewing the Google Chrome device emulator:




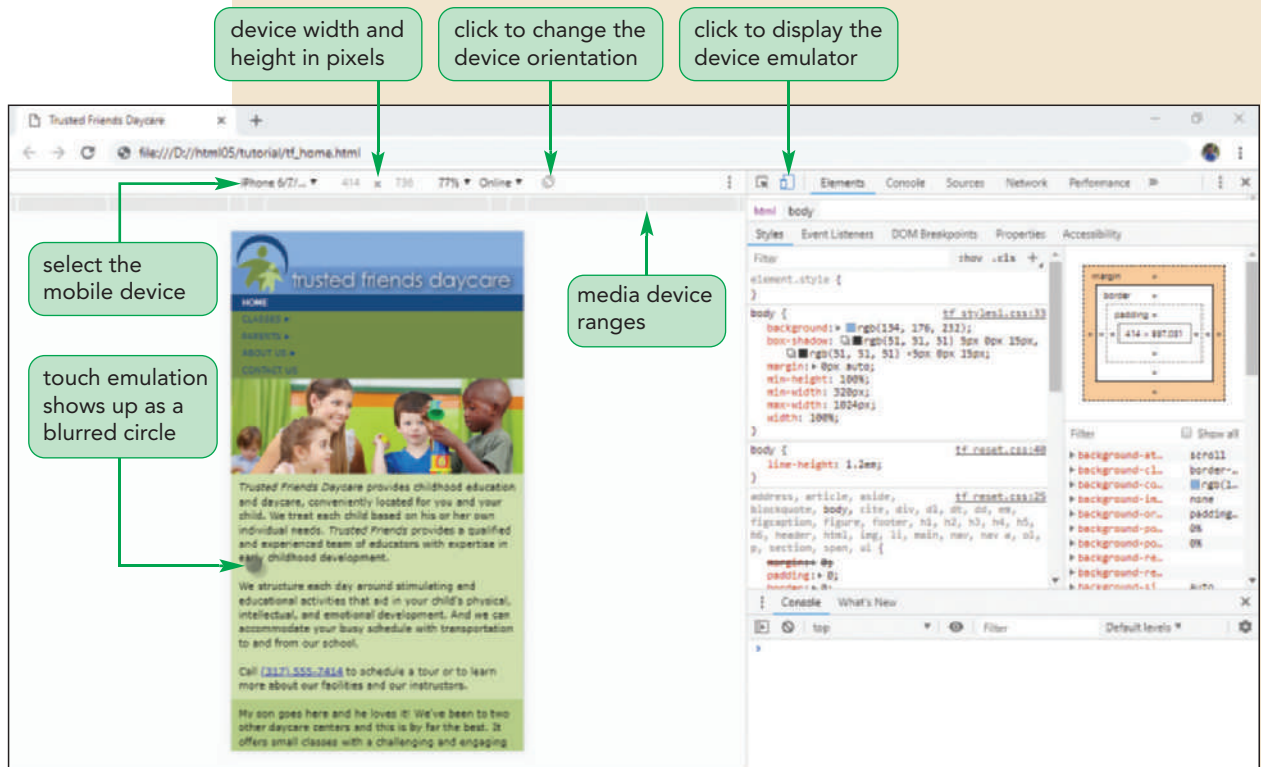
1. Return to the **tf\_home.html** file in the Google Chrome browser and press **F12** to open the developer tools pane.
2. If necessary, click the **device** icon  located at the top of the developer pane to display the device toolbar.
3. Select a device of your choosing from the drop-down list of devices on the developer toolbar.
4. Refresh or reload the web page to ensure that the display parameters of your selected device are applied to the rendered page.  
The emulator also allows you to view the effect of changing the orientation of the phone from portrait to landscape.
5. Click the **rotate** button  located on the device toolbar to switch to landscape orientation. Click the **rotate** button again to switch back to portrait mode.  
Google Chrome's device emulator can also emulate the touch action. The touch point is represented by a semitransparent circle .
6. Move the touch point over Classes, Parents, or About Us and verify that when you click (tap) the touch point on a submenu title the nested submenu contents are displayed.
7. Verify that you when you click elsewhere in the page the submenu contents are hidden.

Figure 5–16 shows the effect of opening a submenu with the touch emulator.

Figure 5–16 Using the Google Chrome device emulator tool



© Robert Kneschke/Shutterstock.com

8. Continue to explore Google Chrome's device emulators, trying out different combinations of devices and screen orientations. Press **F12** again to close the developer window.

An important aspect of mobile design is optimizing your site's performance under varying network conditions. Thus, in addition to emulating the properties of the mobile device, Google Chrome's device emulator can also emulate network connectivity.

Marjorie wants to increase the font size of the links in the navigation list to make them easier to access using touch. She also wants to hide the customer comments that have been placed in the `aside` element (because she doesn't feel this will be of interest to mobile users). Because these changes only apply to the mobile device version of the page, you'll add the style rules within the media query for mobile devices.

### To hide the customer comments:

1. Return to the `tf_styles1.css` file in your editor and go to the Mobile Styles section.
2. Within the media query for screen devices with a maximum width of 480 pixels, add the following style rule to increase the font size of the hypertext links in the navigation list. Indent the style rule to offset it from the braces around the media query.

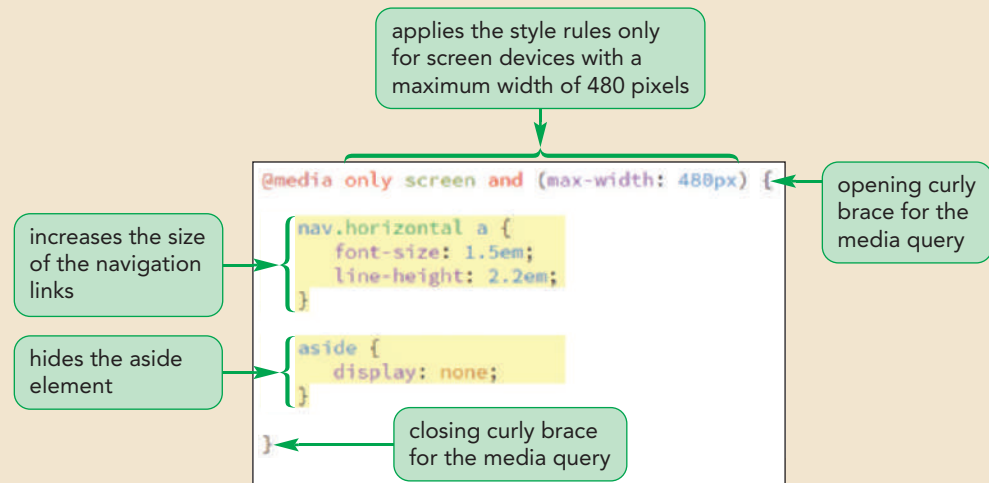
```
nav.horizontal a {
  font-size: 1.5em;
  line-height: 2.2em;
}
```

The styles rules for a media query must always be placed within curly braces to define the extent of the query.

3. Add the following style rule to hide the `aside` element (once again indented from the surrounding media query):

```
aside {  
    display: none;  
}
```

Figure 5–17 highlights the style rules in the media query for mobile devices.

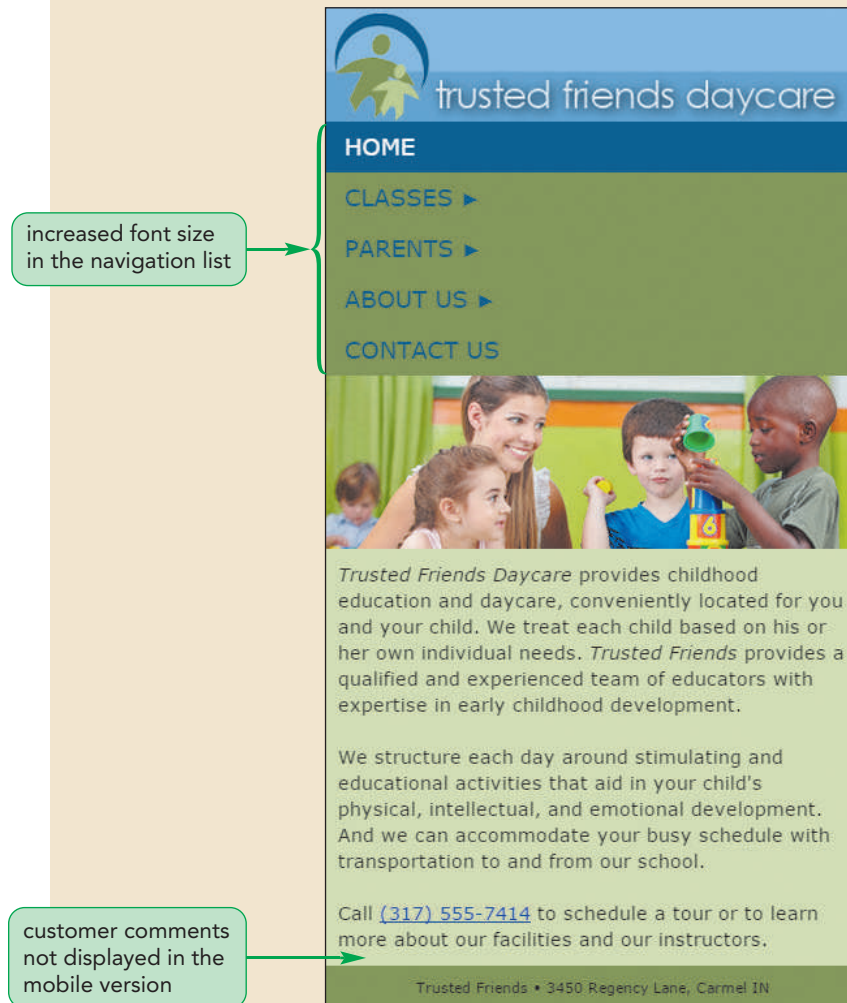
**Figure 5–17****Hiding the aside element for mobile devices**

4. Save your changes to the file and then reload the `tf_home.html` file in your browser. Reduce the width of the browser window to 480 pixels or below (or view the page in your mobile emulator). Verify that the customer comments are no longer displayed on the web page and that the size of the navigation links has been increased.

Figure 5–18 shows the final design of the mobile version.

Figure 5–18

Final design of the mobile version of the home page



Now that you've completed the mobile design of the page, you'll start to work on the design for tablet devices.

## Creating a Tablet Design

Under the media query you've set up, your design for tablet devices will be applied for screen widths greater than 480 pixels. The pulldown menu you created was part of the base styles, so it is already part of the tablet design; however, with the wider screen, Marjorie would like the submenus displayed horizontally rather than vertically. You can accomplish this by adding a style rule to the tablet media query to float the submenus side-by-side.

### To begin writing the tablet design:

1. Return to the `tf_styles1.css` file in your editor and scroll down to the media query for the tablet styles.

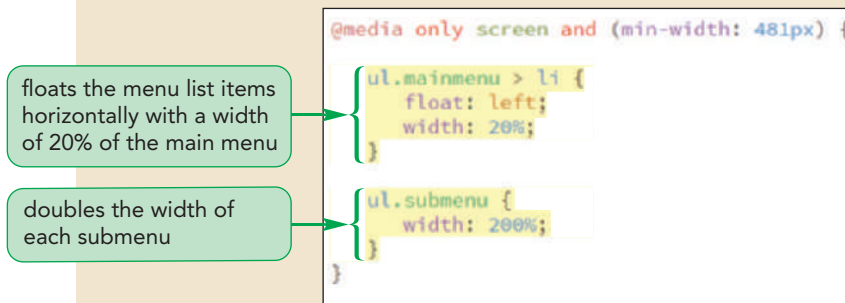
2. Within the media query, add the following style to float the five list items, which are direct children of the main menu, side-by-side. Set the width of each list item to 20% of the total width of the main menu.

```
ul.mainmenu > li {  
  float: left;  
  width: 20%;  
}
```

3. Double the widths of the submenus so that they stand out better from the main menu titles by adding the following style rule.

```
ul.submenu {  
  width: 200%;  
}
```

Figure 5–19 highlights the style rule within the media query for tablet devices.

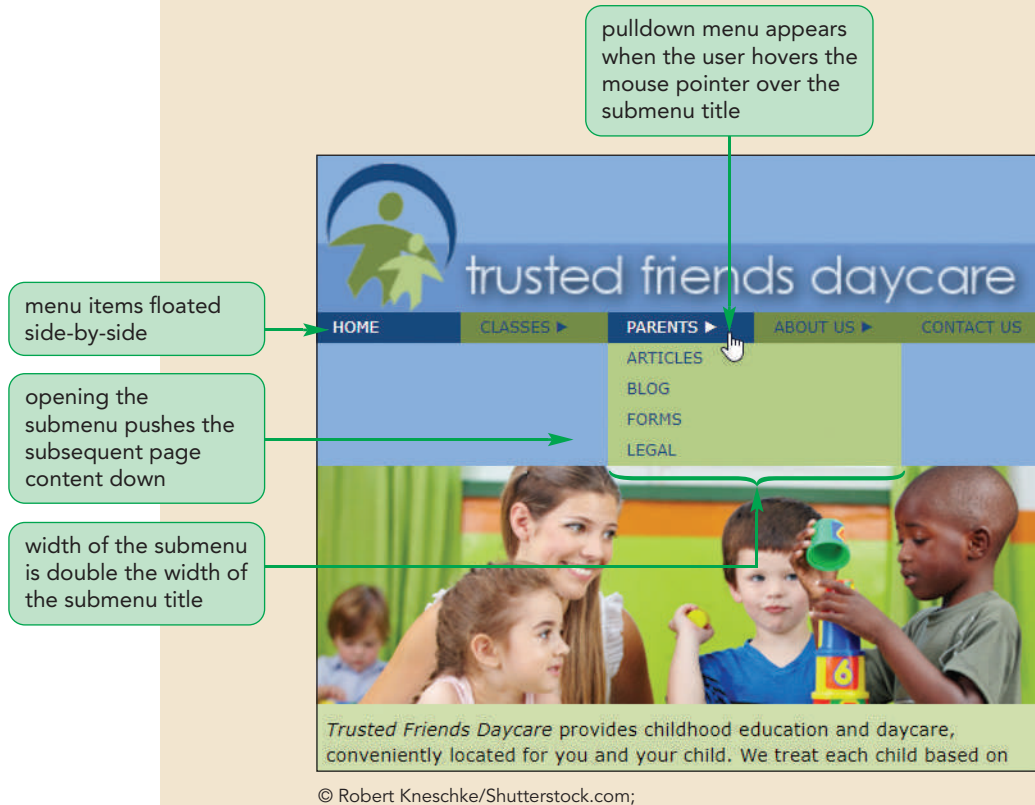
**Figure 5–19****Formatting the navigation menus for tablet devices**

4. Save your changes to the style sheet and then reload the `tf_home.html` file in your web browser.
5. Increase the width of the browser window beyond 480 pixels to switch from the mobile design to the tablet design. Verify that the submenu titles are now laid out horizontally and that if you hover your mouse pointer over the submenu titles, the contents of the submenu are made visible on the screen. See Figure 5–20.



Figure 5–20

## Pull-down menus for the tablet layout



6. Scroll down as needed and note that the customer comments now appear at the bottom of the page because they were only hidden for the mobile version of this document.

Marjorie notices that opening the submenus pushes the subsequent page content down to make room for the submenu. She prefers the submenus to overlay the page content. You can accomplish this by placing the submenus with absolute positioning. Remember that objects placed with absolute positioning are removed from the document flow and thus, will overlay subsequent page content. To keep the submenus in their current position on the page, you'll make each main list item a container for its submenu by setting its `position` property to `relative`. Thus, each submenu will be placed using absolute positioning with its main list item. You will not need to set the `top` and `left` coordinates for these items because you'll use the default value of 0 for both. Because the submenus will overlay page content, Marjorie suggests you add a drop shadow so, when a submenu is opened, it will stand out more from the page content.

### To position the navigation submenus:

1. Return to the `tf_styles1.css` style sheet in your editor.
2. Locate the style rule for the `ul.mainmenu > li` selector in the Tablet Styles section and add the following style:

```
position: relative;
```

- 3. Add the following style to the `ul.submenu` selector in the Tablet Styles section:

```
box-shadow: rgb(51, 51, 51) 5px 5px 15px;
position: absolute;
```

Figure 5–21 highlights the new styles.

Figure 5–21

## Placing the pulldown menus with absolute positioning

applies the style rules only for screen devices with a minimum width of 481 pixels

```
@media only screen and (min-width: 481px) {
  ul.mainmenu > li {
    float: left;
    position: relative;
    width: 28%;
  }
  ul.submenu {
    box-shadow: rgb(51, 51, 51) 5px 5px 15px;
    position: absolute;
    width: 200%;
  }
}
```

places the menu list items using relative positioning

absolutely positions the submenus within each menu list item

adds a drop shadow to each submenu

- 4. Save your changes to the style sheet and then reload the `tf_home.html` file in your web browser.
- 5. Verify that when you open the pulldown menus, the subsequent page content is not shifted downward. Figure 5–22 highlights the final design for the tablet version of the home page.

Figure 5–22

## Revised design of the pulldown menus

page content does not shift when the pulldown menu is opened

Trusted Friends Daycare provides childhood education and daycare, conveniently located for you and your child. We treat each child based on his

You'll complete your work on the home page by creating the desktop version of the page design.

## Creating a Desktop Design

Some of the designs that will be used in the desktop version of the page have already been placed in the Base Styles section of the `tf_styles1.css` style sheet. For example, the maximum width of the web page has been set to 1024 pixels. For browser windows that exceed that width, the web page will be displayed on a fixed background image of children playing. Other styles are inherited from the style rules for tablet devices. For example, desktop devices will inherit the style rule that floats the navigation submenus alongside each other within a single row. All of which illustrates an important principle in designing for multiple devices: *don't reinvent the wheel*. As much as possible allow your styles to build upon each other as you move to wider and wider screens.

However, there are some styles that you will have to implement only for desktop devices. With the wider screen desktop screens, you don't need to hide the submenus in a pulldown menu system. Instead you can display all of the links from the navigation list. You'll change the submenu background color to transparent so that it blends in with the navigation list and you'll remove the drop shadows you created for the tablet design. The submenus will always be visible, so you'll change their `display` property from `none` to `block`. Finally, you'll change their position to `relative` because you no longer want to take the submenus out of the document flow and you'll change their width to 100%. Apply the styles now to modify the appearance of the submenus.

