TUTORIAL 5

Designing for the Mobile Web

Creating a Mobile Website for a Daycare Center

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

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

tutorial
tf_articles_txt.html
tf_home_txt.html
tf_prek_txt.html
tf_flex_txt.css
tf_navicon_txt.css
tf_print_txt.css
tf_styles1_txt.css
+ 9 files

review
tf_tips_txt.html
tf_print2_txt.css
tf_styles4_txt.css
+ 6 files

code1
code5-1_txt.html
code5-1_media_txt.css
+ 3 files

code2
code5-2_txt.html
code5-2_flex_txt.css
+ 14 files

code3
code5-3_txt.html
code5-3_print_txt.css
+ 2 files

code4
code5-4_txt.html
code5-4_debug_txt.css
+ 3 files

case1
gp_cover_txt.html
gp_page1_txt.html
gp_page2_txt.html
gp_page3_txt.html
gp_layout_txt.css
gp_print_txt.css
+ 23 files

case2
cw_home_txt.html
cw_styles_txt.css
+ 12 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.

Within a media query are style rules that are only applied to devices that match the query.

This media query matches screens with a maximum width of 480 pixels.

This media query matches screens with a minimum width of 481 pixels.

This media query matches screens with a minimum width of 769 pixels.

```
/* ===================================
   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

}
```

Media Queries

dotshock/Shutterstock.com; Robert Kneschke/Shutterstock.com; Jmlevick/openclipart; Easy/openclipart; BenBois/openclipart

Introducing Responsive Design

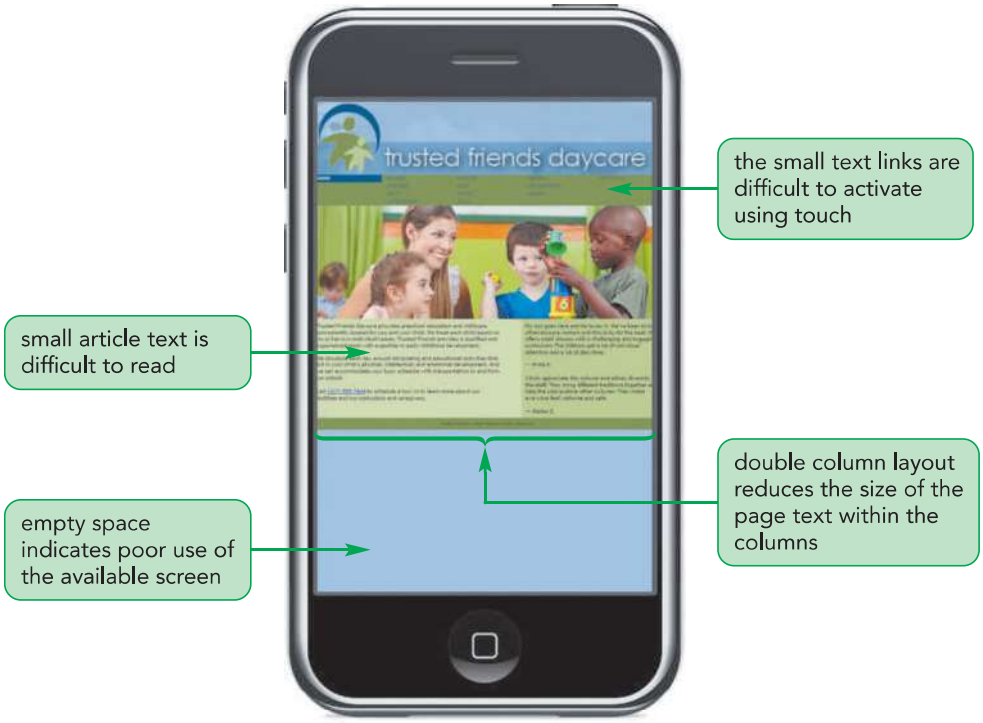
In the first four tutorials, you created a single set of layout and design styles for your websites without considering what type of device would be rendering the site. However, this is not always a practical approach and with many users increasingly accessing the web through mobile devices, a web designer 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**



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
`aspect-ratio`	The ratio of the width of the display area to its height
`color`	The number of bits per color component of the output device; if the device does not support color, the value is 0
`color-index`	The number of colors supported by the output device
`device-aspect-ratio`	The ratio of the `device-width` value to the `device-height` value
`device-height`	The height of the rendering surface of the output device
`device-width`	The width of the rendering surface of the output device
`height`	The height of the display area of the output device
`monochrome`	The number of bits per pixel in the device's monochrome frame buffer
`orientation`	The general description of the aspect ratio: equal to `portrait` when the height of the display area is greater than the width; equal to `landscape` otherwise
`resolution`	The resolution of the output device in pixels, expressed in either dpi (dots per inch) or dpcm (dots per centimeter)
`width`	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.