### To start working on the desktop design:

1. Return to the `tf_styles1.css` style sheet in your editor and within the media query for devices with screen widths 769 pixels or greater insert the following style rule to format the appearance of the navigation submenus.

```
ul.submenu {
  background: transparent;
  box-shadow: none;
  display: block;
  position: relative;
  width: 100%;
}
```

2. The navigation list itself needs to expand so that it contains all of its floated content. Add the following style rule to the media query for desktop devices:

```
nav.horizontal::after {
  clear: both;
  content: "";
  display: table;
}
```

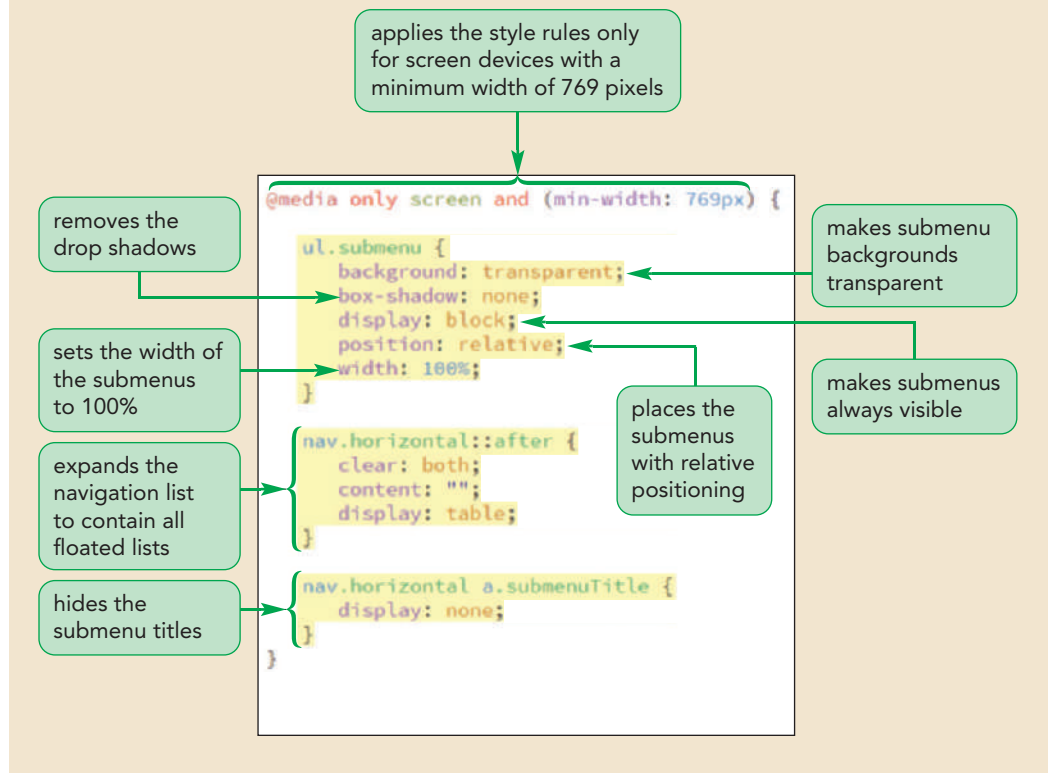
3. Finally with no hidden submenus, there is no reason to have a submenu title. Add the following style rule to remove the submenu titles:

```
nav.horizontal a.submenuTitle {
  display: none;
}
```

Figure 5–23 highlights the new style rules in the desktop media query.

Figure 5–23

## Adding design styles for the browser background and page body



With a wider screen, you want to order to avoid long lines of text, which are difficult to read. Modify the layout of the desktop design so that the main article and the customer comments are floated side-by-side within the same row.

### To change the layout of the article and aside elements:

1. Within the media query for desktop devices, add the following style rules to float the article and aside elements:

```

article {
  float: left;
  margin-right: 5%;
  width: 55%;
}
aside {
  float: left;
  width: 40%;
}

```

Figure 5–24 highlights the final style rules in the desktop media query.

Figure 5–24

## Styles for the article and aside elements

floats the main article with a width of 55% and a right margin of 5%

floats the aside element with a width of 40%

```
nav.horizontal a.submenuTitle {
  display: none;
}

article {
  float: left;
  margin-right: 5%;
  width: 55%;
}

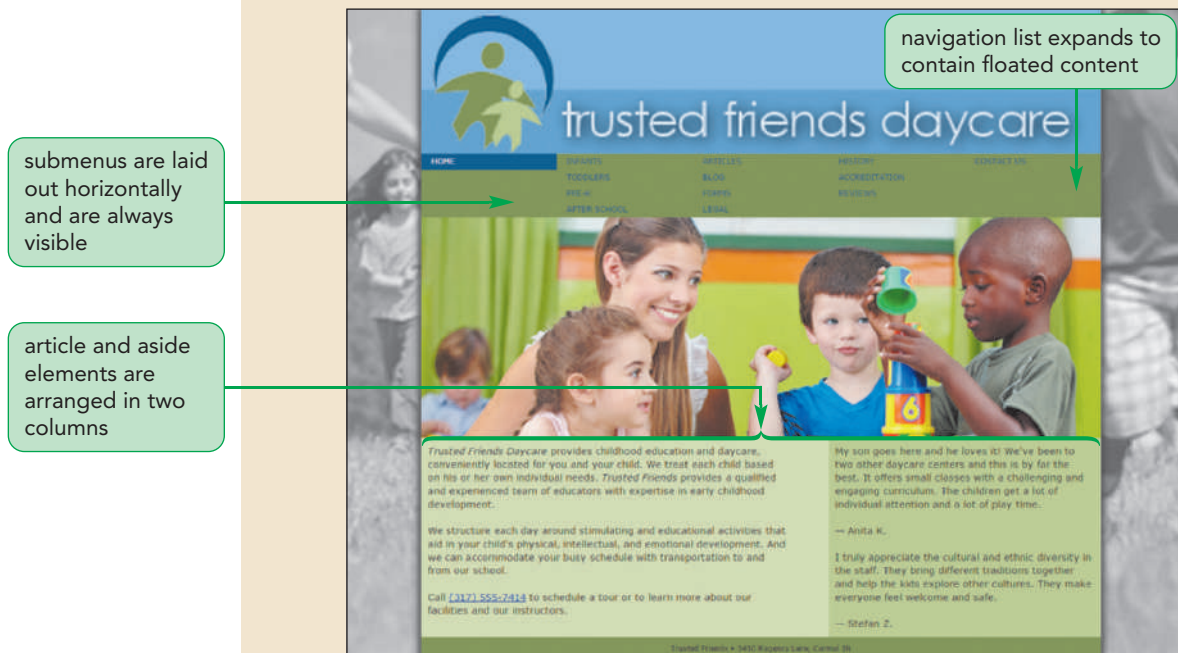
aside {
  float: left;
  width: 40%;
}
```

2. Save your changes to the style sheet and then reload `tf_home.html` in your browser.

Figure 5–25 shows the final appearance of the desktop design.

Figure 5–25

## Final desktop design for the Trusted Friends daycare



submenus are laid out horizontally and are always visible

article and aside elements are arranged in two columns

navigation list expands to contain floated content

© Robert Kneschke/Shutterstock.com; © dotshock/Shutterstock.com

3. Resize your web browser and verify that as you change the browser window width, the layout changes from the mobile to the tablet to the desktop design.

You show the final design of the home page to Marjorie. She is pleased by the changes you've made and likes that the page's content and layout will automatically adapt to different screen widths.

**PROSKILLS**

### *Problem Solving: Optimizing Your Site for the Mobile Web*

The mobile browser market is a rapidly evolving and growing field with more new devices and apps introduced each month. Adapting your website for the mobile web is not a luxury, but a necessity.

A good mobile design matches the needs of consumers. Mobile users need quick access to main sources of information without a lot of the extra material often found in the desktop versions of their favorite sites. Here are some things to keep in mind as you create your mobile designs:

- *Keep it simple.* To accommodate the smaller screen sizes and slower connection speeds, scale down each page to a few key items and articles. Users are looking for quick and obvious information from their mobile sites.
- *Resize your images.* Downloading several images can bring a mobile device to a crawl. Reduce the number of images in your mobile design, and use a graphics package to resize the images so they are optimized in quality and sized for a smaller screen.
- *Scroll vertically.* Readers can more easily read your page when they only have to scroll vertically. Limit yourself to one column of information in portrait orientation and two columns in landscape.
- *Make your links accessible.* Clicking a small hypertext link is extremely difficult to do on a mobile device with a touch screen interface. Create hypertext links that are easy to locate and activate.

Above all, test your site on a variety of devices and under different conditions. Mobile devices vary greatly in size, shape, and capability. What works on one device might fail utterly on another. Testing your code on a desktop computer is only the first step; you may also need access to the devices themselves. Even emulators cannot always capture the nuances involved in the performance of an actual mobile device.

You've completed your work on the design of the Trusted Friends home page with a style sheet that seamlessly transitions between mobile, tablet, and desktop devices. In the next session, you'll explore how to use flexible boxes to achieve a responsive design.

### Session 5.1 Quick Check

- Which of the following is *not* a part of responsive design theory?
  - flexible layouts
  - pulldown menus
  - image rescaling
  - media queries
- Which attribute do you add to a `link` element for aural browsers?
  - `media = "aural"`
  - `type = "aural"`
  - `media = "speech"`
  - `type = "speech"`
- What `@rule` do you use for braille device?
  - `@media braille`
  - `@braille true`
  - `@type braille`
  - `@media nonscreen`
- What `@rule` loads style rules for screen devices up to a maximum width of 780 pixels?
  - `@screen: 780px`
  - `@media screen and (width: 780px)`
  - `@screen and (width <= 780px)`
  - `@media screen and (max-width: 780px)`
- What attribute would you add to a `link` element for screen devices whose width ranges from 480 pixels up to 780 pixels (inclusive)?
  - `media="screen" min-width="480px" max-width="780px"`
  - `media="screen and (width=480px - 780px)"`
  - `minScreenWidth = "480px" maxScreenWidth = "780px"`
  - `media="screen and (min-width: 480px and max-width: 780px)"`
- In general, what media rules should be listed first in your media queries if you want to support mobile, tablet, laptop, and desktop devices?
  - mobile
  - tablet
  - laptop
  - desktop
- Which viewport displays the web page content that fits within mobile screen?
  - layout
  - visual
  - webpage
  - browser
- Which viewport contains the entire content of the page, some of which may be hidden from the user?
  - layout
  - visual
  - webpage
  - browser



# Session 5.2 Visual Overview:

A **flexbox** contains items whose size automatically expands or contracts to match the dimensions of the box.

To create a flexbox, set the **display** property to **flex**.

```
body {  
  display: flex;  
  flex-flow: row wrap;  
}
```

To define the orientation of the flexbox and whether items can wrap to a new line, apply the **flex-flow** property.

Use the **flex** property to define the size of the flex items and how they will grow or shrink in response to the changing size of the flexbox.

```
aside {  
  flex: 1 1 120px;  
}
```

The **flex-shrink** value specifies how fast the item shrinks below its basis size relative to other items in the flexbox.

The **flex-grow** value specifies how fast the item grows above its basis size relative to other items in the flexbox.

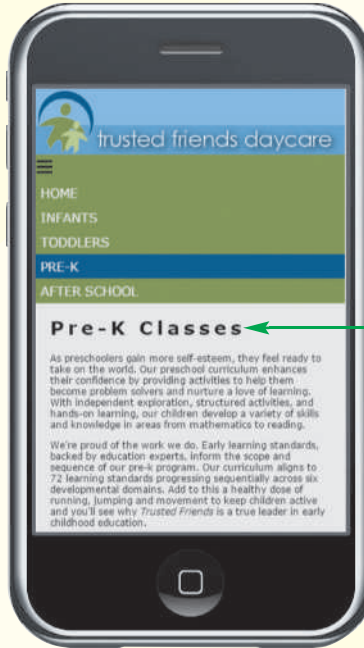
```
section#main {  
  flex: 3 1 361px;  
}
```

The **flex-basis** value provides the basis or initial size of the item prior to flexing.

```
section#topics {  
  display: flex;  
  flex-flow: row wrap;  
}
```

```
section#topics article {  
  flex: 1 1 200px;  
}
```

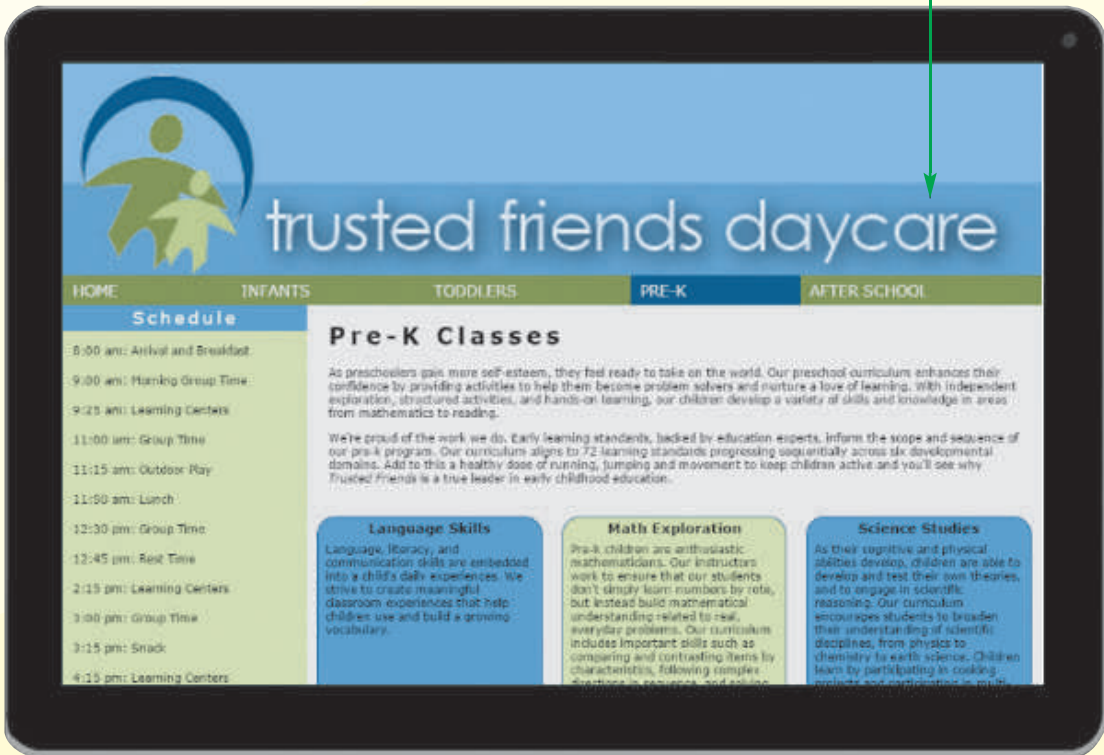
# Flexible Layouts



BenBois/opencipart

With narrower screens, a flexbox layout automatically places items within a single column.

With wider screens, the items are free to expand, automatically placing themselves into multiple columns.



Jmlevick/opencipart

## Introducing Flexible Boxes

So far our layouts have been limited to a grid system involving floating elements contained within a fixed or fluid grid of rows and columns. One of the challenges of this approach under responsive design is that you need to establish a different grid layout for each class of screen size. It would be much easier to have a single specification that automatically adapts itself to the screen width without requiring a new layout design. One way of achieving this is with flexible boxes.

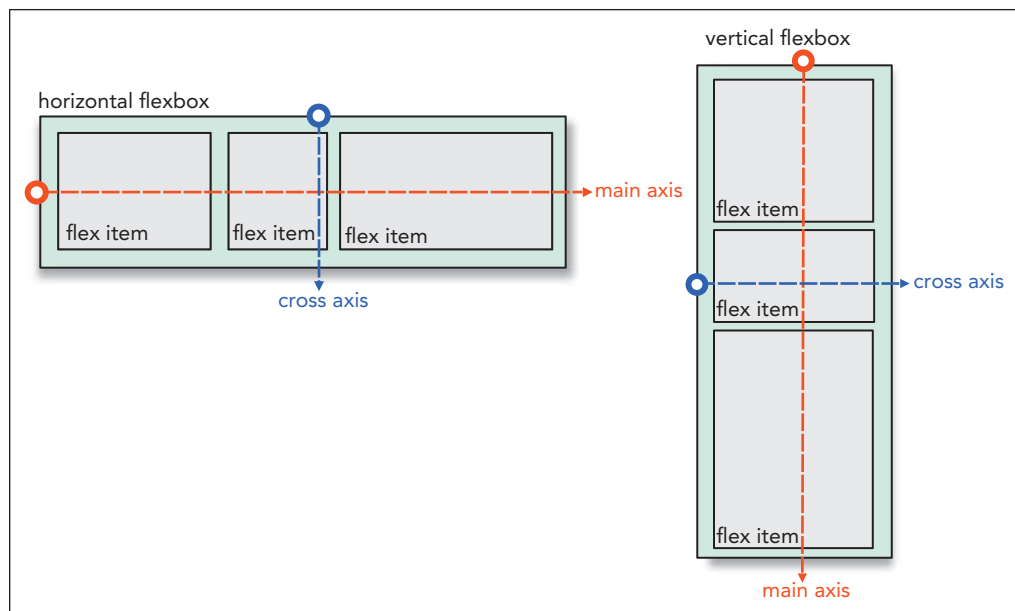
### Defining a Flexible Box

A flexible box or flexbox is a box containing items whose sizes can shrink or grow to match the boundaries of the box. Thus, unlike a grid system in which each item has a defined size, flexbox items adapt themselves automatically to the size of their container. This makes flexboxes a useful tool for designing layouts that can adapt to different page sizes.

Items within a flexbox are laid out along a **main axis**, which can point in either the horizontal or vertical direction. Perpendicular to the main axis is the **cross axis**, which is used to define the height or width of each item. Figure 5–26 displays a diagram of two flexboxes with items arranged either horizontally or vertically along the main axis.

Figure 5–26

Horizontal and vertical flexboxes



To define an element as a flexbox, apply either of the following `display` styles

```
display: flex;
```

or

```
display: inline-flex;
```

where a value of `flex` starts the flexbox on a new line (much as a block element starts on a new line) and a value of `inline-flex` keeps the flexbox in-line with its surrounding content.

## Cross-Browser Flexboxes

The syntax for flexboxes has gone through major revisions as it has developed from the earliest drafts to the latest specifications. Many older browsers employ a different flexbox syntax, in some cases replacing the word *flex* with *box* or *flexbox*. The complete list of browser extensions that define a flexbox would be entered as:

```
display: -webkit-box;
display: -moz-box;
display: -ms-flexbox;
display: -webkit-flex;
display: flex;
```

To simplify the code in the examples that follow, you will limit your code to the W3C specification. This will cover the current browsers at the time of this writing. However, if you need to support older browsers, you may have to include a long list of browser extensions for each flex property.

## Setting the Flexbox Flow

By default, flexbox items are arranged horizontally starting from the left and moving to the right. To change the orientation of the flexbox, apply the following `flex-direction` property

```
flex-direction: direction;
```

where *direction* is `row` (the default), `column`, `row-reverse`, or `column-reverse`. The `row` option lays out the flex items from left to right, `column` creates a vertical layout starting from the top and moving downward, and the `row-reverse` and `column-reverse` options lay out the items bottom-to-top and right-to-left respectively.

Flex items will all try to fit within a single line, either horizontally or vertically. But if they can't, those items can wrap to a new line as needed by applying the following `flex-wrap` property to the flexbox

```
flex-wrap: type;
```

where *type* is either `nowrap` (the default), `wrap` to wrap the flex items to a new line, or `wrap-reverse` to wrap flex items to a new line starting in the opposite direction from the current line. For example, the following style rules create a flexbox in which the items are arranged in a column starting from the top and going down with any flex items that wrap to the second column starting from the bottom and moving up.

```
display: flex;
flex-direction: column;
flex-wrap: wrap-reverse;
```

### TIP

Some older browsers do not support the `flex-flow` property, so for full cross-browser support, you might use the `flex-direction` and `flex-wrap` properties instead.

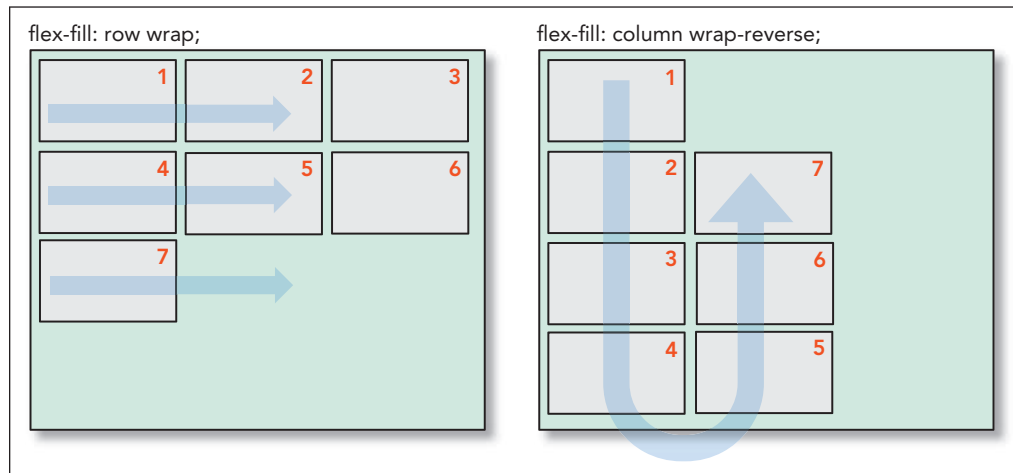
Additional items in this flexbox will continue to follow a snake-like curve with the third column starting at the top, moving down, and so forth.

Both the `flex-direction` and `flex-wrap` properties can be combined into the following `flex-flow` style

```
flex-flow: direction wrap;
```

where *direction* is the direction of the flex items and *wrap* defines whether the items will be wrapped to a new line when needed. Figure 5–27 shows an example of flexboxes laid out in rows and columns in which the flex items are forced to wrap to a new line. Note that the column-oriented flexbox uses `wrap-reverse` to start the new column on the bottom rather than the top.

Figure 5–27 Flexbox layouts



## REFERENCE

### Defining a Flexbox

- To display an element as a flexbox, apply the `display` style
 

```
display: flex;
```
- To set the orientation of the flexbox, apply the style
 

```
flex-direction: direction;
```

 where *direction* is `row` (the default), `column`, `row-reverse`, or `column-reverse`.
- To define whether or not flex items wrap to a new line, apply the style
 

```
flex-wrap: type;
```

 where *type* is either `nowrap` (the default), `wrap` to wrap flex items to a new line, or `wrap-reverse` to wrap flex items to a new line starting in the opposite direction from the current line.
- To define the flow of items within a flexbox, apply the style
 

```
flex-flow: direction wrap;
```

 where *direction* is the direction of the flex items and *wrap* defines whether the items will be wrapped to a new line when needed.

Marjorie wants you to use flexboxes to design a page she's created describing the pre-k classes offered by Trusted Friends. She has already created the content of the page and several style sheets to format the appearance of the page elements. You'll create a style sheet that lays out the page content drawing from a library of flexbox styles.

### To open the pre-k page and style sheet:

1. Use your editor to open the `tf_prek_txt.html` and `tf_flex_txt.css` files from the `html05` ► tutorial folder. Enter **your name** and **the date** in the comment section of each file and save them as `tf_prek.html` and `tf_flex.css` respectively.
2. Return to the `tf_prek.html` file in your editor and, within the document head, create links to the `tf_reset.css`, `tf_styles2.css`, and `tf_flex.css` style sheets in that order.

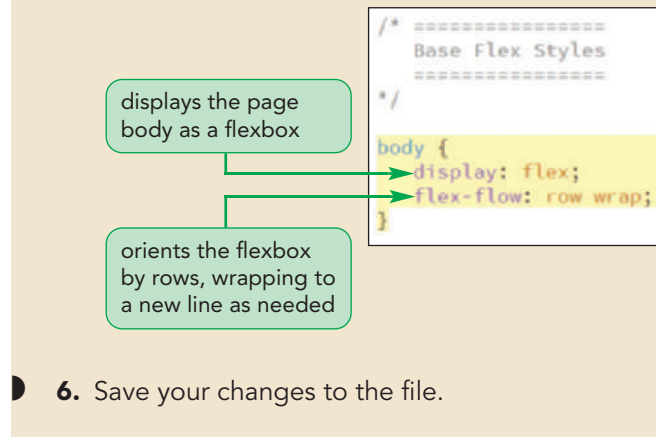
- 3. Take some time to scroll through the contents of the document to become familiar with its contents and structure and then save your changes to the file, leaving it open.
- 4. Go to the **tf\_flex.css** file in your editor.
- 5. Go to the Base Flex Styles section and insert the following style rules to display the entire page body as a flexbox oriented horizontally with overflow flex items wrapped to a new row as needed:

```
body {
    display: flex;
    flex-flow: row wrap;
}
```

Figure 5–28 highlights the new flexbox styles in the style sheet.

Figure 5–28

### Setting the flex display style



- 6. Save your changes to the file.

Now that you’ve defined the page body as a flexbox, you’ll work with styles that define how items within a flexbox expand and contract to match the flexbox container.

## Working with Flex Items

Flex items behave a lot like floated objects though with several advantages, including that you can float them in either the horizontal or vertical direction and that you can change the order in which they are displayed. While the size of a flex item can be fixed using the CSS `width` and `height` properties, they don’t have to be. They can also be “flexed”—automatically adapting their size to fill the flexbox. A flex layout is fundamentally different from a grid layout and requires you to think about sizes and layout in a new way.

### TIP

Because flexboxes can be aligned horizontally or vertically, the `flex-basis` property sets either the initial width or the initial height of the flex item depending on the orientation of the flexbox.

### Setting the Flex Basis

When items are allowed to “flex” their rendered size is determined by three properties: the basis size, the growth value, and the shrink value. The basis size defines the initial size of the item before the browser attempts to fit it to the flexbox and is set using the following `flex-basis` property

```
flex-basis: size;
```

where *size* is one of the CSS units of measurement, a percentage of the size of the flexbox, or the keyword `auto` (the default), which sets the initial size of the flex item based on its content or the value of its `width` or `height` property. For example, the following style rule sets the initial size of the `aside` element to 200 pixels:

```
aside {  
    flex-basis: 200px;  
}
```

The `flex-basis` property should not be equated with the `width` and `height` properties used with grid layouts; rather, it serves only as a starting point. The actual rendered size of the `aside` element in this example is not necessarily 200 pixels but will be based on the size of the flexbox, as well as the size of the other items within the flexbox.

## Defining the Flex Growth

Once the basis size of the item has been defined, the browser will attempt to expand the item into its flexbox. The rate at which a flex item grows from its basis size is determined by the following `flex-grow` property

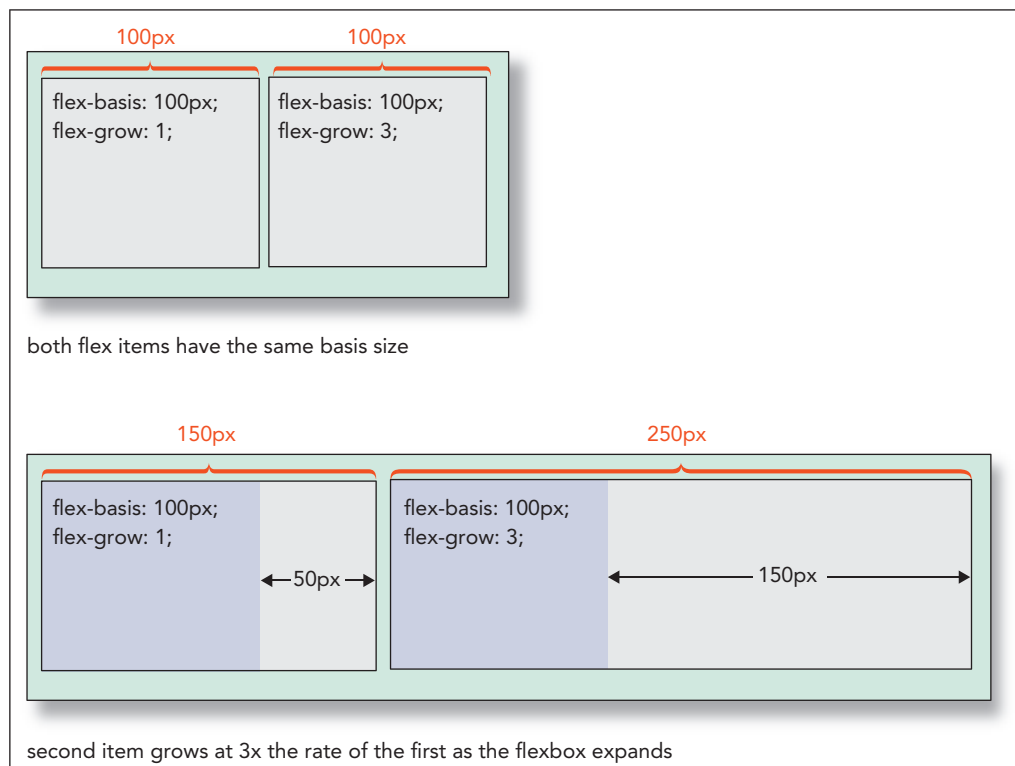
```
flex-grow: value;
```

where *value* is a non-negative value that expresses the growth of the flex item relative to the growth of the other items in the flexbox. The default `flex-grow` value is 0, which is equivalent to not allowing the flex item to grow but to remain at its basis size. Different items within a flexbox can have different growth rates and the growth rate largely determines how much of the flexbox is ultimately occupied by each item.

Figure 5–29 shows an example of how changing the size of a flexbox alters the size of the individual flexbox items.

Figure 5–29

Growing flex items beyond their basis size





In the figure, the basis sizes of the two items are 100 pixels each with the growth of the first item set to 1 and the growth of the second item set to 3. The growth values indicate that as the flex items expand to fill the flexbox, item1 will increase 1 pixel for every 3 pixels that item2 increases. Thus, to fill up the remaining 200 pixels of a 400-pixel wide flexbox, 50 pixels will be allotted to the first item and 150 pixels will be allotted to the second item, resulting in final sizes of 150 pixels and 250 pixels respectively. If the width of the flexbox were to increase to 600 pixels, item1 and item2 will divide the extra 400 pixels once again in a ratio of 1 to 3. Item1 will have a total size of 200 pixels (100px + 100px) and item2 will expand to a size of 400 pixels (100px + 300px).

**TIP**

If all items have `flex-grow` set to 1 and an equal flex basis, they will always have an equal size within the flexbox.

Notice that unlike a grid layout, the relative proportions of the items under a flex layout need not be constant. For the layout shown in Figure 5–29, the two items share the space equally when the flexbox is 200 pixels wide, but at 400 pixels the first item occupies 37.5% of the box while the second item occupies the remaining 62.5%.

To keep a constant ratio between the sizes of the flex items, set their basis sizes to 0 pixels. For example, the following style rules will result in a flexbox in which the first item is always half the size of the second item no matter how wide or tall the flexbox becomes.

```
div#item1 {
  flex-basis: 0px;
  flex-grow: 1;
}
div#item2 {
  flex-basis: 0px;
  flex-grow: 2;
}
```

One of the great advantages of the flexible box layout is that you don't need to know how many items are in the flexbox to keep their relative proportions the same. The following style rule creates a layout for a navigation list in which each list item is assigned an equal size and grows at the same rate.

```
nav ul {
  display: flex;
}
nav ul li {
  flex-basis: 0px;
  flex-grow: 1;
}
```

If there are four items in this navigation list, each will be 25% of the total list size and if at a later date a fifth item is added, those items will then be allotted 20% of the total size. Thus, unlike a grid layout, there is no need to revise the percentages to accommodate new entries in the navigation list; a flexible box layout handles that task automatically.

Note that if the `flex-grow` value is set to 0, the flex item will not expand beyond its basis size, making that basis value the maximum width or height of the item.

## Defining the Shrink Rate

What happens when the flexbox size falls below the total space allotted to its flex items? There are two possibilities depending on whether the flexbox is defined to wrap its contents to a new line. If the `flexbox-wrap` property is set to `wrap`, one or more of the flex items will be shifted to a new line and expanded to fill in the available space on that line. Figure 5–30 shows a flexbox layout in which three items each have a basis size of 200 pixels with the same growth value of 1.

Figure 5–30

## Shrinking flex items smaller than their basis size



As shown in the figure, as long as the flexbox is at least 600 pixels wide, the items will equally share a single row. However, once the flexbox size falls below 600 pixels, the three items can no longer share that row and the last item is wrapped to a new row. Once on that new row, it's free to fill up the available space while the first two items equally share the space on the first row. As the flexbox continues to contract, falling below 400 pixels, the first two items can no longer share a row and the second item now wraps to its own row. At this point the three items fill separate rows and as the flexbox continues to shrink, their sizes also shrink.

If the flexbox doesn't wrap to a new line as it is resized, then the flex items will continue to shrink, still sharing the same row or column. The rate at which they shrink below their basis size is given by the following `flex-shrink` property

```
flex-shrink: value;
```

where *value* is a non-negative value that expresses the shrink rate of the flex item relative to the shrinkage of the other items in the flexbox. The default `flex-shrink` value is 1. For example, in the following style rules, `item1` and `item2` will share the flexbox equally as long as the width of the flexbox is 400 pixels or greater.

```
div {
  display: flex;
  flex-wrap: nowrap;
}
div #item1 {
  flex-basis: 200px;
  flex-grow: 1;
  flex-shrink: 3;
}
```

```
div #item2 {
  flex-basis: 200px;
  flex-grow: 1;
  flex-shrink: 1;
}
```

However, once the flexbox falls below 400 pixels, the two items begin to shrink with item1 losing 3 pixels for every 1 pixel lost by item2. Note that if the `flex-shrink` value is set to 0, then the flex item will not shrink below its basis value, making that basis value the minimum width or height of the item.

## The flex Property

All of the size values described above are usually combined into the following `flex` property

```
flex: grow shrink basis;
```

where *grow* defines the growth of the flex item, *shrink* provides its shrink rate, and *basis* sets the item's initial size. The default `flex` value is

```
flex: 0 1 auto;
```

which automatically sets the size of the flex item to match its content or the value of its `width` and `height` property. The flex item will not grow beyond that size but, if necessary, it will shrink as the flexbox contracts.

The `flex` property supports the following keywords:

- `auto` Use to automatically resize the item from its default size (equivalent to `flex: 1 1 auto;`)
- `initial` The default value (equivalent to `flex: 0 1 auto;`)
- `none` Use to create an inflexible item that will not grow or shrink (equivalent to `flex: 0 0 auto;`)
- `inherit` Use to inherit the flex values of its parent element

As with other parts of the flex layout model, the `flex` property has gone through several syntax changes on its way to its final specification. To support older browsers, use the browser extensions: `-webkit-box`, `-moz-box`, `-ms-flexbox`, `-webkit-flex`, and `flex` in that order.

### REFERENCE

#### Sizing Flex Items

- To set the initial size of a flex item, apply the style

```
flex-basis: size;
```

where *size* is measured in one of the CSS units of measurement or as a percentage of the size of the flexbox or the keyword `auto` (the default).

- To define the rate at which a flex item grows from its basis size, apply the style

```
flex-grow: value;
```

where *value* is a non-negative value that expresses the growth of the flex item relative to the growth of the other items in the flexbox (the default is 0).

- To define the rate at which a flex item shrinks below its basis value, apply

```
flex-shrink: value;
```

where *value* is a non-negative value that expresses the shrink rate of the flex item relative to other items in the flexbox (the default is 0).

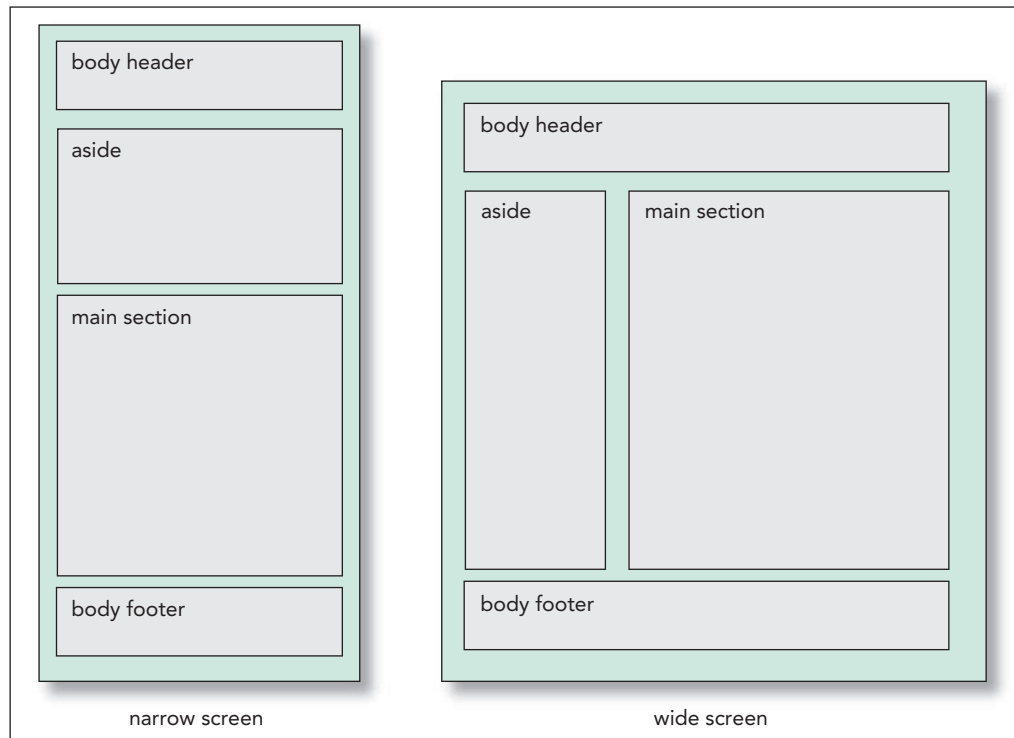
- To define the overall resizing of a flex item, apply

```
flex: grow shrink basis;
```

where *grow* defines the growth of the flex item, *shrink* provides its shrink rate, and *basis* sets the item's initial size.

## Applying a Flexbox Layout

Now that you've seen how to size items within a flexbox, you can return to the layout for the Pre-K Classes page at Trusted Friends Daycare. The `body` element, which you already set up as a flexbox, has four child elements: the page header, an `aside` element describing the daily class schedule, a `section` element describing the classes, and the page footer. Marjorie wants the header and the footer to always occupy a single row at 100% of the width of the page body. For wide screens, she wants the `aside` and `section` elements displayed side-by-side with one-fourth of the width assigned to the `aside` element and three-fourths to the `section` element. For narrow screens, she wants the `aside` and `section` elements displayed within a single column. Figure 5–31 displays the flex layout that Marjorie wants you to apply.

**Figure 5–31****Proposed flex layout for the Pre-K page**

Using the techniques of the first session, this would require media queries with one grid layout for narrow screens and a second grid layout for wide screens. However, you can accomplish the same effect with a single flex layout. First, you set the width of the body header and footer to 100% because they will always occupy their own row:

```
header, footer {  
    width: 100%;  
}
```

Then, you set the basis size of the `aside` and `section` elements to 120 and 361 pixels respectively. As long as the screen width is 481 pixels or greater, these two elements will be displayed side-by-side; however, once the screen width drops below 481 pixels, the elements will wrap to separate rows as illustrated in the narrow screen image in Figure 5–31. Because you want the main `section` element to grow at a rate three times faster than the `aside` element (in order to maintain the 3:1 ratio in their sizes), you set the `flex-growth` values to 1 and 3 respectively. The flex style rules are

```
aside {  
    flex: 1 1 120px;  
}
```

```
section#main {
  flex: 3 1 361px;
}
```

Note that you choose 481 pixels as the total initial size of the two elements to match the cutoff point in the media query between mobile and tablet/desktop devices. Generally, you want your flex items to follow the media query cutoffs whenever possible. Add these style rules to the `tf_flex.css` style sheet now.

### To define the flex layout:

1. Within the `tf_flex.css` file in your editor, add the following style rules to the Base Flex Styles section:

```
header, footer {
  width: 100%;
}

aside {
  flex: 1 1 120px;
}

section#main {
  flex: 3 1 361px;
}
```

Figure 5–32 highlights the newly added style rules to define the flex item sizes.

Figure 5–32

### Set the flex properties of the flex items in the page body

displays the header and footer at a width of 100%, occupying an entire row

sets the initial size of the aside element to 120 pixels and sets the growth and shrink factors to 1

sets the initial size of the main section to 361 pixels and has it grow and shrink at a 3:1 ratio compared to the aside element

```
body {
  display: flex;
  flex-flow: row wrap;
}

header, footer {
  width: 100%;
}

aside {
  flex: 1 1 120px;
}

section#main {
  flex: 3 1 361px;
}
```

2. Save your changes to the file and then open the `tf_prek.html` file in your web browser.
3. Change the size of the browser window or use the device emulator tools in your browser to view the page under different screen widths. As shown in Figure 5–33, the layout of the page changes as the screen narrows and widens.

Figure 5–33

Flex layout under different screen widths



Flexboxes can be nested within one another and a flex item can itself be a flexbox for its child elements. Within the topics section, Marjorie has created six articles describing different features of the center's pre-k curriculum. She wants these articles to share equal space within a row-oriented flexbox, with each article given a basis size of 200 pixels. The style rules are:

```
section#topics {
  display: flex;
  flex-flow: row wrap;
}

section#topics article {
  flex: 1 1 200px;
}
```

Marjorie also wants the items in the navigation list to appear in a row-oriented flexbox for tablet and desktop devices by adding the following style rules to the media query for screen devices whose width exceeds 480 pixels:

```
nav.horizontal ul {
  display: flex;
  flex-flow: row nowrap;
}

nav.horizontal li {
  flex: 1 1 auto;
}
```

The navigation list items will appear in a single row with no wrapping and the width of each item will be determined by the item's content so that longer entries are given more horizontal space. With the growth and shrink values set to 1, each list item will grow and shrink at the same rate, keeping the layout consistent across different screen widths.

Add these style rules now.

### To lay out the topic articles and navigation list:

1. Return to the `tf_flex.css` file in your editor and go to the Base Flex Styles section.
2. Add the following style rules to create a flex layout for the page articles.

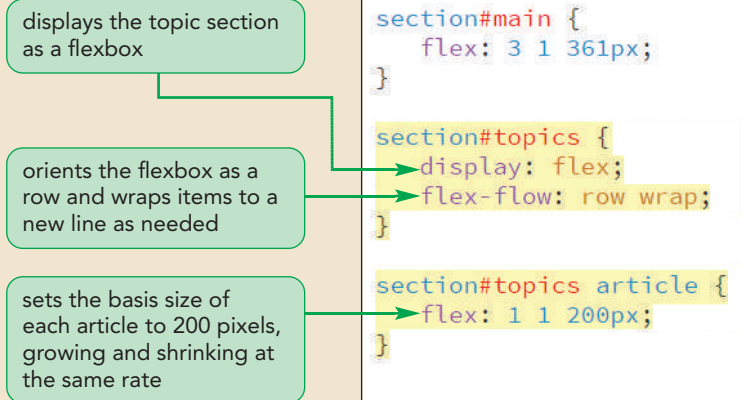
```
section#topics {
  display: flex;
  flex-flow: row wrap;
}

section#topics article {
  flex: 1 1 200px;
}
```

Figure 5–34 highlights the style rules for the article topics layout.

Figure 5–34

### Creating a flex layout for articles in the topics section



3. Scroll down to the media query for tablet and desktop devices and add the following style rule to create a flex layout for the navigation list. (Indent your code to set it off from the media query braces.)

```
nav.horizontal ul {
  display: flex;
  flex-flow: row nowrap;
}

nav.horizontal li {
  flex: 1 1 auto;
}
```

Figure 5–35 highlights the style rules for the navigation list and list items.



Figure 5–35 Creating a flex layout for the navigation list

```

@media only screen and (min-width: 481px) {
  nav.horizontal ul {
    display: flex;
    flex-flow: row nowrap;
  }
  nav.horizontal li {
    flex: 1 1 auto;
  }
}

```

orients the flexbox in the row direction with no wrapping

displays the unordered list as a flexbox

bases the size of each item on its content and has them grow and shrink at the same rate

4. Save your changes to the file and reload the `tf_prek.html` file in your web browser.
5. View the page under different screen widths and verify that, for tablet and desktop screen widths, the navigation list entries appear in a single row. Also, verify that the articles in the topics section flex from a single column layout to two or more rows of content. See Figure 5–36.

Figure 5–36 Flex layout under a desktop screen width

navigation list appears in a single row for tablet and desktop devices

articles flex in layout from a single column to a 2 × 3 grid, depending on the screen width

**trusted friends daycare**

HOME INFANTS TODDLERS PRE-K AFTER SCHOOL

**Schedule**

8:00 am: Arrival and Breakfast  
 9:00 am: Morning Group Time  
 9:25 am: Learning Centers  
 11:00 am: Group Time  
 11:15 am: Outdoor Play  
 11:50 am: Lunch  
 12:30 pm: Group Time  
 12:45 pm: Rest Time  
 2:15 pm: Learning Centers  
 3:00 pm: Group Time  
 3:15 pm: Snack  
 4:15 pm: Learning Centers  
 5:00 pm: Departure and Cleanup

**Pre-K Classes**

As preschoolers gain more self-esteem, they feel ready to take on the world. Our preschool curriculum enhances their confidence by providing activities to help them become problem solvers and nurture a love of learning. With independent exploration, structured activities, and hands-on learning, our children develop a variety of skills and knowledge in areas from mathematics to reading.

We're proud of the work we do. Early learning standards, backed by education experts, inform the scope and sequence of our pre-k program. Our curriculum aligns to 72 learning standards progressing sequentially across six developmental domains. Add to this a healthy dose of running, jumping and movement to keep children active and you'll see why Trusted Friends is a true leader in early childhood education.

**Language Skills**  
 Language, literacy, and communication skills are embedded into a child's daily experiences. We strive to create meaningful classroom experiences that help children use and build a growing vocabulary.

**Math Exploration**  
 Pre-k children are enthusiastic mathematicians. Our instructors work to ensure that our students don't simply learn numbers by rote, but instead build mathematical understanding related to real, everyday problems. Our curriculum includes important skills such as comparing and contrasting items by characteristics, following complex directions in sequence, and solving simple number problems.

**Science Studies**  
 As their cognitive and physical abilities develop, children are able to develop and test their own theories, and to engage in scientific reasoning. Our curriculum encourages students to broaden their understanding of scientific disciplines, from physics to chemistry to earth science. Children learn by participating in cooking projects and participating in multi-skill experiments, handling mechanical tools.

**Creative Expressions**  
 Pre-kindergarten is an ideal time to introduce children to artistic expression. Our pre-k teachers extend their student's skills and knowledge through process-oriented art projects. We teach sculpting with clay and etching tools, writing and illustrating books, and our students experience acting out original stories with costumes, props, and masks.

**Cultural Sharing**  
 Children are innately interested in the diversity of people and cultures. We foster the development of all areas of a child's emotional intelligence including interpersonal skills, compassion, and acceptance of personal responsibility. We believe in fostering respect for different cultures, traditions, and life styles.

**Physical Wellness**  
 Pre-k children learn about becoming responsible for their own choices and decisions. Our curriculum encourages students to learn physical wellness through physical activity, healthy eating, and personal hygiene. Everyday our children learn about themselves and others through supportive sharing times.

Trusted Friends • 3450 Regency Lane, Carmel IN

Marjorie likes how using flexboxes has made it easy to create layouts that match a wide variety of screen sizes. However, she is concerned that under the single column layout used for mobile devices the daily schedule appears first before any description of the classes. She would like the daily schedule to appear at the bottom of the page. She asks if you can modify the layout to achieve this.

## Reordering Page Content with Flexboxes

One of the principles of web page design is to, as much as possible, separate the page content from page design. However, a basic feature of any design is the order in which the content is displayed. Short of editing the content of the HTML file, there is not an easy way to change that order.

That at least was true before flexboxes. Under the flexbox model you can place the flex items in any order you choose using the following `order` property

```
order: value;
```

where *value* is an integer where items with smaller `order` values are placed before items with larger `order` values. For example, the following style arranges the `div` elements starting first with `item2`, followed by `item3`, and ending with `item1`. This is true regardless of how those `div` elements have been placed in the HTML document.

```
div#item1 {order: 100;}
div#item2 {order: -1;}
div#item3 {order: 5;}
```

### TIP

If flex items have the same `order` value, they are arranged in document order.

Note that `order` values can be negative. The default `order` value is 0.

For complete cross-browser support, you can apply the following browser extensions with flex item ordering:

```
-webkit-box-ordinal-group: value;
-moz-box-ordinal-group: value;
-ms-flex-order: value;
-webkit-order: value;
order: value;
```

Most current browsers support the CSS specifications, so you will limit your code to those properties.

### REFERENCE

#### Reordering a Flex Item

- To reorder a flex item, apply the style

```
order: value;
```

where *value* is an integer where items with smaller `order` values are placed before items with larger `order` values.

For mobile devices, Marjorie wants the page header displayed first, followed by the main section, the `aside` element, and ending with the page footer. Add style rules now to the mobile device media query in the `tf_flex.css` style sheet to reorder the flex items.

### To lay out the topic articles and navigation list:

1. Return to the `tf_flex.css` file in your editor and go to the Mobile Devices media query.
2. Add the following style rules, indented to offset them from the braces in the media query:

```
aside {  
    order: 99;  
}  
footer {  
    order: 100;  
}
```

Note that the other flex items will have a default order value of 0 and thus will be displayed in document order before the `aside` and `footer` elements.

Figure 5–37 highlights the style rules to set the order of the `aside` and `footer` elements.

Figure 5–37

### Setting the order of a flex item

places the aside element before the body footer

places the body footer at the end of the flexbox

```
/* =====  
   Mobile Styles: 0 to 480px  
   =====  
*/  
  
@media only screen and (max-width: 480px) {  
  
    aside {  
        order: 99;  
    }  
  
    footer {  
        order: 100;  
    }  
  
}
```

3. Save your changes to the file and then reload the `tf_prek.html` file in your web browser.
4. Reduce the width of the browser window below 480 pixels to show the mobile layout. Verify that the class schedule now appears at the bottom of the file directly before the body footer.

You've completed the ordering and flex layout of the Pre-K Classes page. You'll conclude your review of flexboxes by examining how flex items can be arranged within the flexbox container.

## Exploring Flexbox Layouts

You can control how flex items are laid out using the `justify-content`, `align-items`, and `align-content` properties. You examine each property to see how flexboxes can be used to solve layout problems that have plagued web designers for many years.

### Aligning Items along the Main Axis

Recall from Figure 5–26 that flexboxes have two axes: the main axis along which the flex items flow and the cross axis, which is perpendicular to the main axis. By default, flex items are laid down at the start of the main axis. To specify a different placement, apply the following `justify-content` property

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

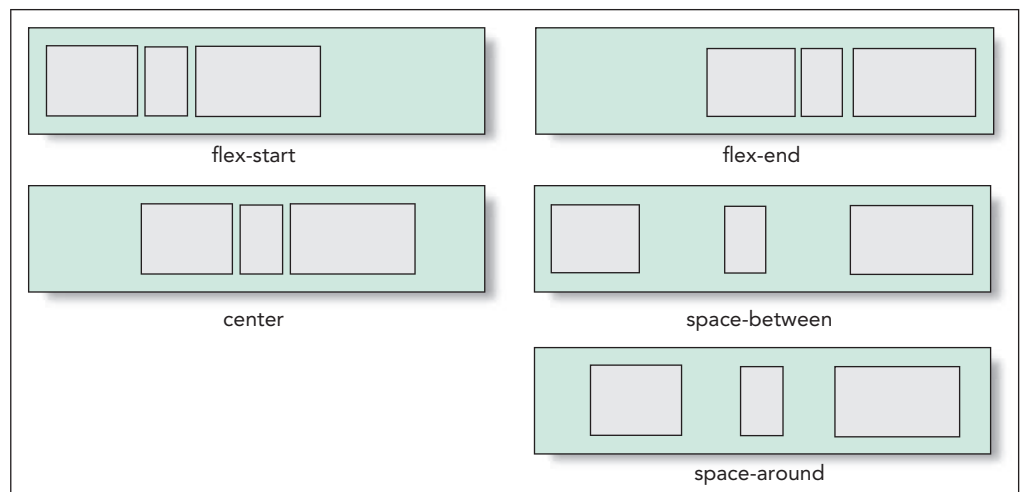
where *placement* is one of the following keywords:

- `flex-start` Items are positioned at the start of the main axis (the default).
- `flex-end` Items are positioned at the end of the main axis.
- `center` Items are centered along the main axis.
- `space-between` Items are distributed evenly with the first and last items aligned with the start and end of the main axis.
- `space-around` Items are distributed evenly along the main axis with equal space between them and the ends of the flexbox.

Figure 5–38 shows the impact of different `justify-content` values on a flexbox oriented horizontally.

Figure 5–38

Values of the `justify-content` property



Remember that, because items can flow in any direction within a flexbox, these diagrams will look different for flexboxes under column orientation or when the content flows from the right to the left. Note that the `justify-content` property has no impact when the items are flexed to fill the entire space. It is only impactful for flex items with fixed sizes that do not fill in the entire flexbox.

## Aligning Flex Lines

The `align-content` property is similar to the `justify-content` property except that it arranges multiple lines of content along the flexbox's cross axis. The syntax of the `align-content` property is:

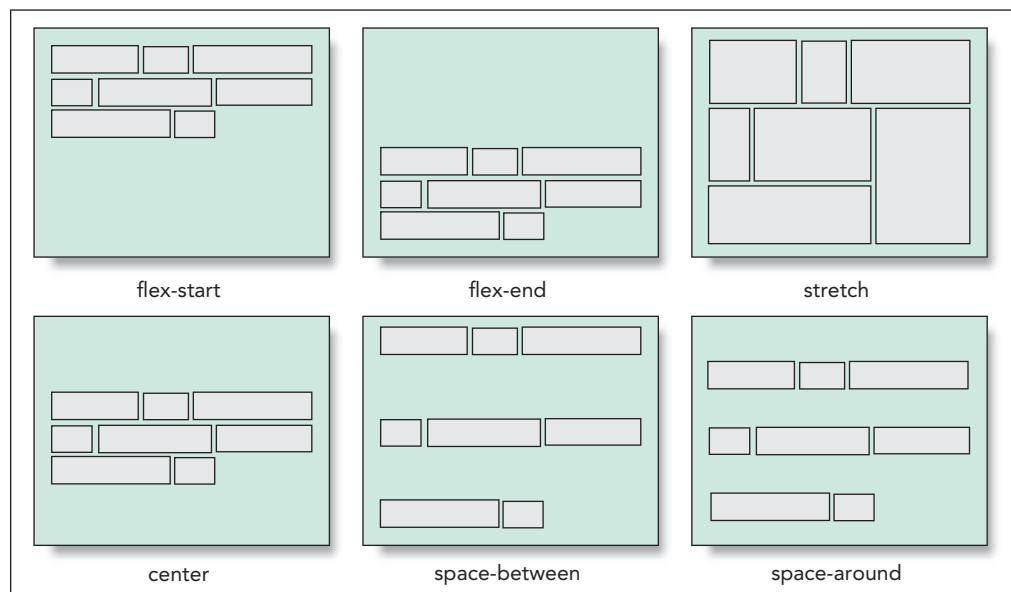
```
align-content: value;
```

where `value` is one of the following keywords:

- `flex-start` Lines are positioned at the start of the cross axis.
- `flex-end` Lines are positioned at the end of the cross axis.
- `stretch` Lines are stretched to fill up the cross axis (the default).
- `center` Lines are centered along the cross axis.
- `space-between` Lines are distributed evenly with the first and last lines aligned with the start and end of the cross axis.
- `space-around` Lines are distributed evenly along the cross axis with equal space between them and the ends of the cross axis.

Figure 5–39 displays the effect of the `align-content` values on three lines of flex items arranged within a flexbox.

Figure 5–39 Values of the `align-content` property



Note that the `align-content` property only has an impact when there is more than one line of flex items, such as occurs when wrapping is used with the flexbox.

## Aligning Items along the Cross Axis

Finally, the `align-items` property aligns each flex item about the cross axis, having the syntax

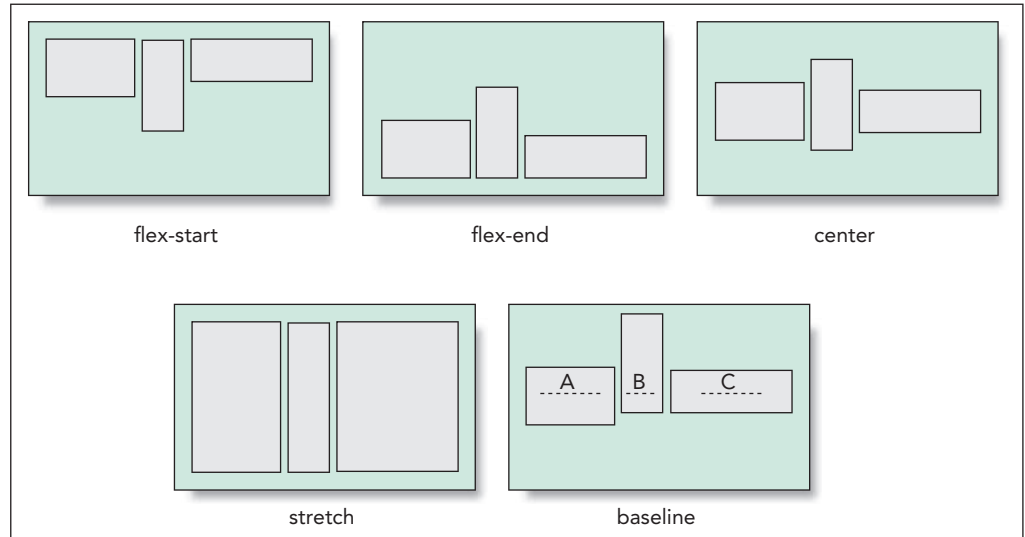
```
align-items: value;
```

where `value` is one of the following keywords:

- `flex-start` Items are positioned at the start of the cross axis.
- `flex-end` Items are positioned at the end of the cross axis.
- `center` Items are centered along the cross axis.
- `stretch` Items are stretched to fill up the cross axis (the default).
- `baseline` Items are positioned so that the baselines of their content align.

Figure 5–40 displays the effect of the `align-items` values on three flex items placed within a single line.

Figure 5–40 Values of the `align-items` property



Note that the `align-items` property is only impactful when there is a single line of flex items. With multiple lines, you use the `align-content` property to layout the flexbox content. To align a single item out of a line of flex items, use the following `align-self` property

```
align-self: value;
```

where `value` is one of the alignment choices supported by the `align-items` property. For example, the following style rule places the footer at the end of the flexbox cross axis, regardless of the placement of the other flex items.

```
footer {  
    align-self: flex-end;  
}
```

Both the `align-content` and `align-items` properties have a default value of `stretch` so that the flex items are stretched to fill the space along the cross-axis. The effect is that all flex items within a row will share a common height. This can be observed earlier in Figure 5–36 in which all of the article boxes have the same height, regardless of their content. It's difficult to achieve this simple effect in a grid layout unless the height of each item is explicitly defined, but flexboxes do it automatically.

### Solving the Centering Problem with Flexboxes

One of the difficult layout challenges in web design is vertically centering an element within its container. While there are many different fixes and “hacks” to create vertical centering, it has not been easily achieved until flexboxes. By using the `justify-content` and `align-items` properties, you can center an object or group of objects within a flexbox container. For example, the following style rule centers the child elements of the `div` element both horizontally and vertically:

```
div {
  display: flex;
  justify-content: center;
  align-content: center;
}
```

For a single object or a group of items on a single line within a container, use the `align-items` property as follows:

```
div {
  display: flex;
  justify-content: center;
  align-items: center;
}
```

You can also use the `align-self` property to center one of the items in the flexbox, leaving the other items to be placed where you wish.

## Creating a Navicon Menu

A common technique for mobile websites is to hide navigation menus but to indicate their presence with a **navicon**, which is a symbol usually represented as three horizontal lines ☰. When the user hovers or touches the icon, the navigation menu is revealed.

Marjorie has supplied you with a navicon image that she wants you to use with the mobile layout of the Pre-K Classes page. Add this image to the Pre-K Classes web page within the navigation list in the body header.

### To insert the navicon image:

1. Return to the `tf_prek.html` file in your editor.
2. Directly after the opening `<nav>` tag in the body header, insert the following hypertext link and inline image.

```
<a id="navicon" href="#">
  
</a>
```

Figure 5–41 highlights the code to create the navicon.




Figure 5-41

## Inserting the navicon

```

<nav class="horizontal">
  <a id="navicon" href="#"></a>
  <ul>
    <li><a href="tf_home.html">Home</a></li>
    <li><a href="#">Infants</a></li>
    <li><a href="#">Toddlers</a></li>
    <li><a href="#" id="currentPage">Pre-K</a></li>
    <li><a href="#">After School</a></li>
  </ul>
</nav>

```



Next, you'll insert the styles to hide and display the contents of the navigation list in a style sheet named `tf_navicon.css`. You'll apply the same styles for `navicon` that you used in the last session to hide and display the navigation submenus in the Trusted Friends home page. As with those menus, you'll use the hover pseudo-class to display the navigation list links whenever the user hovers over the navicon, or in the case of mobile devices, touches the navicon. Add these styles now.

**To add styles for the navicon image:**

1. Within the document head of the `tf_prek.html` file, add a link to the **`tf_navicon.css`** style sheet file after the link for the `tf_flex.css` file. Save your changes to the file.
2. Use your editor to open the **`tf_navicon_txt.css`** files from the `html05` ► tutorial folder. Enter ***your name*** and ***the date*** in the comment section of the file and save it as **`tf_navicon.css`**.
3. By default, the navicon will be hidden from the user. Go to the Base Styles section and add the following style rule:

```

a#navicon {
  display: none;
}

```

4. The navicon will be displayed only for mobile devices. Go to the media query for mobile devices and add the following style rule to display the navicon.

```

a#navicon {
  display: block;
}

```

5. When the navicon is displayed, you want the contents of the navigation list to be hidden. Add the following style rule within the mobile device media query:

```

nav.horizontal ul {
  display: none;
}

```

6. Finally, add the following style rule to the mobile device query that displays the contents of the navigation list when the user hovers over the navicon or the contents of the navigation list.

```
a#navicon:hover+ul, nav.horizontal ul:hover {  
    display: block;  
}
```

Figure 5–42 highlights the style rules for the navicon hypertext link.

**Figure 5–42****Style rules for the navicon image**

```
a#navicon {  
    display: none;  
}  
  
/* =====  
   Mobile Devices: 0 to 480px  
   =====  
*/  
  
@media only screen and (max-width: 480px) {  
  
    a#navicon {  
        display: block;  
    }  
  
    nav.horizontal ul {  
        display: none;  
    }  
  
    a#navicon:hover+ul, nav.horizontal ul:hover {  
        display: block;  
    }  
  
}
```

does not display the navicon for most devices

displays the navicon for mobile devices

hides the navigation list for mobile devices

displays the navigation list when the user hovers over the navicon or moves the mouse pointer over the navigation list

7. Save your changes to the file and then reload the `tf_prek.html` file in your browser or mobile devices. Resize the viewport as needed to display the mobile layout.
8. Verify that as you hover over or touch the navicon, the navigation list appears, as shown in Figure 5–43.

Figure 5-43

## Action of the navicon for mobile devices



BenBois/openciptart

9. Verify that hovering over or touching other parts of the page hides the navigation list.

The methods you used in this tutorial to create pulldown menus and navicon menus represent what you can accomplish when limited to CSS and the hover pseudo-class. As you increase your skill and knowledge of HTML, you'll learn other, more efficient ways of creating mobile navigation menus using program scripts and web frameworks. If you want to explore how to take advantage of these tools, search the web for navicon libraries of prewritten code that can be inserted into your website.



## PROSKILLS

## Written Communication: Speeding Up Your Website by Minifying and Compressing

Once your website is working and you are ready to distribute it to the web, you have one task remaining: minifying your code. **Minifying** refers to the process of removing unnecessary characters that are not required for your site to execute properly. For example, the following text in a CSS file contains comments and line returns and blank spaces, which makes the text easy to read, but these features are not required and have no impact on how the browser renders the page:

```
/* Tablet Styles */

nav.horizontal > ul > li {
  display: block;
}
```

A minified version of this code removes the comment and the extraneous white-space characters leaving the following compact version:

```
nav.horizontal>ul>li{display:block;}
```

Minifying has several important advantages:

- Minifying reduces the amount of bandwidth required to retrieve the website because the files are smaller.
- The smaller minified files load faster and are faster to process because extraneous code does not need to be parsed by the browser.
- A faster site provides a better user experience.
- Smaller files means less server space required to host the website.
- Search engines, such as Google, evaluate your website based on page load speed and will downgrade sites with bloated code that take too long to load.

There are several free tools available on the web to automate the minification process including CSS Minifier, Compress HTML, HTML Minifier, and CSS Compressor. Also, many HTML editors include built-in minifying tools. Remember, a minified file is still a text file and can be read (though with difficulty) in a text editor.

To further reduce your file sizes, consider compressing your files using utilities like Gzip. A compressed file is no longer in text format and must be uncompressed before it is readable. All modern browsers support Gzip compression for files retrieved from a server. Make sure you know how to properly configure your web server to serve Gzip-compressed file in a readable format to the browser.

The process of minifying your files is irreversible, so make sure you retain the version with the text in a readable format and all of your comments preserved. Most minifying and compression tools will make a backup of your original files.

You've completed your work on the design of the Pre-K Classes page for Trusted Friends Daycare. In the next session, you'll explore other uses of media queries by designing a page for printed output. You may close your files now.

### Session 5.2 Quick Check

- Which of the following is *not* a style to display an element as a flexbox?
  - `display: -chrome-flex;`
  - `display: -webkit-flex;`
  - `display: -webkit-box`
  - `display: -ms-flexbox;`
- To display items within a flexbox in a column filled from the bottom upward, use:
  - `flex-direction: column up;`
  - `flex-direction: column-bottom;`
  - `flex-direction: column-reverse;`
  - `flex-direction: column-to-top;`
- To set the initial size of a flexbox item to 250 pixels, use:
  - `flex-size: 250px;`
  - `flex-basis: 250px;`
  - `flex: 250px;`
  - `flex-from: 250px;`
- To set the growth rate of a flexbox item to a rate of 4, use:
  - `flex-rate: 4;`
  - `flex: 4x;`
  - `flex-growth: 4;`
  - `flex-grow: 4;`
- Which of the following sets the `div` element to be equal in size regardless of the size of the flexbox container?
  - `div {flex: equal;}`
  - `div {flex: 1 1 100px;}`
  - `div {flex: 1 1 0px;}`
  - `div {flex: 0 0 0px}`
- To reorder of the placement of a flex item within its flexbox, use:
  - `flex-reorder`
  - `flex-move`
  - `flex-basis;`
  - `order`
- To center flex items along the flexbox's main axis, use:
  - `justify-content: center;`
  - `align-content: center;`
  - `flex-position: center;`
  - `flex-main: center;`
- To center flex items along the flexbox's cross axis, use:
  - `justify-content: center;`
  - `align-content: center;`
  - `flex-position: center;`
  - `flex-main: center;`

# Session 5.3 Visual Overview:

```
nav.horizontal, aside, footer {  
  display: none;  
}
```

The `display` property is set to `none` for objects you don't want printed.

The `@page` rule defines the size and margins of the printed page.

```
@page {  
  size: 8.5in 11in portrait;  
  margin: 0.5in;  
}
```

For print layouts, fonts should be sized in points and widths and heights expressed in inches or centimeters.

```
h1 {  
  font-size: 28pt;  
  line-height: 30pt;  
  margin: 0.3in 0in 0.2in;  
}
```

Use the `after` pseudo-element along with the `content` property to display the text of all hypertext URLs.

```
a::after {  
  content: " (" attr(href) ") ";  
  font-weight: bold;  
  word-wrap: break-word;  
}
```

Use the `page-break-inside` property to prohibit page breaks within an element.

```
article:nth-of-type(n+2) {  
  page-break-before: always;  
}
```

Use the `page-break-before` property to insert page breaks before elements.

Use the `widows` property to limit the number of lines stranded at the top of a page.

```
img, ol, ul {  
  page-break-inside: avoid;  
}
```


```
p {  
  orphans: 3;  
  widows: 3;  
}
```

Use the `orphans` property to limit the number of lines stranded at the bottom of a page.


# Print Styles

Page size is set at 8.5 inches by 11 inches with a 0.5 inch margin in portrait orientation.

Trusted Friends: Article of Interest



## An Accredited Center



Pressmaster/Shutterstock.com

At Trusted Friends we believe that every child is capable of excellence. That is why we are committed to pursuing and maintaining our status as an accredited daycare center. By seeking national accreditation, you know that Trusted Friends is striving to give your family the very best daycare experience.

### What is Accreditation?

Every daycare center must meet the state's minimum license requirements. We go beyond that. When a daycare center is awarded national accreditation they are meeting a higher standard that demonstrates its expertise in:

- Classroom Management
- Curriculum Development
- Health and Safety
- Parental Support
- Community Involvement
- Teacher Certification
- Administrative Oversight
- Financial Statements

18

page 1

Page break is not allowed inside the unordered list.

Trusted Friends: Article of Interest

Our commitment to accreditation gives you assurance we provide a positive educational experience for your child.

### How does Accreditation Work?

Every other year we go through an intense review by recognized and esteemed national accreditation agencies. Their positive reports (available for inspection) confirm that we are providing a clean, safe, and positive environment for our children. Accreditation verifies that our teachers are qualified and fully engaged in giving our children a first-class educational experience.

Once we've completed the entire accreditation self-study process, trained professionals from our accrediting agencies conduct on-site visits to validate our compliance with national early childhood education standards. That accreditation doesn't just take place every two years. It's an ongoing process of self-evaluation, discussion, and parental reviews.

We encourage parents to help us improve our center and become better stewards for their children. You can part of the accreditation process as we work together to make Trusted Friends a great neighborhood center.

### Who Provides Accreditation?

There are several national organizations that provide accreditation services. Who a center chooses for oversight is important. Trusted Friends pursues national accreditation from three of the most respected national early childhood accreditation agencies:

- National Association for Youth Care (<http://www.example.com/nayc>)
- United Accreditation for Early Care and Education (<http://www.example.com/uaece>)
- National Daycare Accreditation (<http://www.example.com/nda>)

Feel free to contact us to discuss accreditation and learn more about our standards for care and education.


19

page 2

Hypertext URLs are displayed in bold after the hypertext link.

Trusted Friends: Article of Interest

## Our Community



Gladskikh Tatiana/Shutterstock.com

Trusted Friends is committed to improving the lives of children in our community. Our expertise in caring for the children at our daycare center gives us a unique understanding of child development, education issues, and parenting. Trusted Friends has partnered with several community organizations that advocate for poor and needy children and families.

We don't think of it as charity. It's part of our calling.

### Improving Literacy

Part of Trusted Friends' mission is to promote literacy, which is key to education and a fulfilling life. We support reading programs and national literacy efforts initiated at both the local and national level. These efforts include providing early access to books and other reading material. We are also in the Raised by Reading (<http://www.example.com/rbr>) program, helping parents share the reading experience with their children.

### Promoting Partnerships

We are proud of our support for the Big Siblings (<http://www.example.com/bis>) organization. Several of our educators are Big Sibling mentors and we provide meeting space and monthly activities for this fine group. We are also deeply involved with the Young Care Nursery (<http://www.example.com/ycn>) organization, working to prevent child abuse and neglect. We partner with other caregivers committed to strengthening families in the community. For example we are a charter member of Sunflower Friends (<http://www.example.com/sf>), which creates learning and enrichment opportunities for underprivileged children, helping them to realize their potential and recognize their inherent dignity.

Please contact us if you believe that Trusted Friends can be a partner with your group in improving the lives of children and families in our community.

20

page 3

Page break is inserted before the article element, starting it on a new page.



## Designing for Printed Media

So far your media queries have been limited to screens of different widths. In this session you'll explore how to apply media queries to print devices and work with several CSS styles that apply to printed output. To do this you'll create a **print style sheet** that formats the printed version of your web document.

### Previewing the Print Version

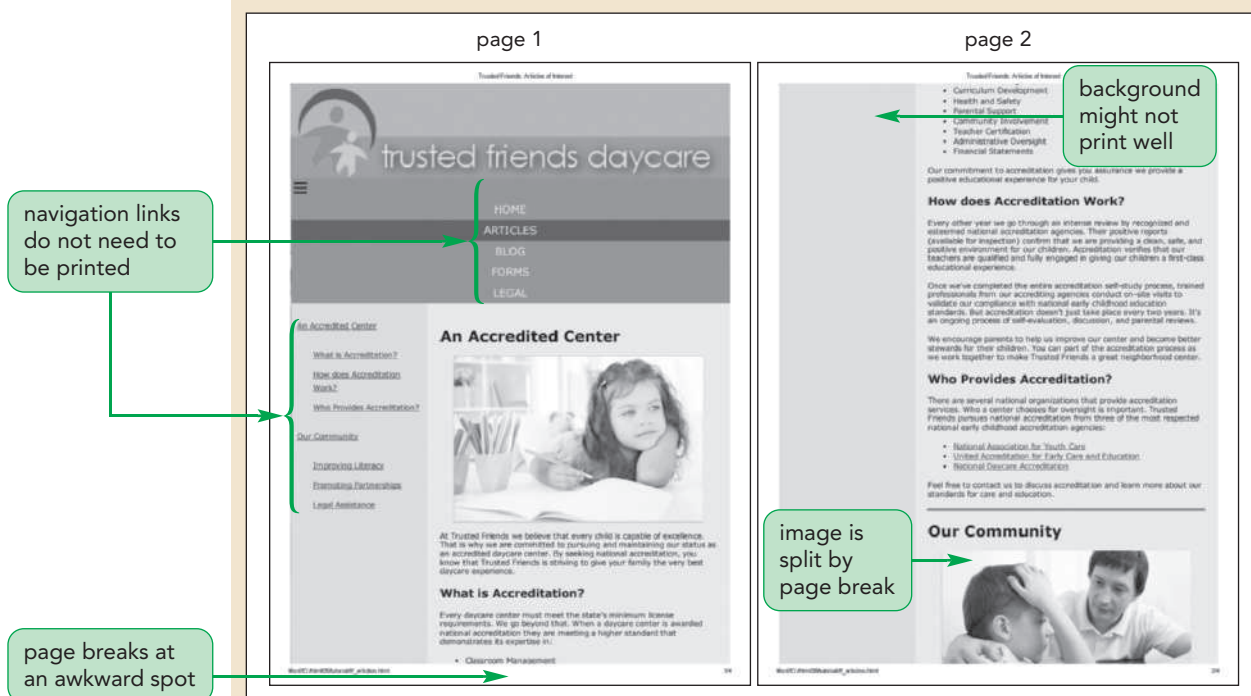
Marjorie has created a page containing articles of interest for parents at Trusted Friends Daycare. She has already written the page content and the style sheets for mobile, tablet, and desktop devices. Open the articles document now.

#### To open the Articles of Interest page:

1. Use your editor to open the **tf\_articles\_txt.html** file from the html05 ► tutorial folder. Enter **your name** and **the date** in the comment section of the file and save it as **tf\_articles.html**.
  2. Within the document head, create links to the **tf\_reset.css** and **tf\_styles3.css** style sheet files in that order.
  3. Scroll through the document to become familiar with its contents and then save your changes to file, but do not close it.
  4. Open the **tf\_articles.html** file in your web browser.
  5. Take some time to view the contents of the page under different screen resolutions, noting how Marjorie has used responsive design to create different page layouts based on the screen width.
- Now, you'll examine how Marjorie's page will appear when printed.
6. Use the Print Preview command within your browser to preview how this page will appear when printed. Figure 5–44 shows a preview of the first two pages of the print version using a black and white printer.

Figure 5–44

Print version of the Articles of Interest page



**Trouble?** Depending on your browser and printer, your print preview might appear different from the preview shown in Figure 5–44.

Browsers support their own internal style sheet to format the print versions of the web pages they encounter. However, their default styles might not always result in the best printouts. Marjorie points out that the print version of her page has several significant problems:

- The printed version includes two navigation lists, neither of which have a purpose in a printout.
- Page breaks have been placed in awkward places, splitting paragraphs and images in two.
- Background colors, while looking good on a screen, might not print well.

Marjorie would like you to design a custom print style sheet that fixes these problems by removing unnecessary page elements and choosing page breaks more intelligently.

## Applying a Media Query for Printed Output

To apply a print style sheet, you use the `media` attribute in your `link` elements to target style sheets to either screen devices or print devices. Modify the `tf_articles.html` file now to access a new style sheet named `tf_print.css` into which you include your print styles.

### To access a print style sheet:

1. Use your editor to open the `tf_print_txt.css` file from the `html05` ► tutorial folder. Enter **your name** and **the date** in the comment section and save it as `tf_print.css`.
2. Return to the `tf_articles.html` file in your editor. Add the attribute `media="all"` to the `link` element for the `tf_reset.css` style sheet to apply it to all devices.
3. Add the attribute `media="screen"` to the `link` element for the `tf_styles3.css` style sheet to apply it only to screen devices.
4. Add the following `link` element for print styles:

```
<link href="tf_print.css" rel="stylesheet" media="print" />
```

Figure 5–45 highlights the revised `link` elements in the file.

To avoid mixing screen styles with print styles, identify styles common to both devices with the media type `all`.

Figure 5–45

### Style sheets for different devices

```
<title>Trusted Friends: Articles of Interest</title>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1" />
<link href="tf_reset.css" rel="stylesheet" media="all" />
<link href="tf_styles3.css" rel="stylesheet" media="screen" />
<link href="tf_print.css" rel="stylesheet" media="print" />
</head>
```

styles for print devices

styles for screen devices

5. Save your changes to the file and close it.

You'll start designing the print version of this page by hiding those page elements that should not be printed, including the navigation list, the `aside` element, and the body footer.

### To hide elements in the print version:

1. Return to the `tf_print.css` file in your editor.
2. Go to the Hidden Objects section and add the following style rule:

```
nav.horizontal, aside, footer {  
    display: none;  
}
```

Figure 5–46 highlights the style rule to hide page elements.

Figure 5–46

### Hiding page elements for printing

sets the display of the navigation list, aside element, and body footer to do not display

```
/* Hidden Objects */  
nav.horizontal, aside, footer {  
    display: none;  
}
```

3. Save your changes to the file and then reload the `tf_articles.html` file in your browser and preview the printed output. Verify that the navigation lists, `aside` elements, and body footer are not displayed in the printed version.

Next, you'll define the page size of the print version of this document.

## Working with the @page Rule

In CSS every printed page is defined as a **page box**, composed of two areas: the **page area**, which contains the content of the document, and the **margin area**, which contains the space between the printed content and the edges of the page.

Styles are applied to the page box using the following `@page` rule

```
@page {  
    style rules  
}
```

where *style rules* are the styles applied to the page. The styles are limited to defining the page size and the page margin. For example, the following `@page` rule sets the size of the page margin to 0.5 inches:

```
@page {  
    margin: 0.5in;  
}
```

The page box does not support all of the measurement units you've used with the other elements. For example, pages do not support the `em` or `ex` measurement units. In general, you should use measurement units that are appropriate to the dimensions of your page, such as inches or centimeters.

## Setting the Page Size

Because printed media can vary in size and orientation, the following `size` property allows web authors to define the dimensions of the printed page

```
size: width height;
```

### TIP

Users can override the page sizes and orientations set in `@page` rule by changing the options in their print dialog box.

where `width` and `height` are the width and height of the page. Thus to define a page that is 8.5 inches wide by 11 inches tall with a 1-inch margin, you would apply the following style rule:

```
@page {  
    size: 8.5in 11in;  
    margin: 1in;  
}
```

You can replace the `width` and `height` values with the keyword `auto` (to let browsers determine the page dimensions) or `inherit` (to inherit the page size from the parent element). If a page does not fit into the dimensions specified in the `@page` rule, browsers will either rotate the page or rescale it to fit within the defined page size.

## Using the Page Pseudo-Classes

By default, the `@page` rule is applied to every page of the printed output. However, if the output covers several pages, you can define different styles for different pages by adding the following pseudo-class to the `@page` rule:

```
@page:pseudo-class {  
    style rules  
}
```

where `pseudo-class` is `first` for the first page of the printout, `left` for the pages that appear on the left in double-sided printouts, or `right` for pages that appear on the right in double-sided printouts. For example, if you are printing on both sides of the paper, you might want to create mirror images of the margins for the left and right pages of the printout. The following styles result in pages in which the inner margin is set to 5 centimeters and the outer margin is set to 2 centimeters:

```
@page:left {margin: 3cm 5cm 3cm 2cm;}  
@page:right {margin: 3cm 2cm 3cm 5cm;}
```

## Page Names and the Page Property

To define styles for pages other than the first, left, or right, you first must create a page name for those styles as follows

```
@page name {  
    style rules  
}
```

where `name` is the label given to the page. The following code defines a page style named `wideMargins` used for pages in which the page margin is set at 10 centimeters on every side:

```
@page wideMargins {  
    margin: 10cm;  
}
```

Once you define a page name, you can apply it to any element in your document. The content of the element will appear on its own page, with the browser automatically

inserting page breaks before and after the element if required. To assign a page name to an element, you use the following `page` property

```
selector {
  page: name;
}
```

where `selector` identifies the element that will be displayed on its own page, and `name` is the name of a previously defined page style. Thus the following style rule causes all block quotes to be displayed on separate page(s) using the styles previously defined as the `wideMargins` page:

```
blockquote {
  page: wideMargins;
}
```

## REFERENCE

### Creating and Applying Page Styles

- To define a page box for the printed version of a document, use the CSS rule

```
@page {
  size: width height;
}
```

where `width` and `height` are the width and height of the page.

- To define the page styles for different output pages, use the rule

```
@page:pseudo-class {
  style rules
}
```

where `pseudo-class` is `first` for the first page of the printout, `left` for the pages that appear on the left in double-sided printouts, or `right` for pages that appear on the right in double-sided printouts.

- To create a named page for specific page styles, apply the rule

```
@page name {
  style rules
}
```

where `name` is the label assigned to the page style.

- To apply a named page style, use the rule

```
selector {
  page: name;
}
```

where `selector` identifies the element that will be displayed on its own page, and `name` is the name of a previously defined page style.

You'll use the `@page` rule to define the page size for the printed version of the Articles of Interest document. Marjorie suggests that you set the page size to 8.5 × 11 inches with 0.5-inch margins.

### To define the printed page size:

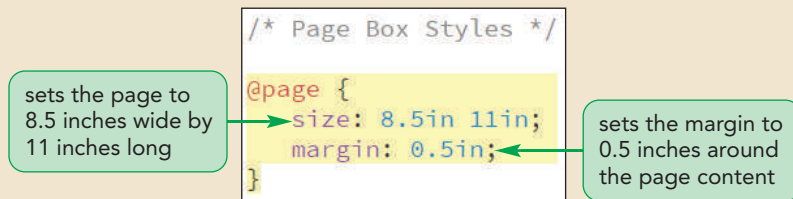
1. Return to the `tf_print.css` file in your editor.
2. Go to the Page Box Styles section and add the following rule:

```
@page {  
    size: 8.5in 11in;  
    margin: 0.5in;  
}
```

Figure 5–47 highlights the rule to set the page size.

Figure 5–47

### Setting the page size



3. Save your changes to the file.

With printed output, widths and heights are measured not in pixels but in inches or centimeters. Font sizes are not measured in pixels but rather in points. With that in mind, create styles to format the sizes of the text and graphics on the page.

### To format the printed text:

1. Go to the Typography Styles section and insert the following styles to format the appearance of h1 and h2 headings and paragraphs:

```
h1 {  
    font-size: 28pt;  
    line-height: 30pt;  
    margin: 0.3in 0in 0.2in;  
}  
  
h2 {  
    font-size: 20pt;  
    margin: 0.1in 0in 0.1in 0.3in;  
}  
  
p {  
    font-size: 12pt;  
    margin: 0.1in 0in 0.1in 0.3in;  
}
```

2. Within the List Styles section, add the following style rules to format the appearance of unordered lists:

```
ul {  
    list-style-type: disc;  
    margin-left: 0.5in;  
}
```

Figure 5–48 shows the typography and list styles in the print style sheet.

Figure 5–48

### Typographical formats

The diagram shows a CSS code block with the following content:

```

/* Typography Styles */
h1 {
  font-size: 28pt;
  line-height: 30pt;
  margin: 0.3in 0in 0.2in;
}
h2 {
  font-size: 20pt;
  margin: 0.1in 0in 0.1in 0.3in;
}
p {
  font-size: 12pt;
  margin: 0.1in 0in 0.1in 0.3in;
}

/* List Styles */
ul {
  list-style-type: disc;
  margin-left: 0.5in;
}

```

Callouts explain the following:

- font sizes are measured in points (points to `28pt`, `30pt`, and `12pt`)
- format of h1 headings (points to the `h1` selector)
- format of h2 headings (points to the `h2` selector)
- format of paragraphs (points to the `p` selector)
- format of unordered lists (points to the `ul` selector)
- margins are measured in inches (points to the `margin` property values)

Next, you'll format the appearance of images on the page.

### To format the printed images:

1. Within the Image Styles section, add the following style rule to format the appearance of inline images within each `article` element:

```

article img {
  border: 2px solid rgb(191, 191,191);
  display: block;
  margin: 0.25in auto;
  width: 65%;
}

```

Figure 5–49 shows the style rule for inline images on the printed page.

Figure 5–49

### Image formats

The diagram shows a CSS code block with the following content:

```

/* Image Styles */
article img {
  border: 2px solid rgb(191, 191,191);
  display: block;
  margin: 0.25in auto;
  width: 65%;
}

```

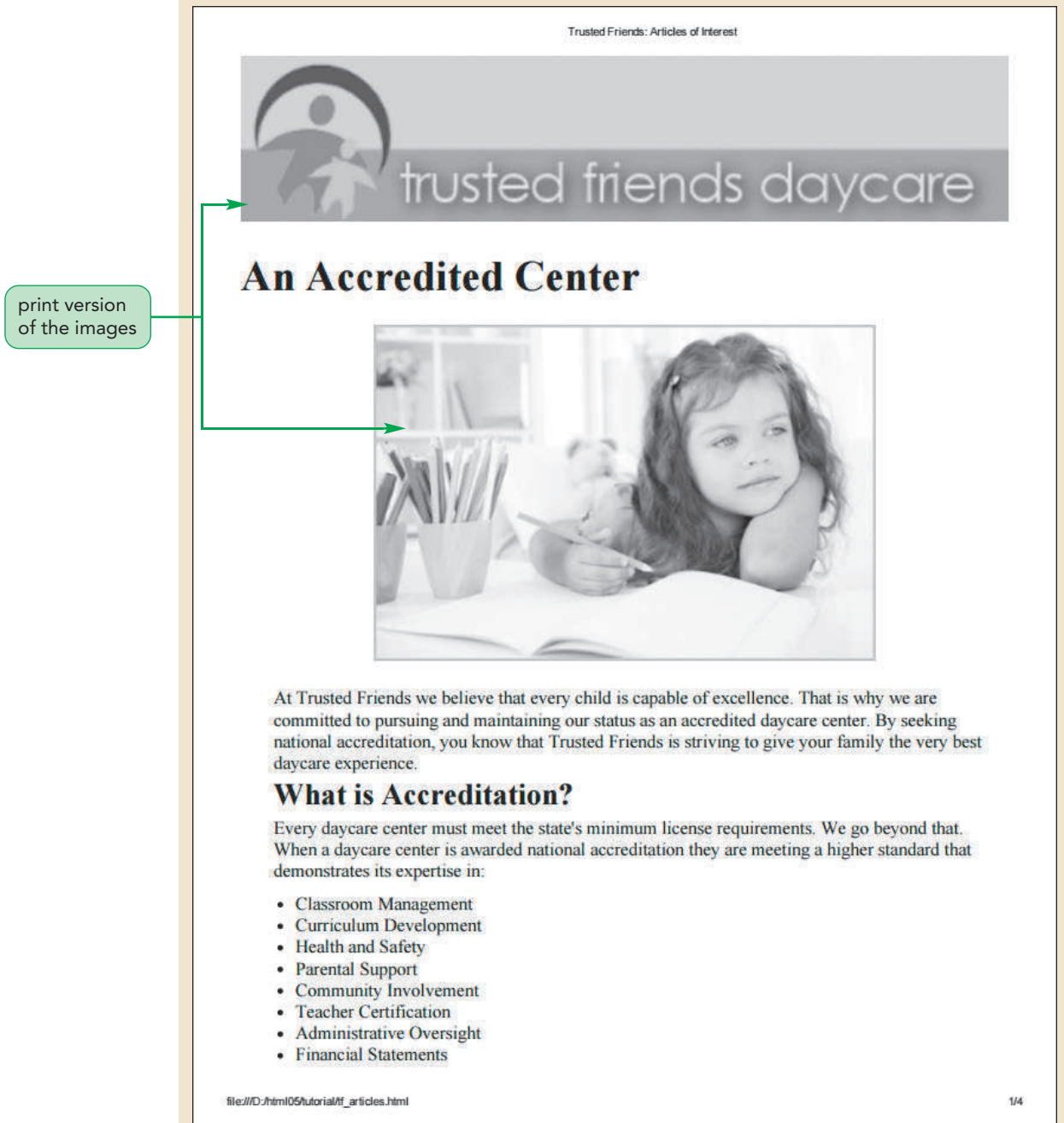
A callout box explains: "displays all article images with a gray border, with a width of 65% of the page body, and centered horizontally" (points to the `margin: 0.25in auto;` and `width: 65%;` lines).

2. Save your changes to the style sheet and then reload the `tf_articles.html` file in your browser and preview the appearance of the printed page. Figure 5–50 shows the appearance of the first page printed using a black and white printer.



Figure 5–50

Preview of the first printed page



© Pressmaster/Shutterstock.com

Marjorie notices that all of the hyperlinks in the document appear in blue and underlined as determined by the default browser style. While this identifies the text as a hypertext link, it doesn't provide the reader any information about that link. She asks you to modify the style sheet to fix this problem.

## Formatting Hypertext Links for Printing

Because printouts are not interactive, it's more useful for the reader to see the URL of a hypertext link so that he or she can access that URL at another time. To append the text of a link's URL to the linked text, you can apply the following style rule:

```
a::after {
  content: " (" attr(href) ") ";
}
```

### TIP

Be sure to include blank spaces around the href value so that the URL does not run into the surrounding text.

This style rule uses the `after` pseudo-element along with the `content` property and the `attr()` function to retrieve the text of the `href` attribute and add it to the contents of the `a` element.

You should be careful when using this technique. Appending the text of a long and complicated URL will make your text difficult to read and might break your page layout if the text string extends beyond the boundaries of its container. One way to solve this problem is to apply the following `word-wrap` property to the URL text:

```
word-wrap: type;
```

where `type` is either `normal` (the default) or `break-word`. A value of `normal` breaks a text string only at common break points such as the white space between words. A value of `break-word` allows long text to be broken at arbitrary points, such as within a word, if that is necessary to make the text string fit within its container. Because a URL has no common break points such as blank spaces, applying the `break-word` option ensures that the text string of the URL will be kept to a manageable length by breaking it as needed to fit within the page layout.

### Formatting Hypertext for Printing

- To add the URL after a hypertext link, apply the style rule:

```
a::after {
  content: " (" attr(href) ") ";
}
```

- To automatically wrap the text of long URLs as needed, add the following style to the link text:

```
word-wrap: break-word;
```

Format the appearance of hypertext links in the document to display each link's URL and to display the hypertext links in a black bold font with no underlining, then use the `word-wrap` property to keep long URLs from extending beyond the boundaries of their container.

## To format the hypertext links:

1. Return to the **tf\_print.css** file in your editor and go to Hypertext Styles section, inserting the following styles to format the appearance of all hypertext links, appending the URL of each link:

```
a {
  color: black;
  text-decoration: none;
}

a::after {
  content: " (" attr(href) ") ";
  font-weight: bold;
  word-wrap: break-word;
}
```

Figure 5–51 describes the style rules used to format printed hypertext links.

Figure 5–51

### Formatting printed hypertext links

```
/* Hypertext Styles */
a {
  color: black;
  text-decoration: none;
}
a::after {
  content: " (" attr(href) ") ";
  font-weight: bold;
  word-wrap: break-word;
}
```

displays hypertext links in black with no underlining

adds the URL of the hypertext link in a bold font

allows the URL to wrap in order to preserve page layout

2. Save your changes to the style sheet and then reload the **tf\_articles.html** file in your browser and preview the page printout. Figure 5–52 shows the appearance of the printed hypertext links found on the second page of Marjorie’s printout.

Figure 5–52

Preview of the hypertext links on page 2

Trusted Friends: Articles of Interest

## How does Accreditation Work?

Every other year we go through an intense review by recognized and esteemed national accreditation agencies. Their positive reports (available for inspection) confirm that we are providing a clean, safe, and positive environment for our children. Accreditation verifies that our teachers are qualified and full engaged in giving our children a first-class educational experience.

Once we've completed the entire accreditation self-study process, trained professionals from our accrediting agencies conduct on-site visits to validate our compliance with national early childhood education standards. But accreditation doesn't just take place every two years. It's an ongoing process of self-evaluation, discussion, and parental reviews. We encourage our parents to help us improve our center and become better stewards for their children.

## Who Provides Accreditation?


Trusted Friends pursues national accreditation from three of the most respected national early childhood accreditation agencies:

- National Association for Youth Care (<http://www.example.com/nayc>)
- United Accreditation for Early Care and Education (<http://www.example.com/uaece>)
- National Daycare Accreditation (<http://www.example.com/nda>)

Feel free to contact us to discuss accreditation and learn more about our standards for care and education.

URL of each hypertext link

## Our Community



Trusted Friends is committed to improving the lives of children in our community. Our expertise in caring for the children at our daycare center gives us a unique understanding of child development, education issues, and parenting. Trusted Friends has partnered with several community organizations that advocate for poor and needy children and families. We don't think of it as charity. It's part of our calling.

2/3

© Gladskikh Tatiana/Shutterstock.com

You can search the web for several free scripting tools that give you more options for how your URLs should be printed, including scripts that automatically append all URLs as footnotes at the end of the printed document.

## Working with Page Breaks

When a document is sent to a printer, the browser determines the location of the page breaks unless that information is included as part of the print style sheet. To manually insert a page break either directly before or directly after an element, apply the following `page-break-before` or `page-break-after` properties:

```
page-break-before: type;
page-break-after: type;
```

where `type` has the following possible values:

- `always` Use to always place a page break before or after the element
- `avoid` Use to never place a page break
- `left` Use to place a page break where the next page will be a left page
- `right` Use to place a page break where the next page will be a right page
- `auto` Use to allow the printer to determine whether or not to insert a page break
- `inherit` Use to insert the page break style from the parent element

For example, if you want each `h1` heading to start on a new page, you would apply the following style rule to insert a page break before each heading:

```
h1 {
  page-break-before: always;
}
```

### REFERENCE

#### Adding a Page Break

- To set the page break style directly before an element, apply the property

```
page-break-before: type;
```

where `type` is `always`, `avoid`, `left`, `right`, `auto`, or `inherit`.

- To set the page break style directly after an element, apply

```
page-break-after: type;
```

After the first article, Marjorie wants each subsequent article to start on a new page. To select every article after the initial article, use the selector

```
article:nth-of-type(n+2)
```

which selects the second, third, fourth, and so on `article` elements in the document (see “Exploring the `nth-of-type` Pseudo-class” in Tutorial 2.) To ensure that each of the selected articles starts on a new page, insert the page break before the article using the following style rule:

```
article:nth-of-type(n+2) {
  page-break-before: always;
}
```

Add this style rule to the print style sheet now.

### To print each article on a new page:

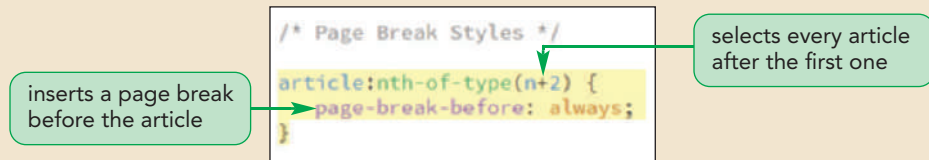
1. Go to the Page Break Styles section and insert the following style rule:

```
article:nth-of-type(n+2) {
    page-break-before: always;
}
```

Figure 5–53 highlights the style rule to insert the article page breaks.

Figure 5–53

### Adding page breaks before the document articles



2. Save your changes to the file and then reload the `tf_articles.html` file in your browser and preview the printed page. Verify that the second article in the document on Community Involvement starts on a new page.

Next, you'll explore how to remove page breaks from the printed version of your web page.

## INSIGHT

### How Browsers Set Automatic Page Breaks

Browsers establish page breaks automatically, unless you manually specify the page breaks with a print style sheet. By default, browsers insert page breaks using the following guidelines:

- Insert all of the manual page breaks as indicated by the `page-break-before`, `page-break-after`, and `page-break-inside` properties
- Break the pages as few times as possible
- Make all pages that don't have a forced page break appear to have the same height
- Avoid page breaking inside page elements that have a border
- Avoid page breaking inside a web table
- Avoid page breaking inside a floating element

Other styles from the print style sheet are applied only after attempting to satisfy these constraints. Note that different browsers apply page breaks in different ways, so while you can apply general rules to your print layout, you cannot, at the current time, make the print versions completely consistent across browsers.

### Preventing Page Breaks

You can prevent a page break by using the keyword `avoid` in the `page-break-after` or `page-break-before` properties. For example, the following style rule prevents page breaks from being added after any heading.

```
h1, h2, h3, h4, h5, h6 {
    page-break-after: avoid;
}
```

Unfortunately in actual practice, most current browsers don't reliably support prohibiting page breaks in this fashion. Thus, to prevent page breaks after an element, you will usually have to manually insert a page break before the element so that the element is moved to the top of the next page.

For other print layouts, you will want to prevent page breaks from being placed inside an element. This usually occurs when you have a long string of text that you don't want broken into two pages. You can prevent printers from inserting a page break by using the following `page-break-inside` property

```
page-break-inside: type;
```

where `type` is `auto`, `inherit`, or `avoid`. Thus, to prevent a page break from appearing within any image you can apply the following style rule:

```
img {
  page-break-inside: avoid;
}
```

Unlike the `page-break-before` and `page-break-after` properties, almost all current browsers support the use of the `avoid` keyword for internal page breaks.

**REFERENCE**

### Preventing Page Breaks Inside an Element

- To prevent a page break from occurring within an element, apply the style:

```
page-break-inside: avoid;
```

Marjorie asks you to revise the print style sheet to prevent page breaks from occurring within images, ordered lists, and unordered lists.

#### To avoid page breaks:

1. Return to the `tf_print.css` file in your editor and go to the Page Break Styles section and insert the following style rule:

```
img, ol, ul {
  page-break-inside: avoid;
}
```

Figure 5–54 highlights the style rule to avoid page breaks in lists and images.

Figure 5–54

#### Avoiding line breaks within lists and images

avoids line breaks within lists and images

```
/* Page Break Styles */
article:nth-of-type(n+2) {
  page-break-before: always;
}
img, ol, ul {
  page-break-inside: avoid;
}
```

2. Save your changes to the file.



Note that the `avoid` type does not guarantee that there will never be a page break within the element. If the content of an element exceeds the dimensions of the sheet of paper on which it's being printed, the browser will be forced to insert a page break.

## Working with Widows and Orphans

Page breaks within block elements, such as paragraphs, can often leave behind widows and orphans. A widow is a fragment of text left dangling at the top of page, while an orphan is a text fragment left at the bottom of a page. Widows and orphans generally ruin the flow of the page text, making the document difficult to read. To control the size of widows and orphans, CSS supports the following properties:

```
widows: value;
orphans: value;
```

where `value` is the number of lines that must appear within the element before a page break can be inserted by the printer. The default value is 2, which means that a widow or orphan must have at least two lines of text before it can be preceded or followed by a page break.

If you wanted to increase the size of widows and orphans to three lines for the paragraphs in a document, you could apply the style rule

```
p {
  widows: 3;
  orphans: 3;
}
```

and the browser will not insert a page break if fewer than three lines of a paragraph would be stranded at either the top or the bottom of the page.

### REFERENCE

#### Controlling the Size of Widows and Orphans

- To set the minimum size of widows (lines stranded at the top of a page), apply the property

```
widows: value;
```

where `value` is the number of lines that must appear at the top of the page before the page break.

- To set the minimum size of orphans (lines stranded at the bottom of a page), apply the property

```
orphans: value;
```

where `value` is the number of lines that must appear at the bottom of the page before the page break.

Use the `widows` and `orphans` properties now, setting their size to 3 for paragraphs in the printed version of the Articles of Interest page.

## To avoid widows and orphans:

1. Within the Page Break Styles section of the **tf\_print.css** file, add the following style rule.

```
p {
  orphans: 3;
  widows: 3;
}
```

Figure 5–55 highlights the style rule for setting the size of widows and orphans.

Figure 5–55

## Setting the size of widows and orphans

widows and orphans set to a minimum of 3 lines each

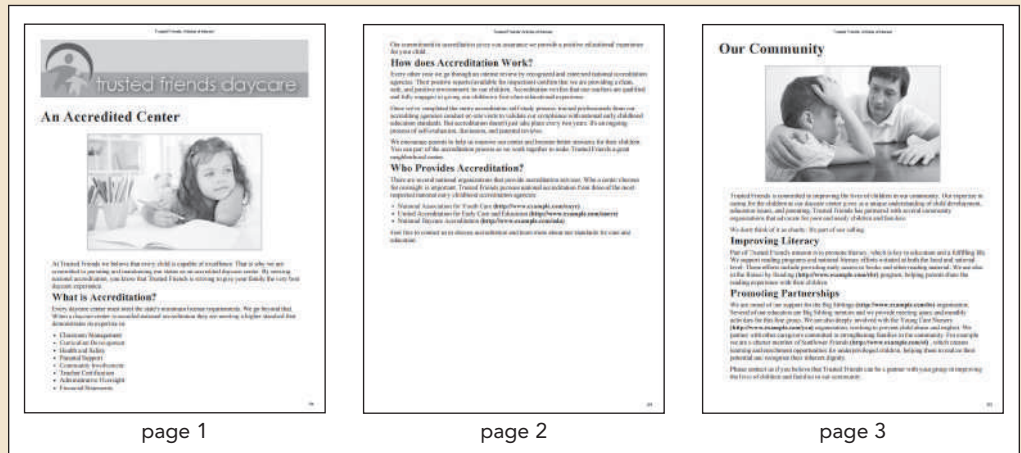
```
img, ol, ul {
  page-break-inside: avoid;
}

p {
  orphans: 3;
  widows: 3;
}
```

2. Save your changes to the file and then reload the **tf\_articles.html** file in your browser. Preview the appearance of the printed document. Figure 5–56 shows the final appearance of the printed version of this document.

Figure 5–56

## Final print version of the document



© Pressmaster/Shutterstock.com; © Gladskikh Tatiana/Shutterstock.com;

**Trouble?** Depending on your browser and your default printer, your printed version may look slightly different from the one shown in Figure 5–56.

You've completed your work on the print styles for the Articles of Interest page. By modifying the default style sheet, you've created a printout that is easier to read and more useful to the parents and customers of Trusted Friends Daycare.



## PROSKILLS

### Written Communication: Tips for Effective Printing

One challenge of printing a web page is that what works very well on the screen often fails when transferred to the printed page. For example, some browsers suppress printing background images, so that white text on a dark background, which appears fine on the computer monitor, is unreadable when printed. Following are some tips and guidelines you should keep in mind when designing the printed version of your web page:

- *Remove the clutter.* A printout should contain only information that is of immediate use to the reader. Page elements such as navigation lists, banners, and advertising should be removed, leaving only the main articles and images from your page.
- *Measure for printing.* Use only those measuring units in your style sheet that are appropriate for printing, such as points, inches, centimeters, and millimeters. Avoid expressing widths and heights in pixels because those can vary with printer resolution.
- *Design for white.* Because many browsers suppress the printing of background images and some users do not have access to color printers, create a style sheet that assumes black text on a white background.
- *Avoid absolute positioning.* Absolute positioning is designed for screen output. When printed, an object placed at an absolute position will be displayed on the first page of your printout, potentially making your text unreadable.
- *Give the user a choice.* Some readers will still want to print your web page exactly as it appears on the screen. To accommodate them, you can use one of the many JavaScript tools available on the web that allows readers to switch between your screen and print style sheets.

Finally, a print style sheet is one aspect of web design that works better in theory than in practice. Many browsers provide only partial support for the CSS print styles, so you should always test your designs on a variety of browsers and browser versions. In general, you will have the best results with a basic style sheet rather than one that tries to implement a complicated and involved print layout.

In this tutorial you've learned how to apply different styles to different types of devices and output formats. Marjorie appreciates the work you've done and will continue to rely on your knowledge of media queries, flexible layouts, and print styles as she redesigns the Trusted Friends website. You can close any open files or applications now.

### Session 5.3 Quick Check

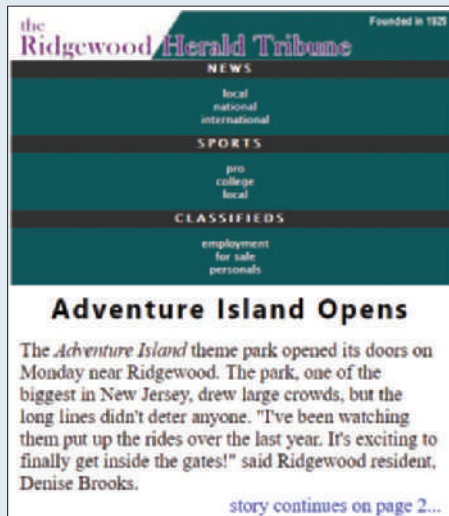
1. What attribute do you add to a link element to indicate that the style sheet is used for printed media?
  - a. `rel="print"`
  - b. `type="print"`
  - c. `media="print"`
  - d. `print="yes"`
2. What `@rule` is used for setting the properties of the printed page box?
  - a. `@page`
  - b. `@print`
  - c. `@margin`
  - d. `@printout`
3. To set the right-side printed page to have a 3 centimeter top/bottom margin and a 5 centimeter left/right margin, use:
  - a. `@page:right {margin: 3cm 5cm;}`
  - b. `@page:right {margin: 5cm 3cm;}`
  - c. `@page {side: right; margin: 5cm 3cm;}`
  - d. `@page.right {margin: 5cm 3cm;}`
4. To apply a page break before every section element, use:
  - a. `section {break: before;}`
  - b. `section {page-break: before;}`
  - c. `section {break-before: true;}`
  - d. `section {break-before: always;}`
5. To prevent a page break from being placed within any header element, use:
  - a. `header {break: never;}`
  - b. `header {page-break-inside: avoid;}`
  - c. `header {inside-break: never;}`
  - d. `header {break: none;}`
6. What style do you apply to allow the browser to wrap long strings of text to a new line whenever needed?
  - a. `word-break: auto;`
  - b. `word-wrap: true;`
  - c. `word-wrap: break-word;`
  - d. `word-inside-break: always;`
7. To limit the size of widows for all `article` elements to 3 lines or more, use:
  - a. `article {widows: 2;}`
  - b. `article {widows: 3;}`
  - c. `article {widows: 3+;}`
  - d. `article {widows: >2;}`
8. To display the URL of a hypertext link, use the property:
  - a. `attr(link)`
  - b. `attr(url)`
  - c. `attr(hypertext)`
  - d. `attr(href)`

## Coding Challenge 1

Data Files needed for this Coding Challenge: `code5-1_txt.html`, `code5-1_media_txt.css`, `code5-1_layout.css`, `code5-1_logo.jpg`, `code5-1_photo.jpg`

Use media queries to create a responsive design for the menu shown in Figure 5–57. You will need to create three menu layouts: one for screen widths 500 pixels or less, another for screen widths of 501 pixels to 710 pixels, and a third for screen widths greater than 710 pixels.

Figure 5–57 Coding Challenge 5-1 example page



© Courtesy Patrick Carey



© Courtesy Patrick Carey

Do the following:

1. Open the `code5-1_txt.html` and `code5-1_media_txt.css` files from the `html05 ► code1` folder. Enter **your name** and **the date** in each document and save the files as `code5-1.html` and `code5-1_media.css` respectively.
2. Go to the `code5-1.html` file in your editor. Within the head section insert `link` elements linking the page to the `code5-1_layout.css` and `code5-1_media.css` files.
3. Add a viewport meta tag to the document head to set the width of the layout viewport equal to the width of the device and set the initial scale of the viewport to 1.0. Review the contents of the file and then save your changes.
4. Go to the `code5-1_media.css` file in your editor.

5. Create a media query for devices with a maximum width of 500 pixels. Within the query do the following:
  - a. Set the display of the `img` element within the article element to `none`.
  - b. Center the text contained within the `ul` element belonging to the submenu class.
6. Create a media query for devices with a minimum width of 501 pixels. Within the query do the following:
  - a. Float the `nav` element on the left page margin.
  - b. Set the width of the `nav` element to 130 pixels and the height to 400 pixels.
  - c. Set the top margin of the `nav` element to 30 pixels, the right margin to 25 pixels, and the bottom and left margins to 0 pixels.
7. Create a media query for devices with a minimum width of 710 pixels. Within the query do the following:
  - a. Set the `float` property of the `nav` element to `none`, its width to 100% and its height to `auto`. Set the `nav` element margins to 0.
  - b. Set the display of `ul` elements of the `mainmenu` class to `flex` with the flex flow in the row direction with no wrapping; justify the contents of the flexbox in the center.
  - c. Set the `flex` property of `li` elements with the `ul.mainmenu` element to have a growth factor of 0, a shrink factor of 1, and a basis value of 120 pixels.
8. Save your changes to the file and then view the page under different screen widths, verifying that the menu format changes as the screen width changes as shown in Figure 5–57.
9. Submit the completed file to your instructor.

## Coding Challenge 2

Data Files needed for this Coding Challenge: `code5-2_txt.html`, `code5-2_flex_txt.css`, `code5-2_layout.css`, and 13 image files

Figure 5–58 shows the layout of a page that displays cards containing social media icons. Create this page using CSS flex styles so that the icons are always laid out in rows and columns for any screen width.

Figure 5–58 Coding Challenge 5-2 example page





Do the following:

1. Open the **code5-2\_txt.html** and **code5-2\_flex\_txt.css** files from the `html05 > code2` folder. Enter **your name** and **the date** in each document and save the files as **code5-2.html** and **code5-2\_flex.css** respectively.
2. Go to the **code5-2.html** file in your editor. Within the head section insert `link` elements linking the page to the `code5-2_layout.css` and `code5-2_flex.css` files. Review the contents of the file and then save your changes.
3. Go to the **code5-2\_flex.css** file in your editor.
4. Display the `section` element as a flexbox. Set the flow of items within the flexbox to go in row order with reverse wrapping so that the first item (Facebook) appears in the bottom-left corner and the last item (E-mail) appears in the top-right corner.
5. Set the growth and shrink rate of the `div` elements of the card class to 1 and 1. Set the flex basis of those elements to 200 pixels.
6. Display each `div` element of the card class itself as a flexbox.
7. Apply the following flex layout to the items within the card `div` elements:
  - a. Lay out the items in column order with no wrapping.
  - b. Justify the content of the items within the flexbox with space between.
  - c. Center each of the items with respect to the cross axis.
8. Save your changes to the style sheet file and then open **code5-2.html** in your browser. Test the layout under different screen widths, verifying that the `div` cards always fill up the page grid and that for a width small enough, the contents are laid out in a single column.
9. Submit the completed file to your instructor.

### Coding Challenge 3

Data Files needed for this Coding Challenge: **code5-3\_txt.html**, **code5-3\_print\_txt.css**, **code5-3\_layout.css**, and 1 image file

A list of the top 15 travel sites on the web is shown in Figure 5–59. The style sheet code for the screen version has already been written, but you have been tasked to create the style sheet for the print version. Complete the web page by writing the print styles and linking them to the web page.



Figure 5–59 Coding Challenge 5-3 example page



**Travel Sites on the Web**

It's easier than ever to find great deals on airfare, hotels, and car rentals using the travel sites available on the Web.

The following are the top 15 most population travel sites on the Web as rated by eBiz/MBA [ <http://www.ebizmba.com/articles/travel-websites> ].

1. Booking [ <http://www.booking.com> ]
2. TripAdvisor [ <http://www.tripadvisor.com> ]
3. Yahoo! Travel [ <http://www.travel.yahoo.com> ]
4. Expedia [ <http://www.expedia.com> ]
5. Priceline [ <http://www.priceline.com> ]
6. Hotels [ <http://www.hotels.com> ]
7. Travelocity [ <http://www.travelocity.com> ]
8. Kayak [ <http://www.kayak.com> ]
9. Orbitz [ <http://www.orbitz.com> ]
10. Hotwire [ <http://www.hotwire.com> ]
11. HomeAway [ <http://www.homeaway.com> ]
12. TravelZoo [ <http://www.travelzoo.com> ]
13. AirBnB [ <http://www.airbnb.com> ]
14. LonelyPlanet [ <http://www.lonelyplanet.com> ]
15. Viator [ <http://www.viator.com> ]

Do the following:

1. Open the **code5-3\_txt.html** and **code5-3\_print\_txt.css** files from the `html05 ▶ code3` folder. Enter **your name** and **the date** in each document and save the files as **code5-3.html** and **code5-3\_print.css** respectively.
2. Go to the **code5-3.html** file in your editor. Within the head section insert `link` elements linking the page to the `code5-3_layout.css` and `code5-3_print.css` files. Use the `code5-3_layout.css` file for screen output and the `code5-3_print.css` file for printed output. Review the contents of the file and then save your changes.
3. Go to the **code5-3\_print.css** file in your editor.
4. Set the printed page size to 8.5 by 11 inches with a 1-inch margin.
5. Remove all underlining from hypertext links.
6. Prevent the browser from inserting page breaks within any `nav` element.
7. Set the line height of every `li` element nested within a `nav` and `ol` element to 0.3 inches.
8. Use the `after` pseudo-element to display printed hypertext link in the following format: *link* [ *url* ] and add the following styles to the `after` pseudo-element:
  - a. Display the *url* text as an inline block.
  - b. Set the left margin of the *url* text to 20 pixels.
  - c. Set the value of the `word-wrap` property to "break-word".
9. Save your changes to the file.
10. View the printed version of the page in your browser to verify that it resembles Figure 5–59.
11. Submit the completed file to your instructor.

## Coding Challenge 4

Data Files needed for this Coding Challenge: `code5-4_txt.html`, `code5-4_debug_txt.css`, `code5-4_layout.css`, `code5-4_logo.jpg`, `code5-4_photo.jpg`

You have been asked to revise a website involving responsive design that contains several errors. A preview of the page under different screen widths is shown in Figure 5–60. Fix the errors in the code for the HTML and CSS file.

Figure 5–60 Coding Challenge 5-4 example page



Do the following:

1. Open the `code5-4_txt.html` and `code5-4_debug_txt.css` files from the `html05 ▶ code4` folder. Enter *your name* and *the date* in each document and save the files as `code5-4.html` and `code5-4_debug.css` respectively.
2. Go to the `code5-4.html` file in your editor. Within the head section insert `link` elements linking the page to the `code5-4_layout.css` and `code5-4_debug.css` files. Review the contents of the file.
3. There is a single error within the head section of the page. Locate the error and fix it. Save your changes to the file.
4. Go to the `code5-4_debug.css` file in your editor. There are 8 separate syntax errors in the stylesheet. Locate and correct all eight errors and then save your changes.
5. Test both the `code5-4.html` and `code5-4_debug.css` files in a validator to confirm that both pass validation with no errors or warnings reported.
6. View the `code5-4.html` file in your browser under different screen widths and compare your page to Figure 5–60, confirming that your page layout matches the one shown in the figure.
7. Submit the completed file to your instructor.

## Review Assignments

Data Files needed for the Review Assignments: [tf\\_print2\\_txt.css](#), [tf\\_styles4\\_txt.css](#), [tf\\_tips\\_txt.html](#), 2 CSS files, 4 PNG files

Marjorie meets with you to discuss the redesign of the blog page showing parenting tips. As with the other pages you've worked on, she wants this page to be compatible with mobile devices, tablet and desktop devices, and printers. Marjorie has already written the page content and has done much of the initial design work. She needs you to complete the project by writing media queries for the different display options. Figure 5–61 shows a preview of the mobile design and the desktop design.

Figure 5–61

Parenting Tips page



© Courtesy Patrick Carey

You'll use several flexboxes to create the layout for these two designs so that the page content automatically rescales as the screen width changes.

Complete the following:

1. Use your HTML editor to open the [tf\\_tips\\_txt.html](#), [tf\\_styles4\\_txt.css](#), and [tf\\_print2\\_txt.css](#) files from the html05 ► review folder. Enter **your name** and **the date** in the comment section of each file, and save them as [tf\\_tips.html](#), [tf\\_styles4.css](#), and [tf\\_print2.css](#) respectively.
2. Go to the [tf\\_tips.html](#) file in your editor. Add a viewport meta tag to the document head to set the width of the layout viewport equal to the width of the device and set the initial scale of the viewport to 1.0.

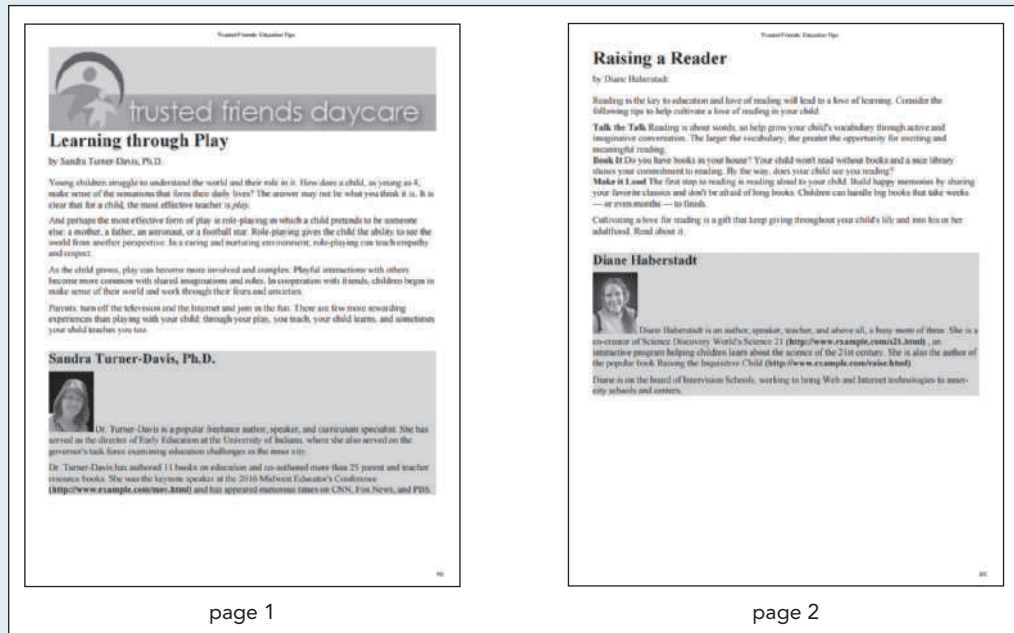


3. Create links to the following style sheets: a) the `tf_base.css` file to be used with all devices, b) the `tf_styles4.css` file to be used with screen devices, and c) the `tf_print2.css` file to be used for printed output.
4. Take some time to study the contents and structure of the document, paying special attention to the IDs and class names of the elements, and then save your changes.
5. Go to the **`tf_styles4.css`** file in your editor. Note that Marjorie has placed all of her styles in the `tf_designs.css` file and imported them into this style sheet. You will not need to edit that style sheet file, but you might want to view it to become familiar with her style rules.
6. Go to the General Flex Styles section. Within this section, you'll create a flexible layout that varies in response to changing screen widths.
7. In the General Flex Styles section create a style rule for the `body` element that displays the page body as a flexbox flowing in the row direction, wrapping content to a new line as needed.
8. The page content is divided into two `section` elements with IDs of `left` and `right`. The left section does not need as much of the screen width. Create a style rule for the left section that sets its flex growth and shrink rates to 1 and 8 respectively and sets its flex basis size to 130 pixels.
9. The right section requires more screen width. Create a style rule for the right section that sets its flex growth and shrink values to 8 and 1 and sets its flex basis size to 351 pixels.
10. Next, set the display of the section element with class ID of `tips` as a flexbox. Have the content of the flexbox flow in the row direction with row wrapping enabled.
11. Create a style rule for the `article` element that lays it out with a flex growth value of 2, flex shrink value of 1, and a flex basis size of 351 pixels.
12. The biographical asides within each tips section need to occupy less screen space. Create a style rule for the `aside` element that lays it out with a flex growth value of 1, flex shrink value of 2, and a flex basis size of 250 pixels.
13. Finally, the horizontal navigation list at the top of the page will also be treated as a flexbox. Create a style rule for the `nav.horizontal ul` selector that displays it as a flexbox in column orientation with wrapping.
14. Go to the Mobile Devices section and create a media query for screen devices with a maximum width of 480 pixels.
15. For mobile devices, the vertical list of links to archived parenting tips should be displayed in several columns at the bottom of the page. Within the media query you created in the last step, add the following style rules:
  - a. for the `nav.vertical ul` selector, create a style rule that displays it as a flexbox in column orientation with wrapping. Set the height of the element to 240 pixels.
  - b. to give the `section` element with an ID of `left` a flex order value of 99 to place it near the bottom of the page.
  - c. to give the `body > footer` selector an order value of 100 to put it at the page bottom.
16. Marjorie wants to hide the navigation list at the top of the page when viewed on a mobile device unless the user hovers (or taps) a `navicon`. Using the technique shown in this tutorial, add the following style rules to set the behavior of the `navicon` within the media query for mobile devices:
  - a. Display the `navicon` by creating a style rule for the `a#navicon` selector to display it as a block.
  - b. Set the display property of the `nav.horizontal ul` selector to `none`.
  - c. Display the navigation list contents in response to a hover or touch by creating a style rule for the `a#navicon: hover+ul, nav.horizontal ul: hover` selector that sets its display value to block.
17. Go to the Tablets and Desktop Devices section. Create a media query for screen devices with a width of at least 481 pixels. Under the wider screens, the contents of the horizontal navigation list at the top of the page should be displayed in several columns. In order to have the list items wrap to a new column, add a style rule to the media query that sets the height of the `ul` element within the horizontal navigation list to 160 pixels.

18. Save your changes to the style sheet and then open the **tf\_tips.html** file in your browser or device emulator. Verify that as you change the screen width the layout of the page automatically changes to match the layout designs shown in Figure 5–61.

Next, you'll create the print styles for the Parenting Tips page. Figure 5–62 shows a preview of the output on a black and white printer.

**Figure 5–62** Parenting Tips print version



© Courtesy Patrick Carey

19. Go to the **tf\_print2.css** file in your editor. Go to the Hidden Objects section and hide the display of the following page elements: all navigation lists, the h1 heading in the body header, the left section element, and the body footer.
20. Go to the Page Box Styles section and set the page size to 8.5 inches by 11 inches with a margin of 0.5 inches.
21. Go the Header Styles section and add a style rule that displays the logo image as a block with a width of 100%.
22. Go to the Typography Styles section and add the following style rules for the text in the printed pages:
  - a. For headers within the `article` element, set the bottom margin to 0.2 inches.
  - b. For h1 headings within the `article` element, set the font size to 24 points and the line height to 26 points.
  - c. For the `aside` element, set the background color to `rgb(211, 211, 211)` and add a top margin of 0.3 inches.
  - d. For h1 headings in `aside` elements, set the font size to 18 points and the line height to 20 points.
  - e. For images within `aside` elements, set the width to 0.8 inches.
  - f. For paragraphs, set the font size to 12 points with a top and bottom margin of 0.1 inches.
23. Go to the Hypertext Styles section and add style rules to display all hypertext links in black with no underline. Also, insert a style rule that adds the text of the URL after the hypertext link in bold with the `word-wrap` property set to `break-word`.

24. Go to the Page Break Styles section and add the following style rules to
  - a. insert page breaks after every `aside` element.
  - b. never allow a page break within an `ol`, `ul`, or `img` element.
  - c. set the size of widows and orphans within paragraphs to 3 lines each.
25. Save your changes to the file.
26. Reload the `tf_tips.html` file in your browser and preview its printed version. Verify that your pages resemble those shown in Figure 5–62 (there may be differences depending on your browser and your printer).

## APPLY

## Case Problem 1

**Data Files needed for this Case Problem:** `gp_cover_txt.html`, `gp_page1_txt.html`, `gp_page2_txt.html`, `gp_page3_txt.html`, `gp_layout_txt.css`, `gp_print_txt.css`, 2 CSS files, 21 PNG files

**Golden Pulps** Devan Ryan manages the website *Golden Pulps*, where he shares tips on collecting and fun stories from the “golden age of comic books”—a period of time covering 1938 through the early 1950s. Devan wants to provide online versions of several classic comic books, which are now in the public domain.

He’s scanned the images from the golden age comic book, *America’s Greatest Comics 001*, published in March, 1941, by Fawcett Comics and featuring Captain Marvel. He’s written the code for the HTML file and wants you to help him develop a layout design that will be compatible with mobile and desktop devices. Figure 5–63 shows a preview of the mobile and desktop version of a page you’ll create.

Figure 5–63

Golden Pulps sample page



Complete the following:

1. Using your editor, open the `gp_cover_txt.html`, `gp_page1_txt.html`, `gp_page2_txt.html`, `gp_page3_txt.html`, `gp_layout_txt.css`, and `gp_print_txt.css` files from the `html05 ▶ case1` folder. Enter *your name* and *the date* in the comment section of each file, and save them as `gp_cover.html`, `gp_page1.html`, `gp_page2.html`, `gp_page3.html`, `gp_layout.css`, and `gp_print.css` respectively.
2. Go to the `gp_cover.html` file in your editor. Add a viewport `meta` tag to the document head, setting the width of the layout viewport to the device width and setting the initial scale of the viewport to 1.0.
3. Create links to the following style sheets: a) the `gp_reset.css` file to be used with all devices, b) the `gp_layout.css` file to be used with screen devices, and c) the `gp_print.css` file to be used for printed output.
4. Take some time to study the contents and structure of the file. Note each panel from the comic book is stored as a separate inline image with the class name `panel` along with class names of `size1` to `size4` indicating the size of the panel. Size1 is the largest panel down to size4, which is the smallest panel. Close the file, saving your changes.
5. Repeat Steps 2 through 4 for the `gp_page1.html`, `gp_page2.html`, and `gp_page3.html` files.
6. Go to the `gp_layout.css` file in your editor. In this style sheet, you'll create the layout styles for mobile and desktop devices. Note that Devan has used the `@import` rule to import the `gp_designs.css` file, which contains several graphical and typographical style rules.
7. Go to the Flex Layout Styles section and insert a style rule to display the `body` element as a flexbox oriented as rows with wrapping.
8. The page body content has two main elements. The `section` element with the ID `sheet` contains the panels from the comic book page. The `article` element contains information about the comic book industry during the Golden Age. Devan wants more of the page width to be given to the comic book sheet. Add a style rule that sets the flex growth and shrink rate of the `section#sheet` selector to 3 and 1 respectively and set its flex basis size to 301 pixels.
9. Less page width will be given to the `article` element. Create a style rule to set its flex growth and shrink values to 1 and 3 respectively and set its flex basis size to 180 pixels.
10. Go to the Mobile Devices section and create a media query for screen devices with a maximum width of 480 pixels.
11. With mobile devices, Devan wants each comic book panel image to occupy a single row. Create a style rule that sets the width of `img` elements belonging to the `panel` class to 100%.
12. For mobile devices, Devan wants the horizontal navigation links to other pages on the Golden Pulps website to be displayed near the bottom of the page. Within the media query, set the flex order of the `nav.horizontal` selector to 99.
13. Create a style rule to set the flex order of the `body > footer` selector to 100.
14. Go to the Tablet and Desktop Devices: Greater than 480 pixels section and create a media query that matches screen devices with widths greater than 480 pixels.
15. For tablet and desktop devices, you'll lay out the horizontal navigation list as a single row of links. Within the media query, create a style rule for the `nav.horizontal ul` selector that displays that element as a flexbox, oriented in the row direction with no wrapping. Set the height of the element to 40 pixels.
16. For the `nav.horizontal ul li` selector set the flex growth shrink, and basis size values to 1, 1, and auto respectively so that each list items grows and shrinks at the same rate.
17. With wider screens, Devan does not want the panels to occupy their own rows as is the case with mobile devices. Instead, within the media query create style rules, define the width of the different classes of comic book panel images as follows:
  - a. Set the width of `size1 img` elements to 100%.
  - b. Set the width of `size2 img` elements to 60%.
  - c. Set the width of `size3 img` elements to 40%.
  - d. Set the width of `size4 img` elements to 30%.
18. Save your changes to the file and then open the `gp_cover.html` file in your browser or device emulator. Click the navigation links to view the contents of the cover and first three pages. Verify



that with a narrow screen the panels occupy their own rows and with a wider screen the sheets are laid out with several panels per row. Further verify that the horizontal navigation list is placed at the bottom of the page for mobile devices.

19. Devan also wants a print style that displays each comic book sheet on its own page and with none of the navigation links. Go to the **gp\_print.css** style sheet in your editor. Add style rules to
  - a. hide the `nav`, `footer`, and `article` elements.
  - b. set the width of the element referenced by the `section#sheet` selector to 6 inches. Set the top/bottom margin of that element to 0 inches and the left/right margin to `auto` in order to center it within the printed page.
  - c. set the width of `img` elements belong to the `size1` class to 5 inches, `size2` images to 3 inches, `size3` images to 2 inches, and `size4` images to 1.5 inches.
20. Save your changes to the file and then reload the contents of the comic book pages in your browser and preview the printed pages. Verify that the printed page displays only the website logo, the name of the comic book, and the comic book panels.

## Case Problem 2

**Data Files needed for this Case Problem: cw\_home\_txt.html, cw\_styles\_txt.css, 2 CSS files, 10 PNG files**

**Cauli-Wood Gallery** Sofia Fonte is the manager of the *Cauli-Wood Gallery*, an art gallery and coffee shop located in Sedona, Arizona. She has approached you for help in redesigning the gallery's website to include support for mobile devices and tablets. Your first project will be to redesign the site's home page following the principles of responsive design. A preview of the mobile and desktop versions of the website's home page is shown in Figure 5–64.

Figure 5-64

Cauli-Wood Gallery home page



Right: © Tischenko Irina/Shutterstock.com; © re\_bekka/Shutterstock.com; © Boyan Dimitrov/Shutterstock.com; © rubtsov/Shutterstock.com; © Fotocrisis/Shutterstock.com; © Anna Ismagilova/Shutterstock.com; © DeepGreen/Shutterstock.com; Source: Facebook; Source: Twitter, Inc.  
 Left: © Tischenko Irina/Shutterstock.com; © Courtesy Patrick Carey; © re\_bekka/Shutterstock.com; © Anna Ismagilova/Shutterstock.com; © rubtsov/Shutterstock.com; Source: Facebook; Source: Twitter, Inc.

Sofia has already written much of the HTML code and some of the styles to be used in this project. Your job will be to finish the redesign and present her with the final version of the page.

Complete the following:

1. Using your editor, open the **cw\_home\_txt.html** and **cw\_styles\_txt.css** files from the html05 ► case2 folder. Enter **your name** and **the date** in the comment section of each file, and save them as **cw\_home.html** and **cw\_styles.css** respectively.
2. Go to the **cw\_home.html** file in your editor. Within the document head, insert a meta element that sets the browser viewport for use with mobile devices. Also, create links to **cw\_reset.css** and **cw\_styles.css** style sheets. Take some time to study the contents and structure of the document and then close the file saving your changes.
3. Return to the **cw\_styles.css** file in your editor. At the top of the file, use the **@import** rule to import the contents of the **cw\_designs.css** file, which contains several style rules that format the appearance of different page elements.
4. At the bottom of the home page is a navigation list with the ID **bottom** containing several ul elements. Sofia wants these ul elements laid out side-by-side. Create a style rule for the **nav#bottom** selector displaying its element as a flexbox row with no wrapping. Set the **justify-content** property so that the flex items are centered along the main axis.
5. For the **nav#bottom ul** selector, create a style rule to set the flex growth rate to 0, the shrink rate to 1, and the basis value to 150 pixels.

6. Sofia wants more highly contrasting colors when the page is displayed in a mobile device. Create a media query for mobile screen devices with maximum widths of 480 pixels. Within that media query, insert a style rule that sets the font color of the `body` element to `rgb(211, 211, 211)` and sets the body background color to `rgb(51, 51, 51)`.
7. Sofia also wants to reduce the clutter in the mobile version of the home page. Hide the following elements for mobile users: the `aside` element, any `img` element within the `article` element, and the spotlight section element.
8. At the top of the web page is a navigation list with the ID `top`. For mobile devices, create a style rule for the `nav#top ul` selector, displaying the element as a flexbox row with wrapping. For each list item within the `nav#top ul` selector, set the font size to `2.2em`. Size the list items by setting their flex values to 1 for the growth and shrink rates and 130 pixels for the basis value.
9. Under the mobile layout, the six list items in the top navigation list should appear as square blocks with different background images. Using the selector `nav#top ul li:nth-of-type(1)` for the first list item, create a style rule that changes the background to the background image `cw_image01.png`. Center the background image with no tiling and size it so that the entire image is contained within the background.
10. Repeat the previous step for the next five list items using the same general format. Use the `cw_image02.png` file for background of the second list item, the `cw_image03.png` file for the third list item background, and so forth up through the `nav#top ul li:nth-of-type(6)` selector.
- ✚ **Explore** 11. Sofia has placed hypertext links for the gallery's phone number and e-mail address in a paragraph with the ID `links`. For mobile users, she wants these two hypertext links spaced evenly within the paragraph that is displayed below the top navigation list. To format these links, create a style rule that displays the element referenced by the `p#links` selector as a flexbox row with no wrapping, then add a style that sets the value of the `justify-content` property to `space-around`.
12. She wants the telephone and e-mail links to be prominently displayed on mobile devices. For each `p#links a` selector, apply the following style rule that: a) displays the link text in white on the background color `rgb(220, 27, 27)`, b) sets the border radius around each hypertext to 20 pixels with 10 pixels of padding, and c) removes any underlining from the hypertext links.
13. Next, you'll define the layout for tablet and desktop devices. Create a media query for screen devices whose width is 481 pixels or greater. Within this media query, display the `body` element as a flexbox in row orientation with wrapping.
14. The page body has four children: the header, the footer, the `article` element, and the `aside` element. The `article` and `aside` elements will share a row with more space given to the `article` element. Set the flex growth, shrink, and basis values of the `article` element to 2, 1, and 400 pixels. Set those same flex values for the `aside` element to 1, 2, and 200 pixels.
- ✚ **Explore** 15. For tablet and desktop devices, the top navigation list should be displayed as a horizontal row with no wrapping. Enter a style rule for the `nav#top ul` selector to display that element as a flexbox with a background color of `rgb(51, 51, 51)` and a height of 50 pixels. Use the `justify-content` and `align-items` property to center the flex items within that flexbox both horizontally and vertically.
16. For the `nav#top ul li` selector, create a style rule for those list items, setting the flex growth rate to 0, the flex shrink rate to 1, and the basis value to 80 pixels.
17. Sofia doesn't want the links paragraph displayed for tablet and desktop devices. Complete the media query for tablet and desktop devices by creating a style rule for the `p#links` selector to hide the paragraph.
18. Save your changes to the style sheet and then open the `cw_home.html` file in your browser or device emulator. Verify that the layout and contents of the page switch between the mobile version and the tablet/desktop version shown in Figure 5-64 as the screen width is increased and decreased.