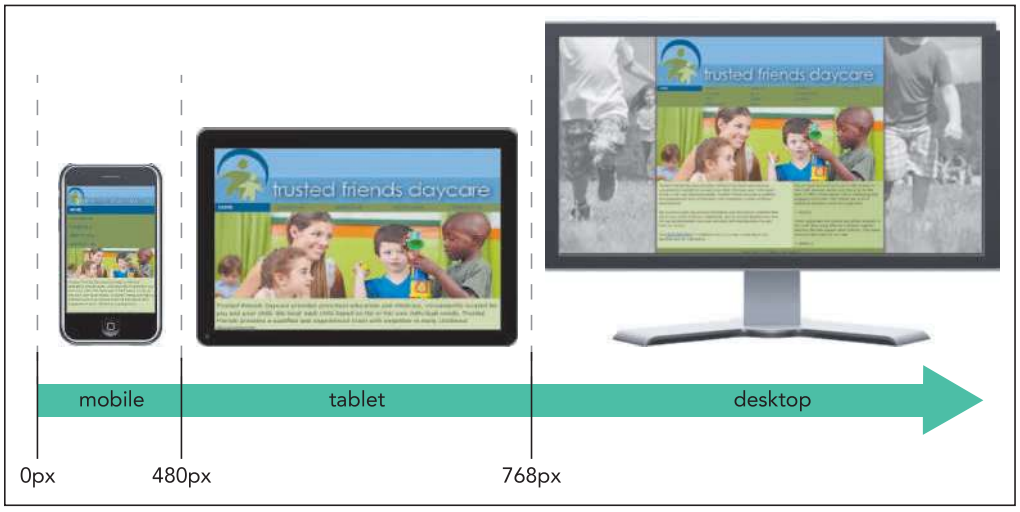
- To import a style sheet based on a media query, apply the following `@import` rule within a CSS style sheet

```
@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**



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:

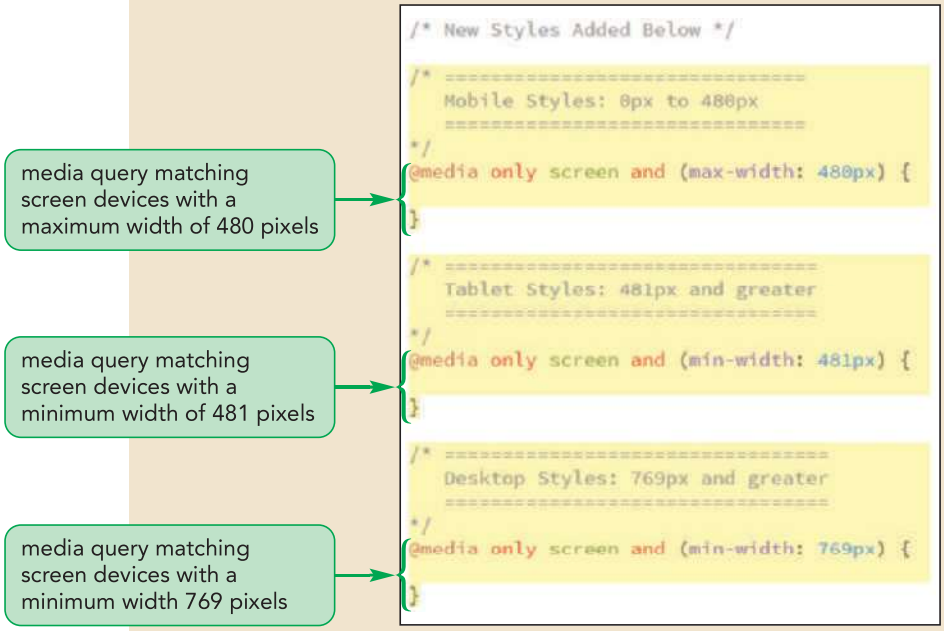
1. Return to the **tf_styles1.css** file in your editor.
2. Marjorie has already inserted the base styles that will apply to all devices at the top of the style sheet file. Take time to review those styles.

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



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

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

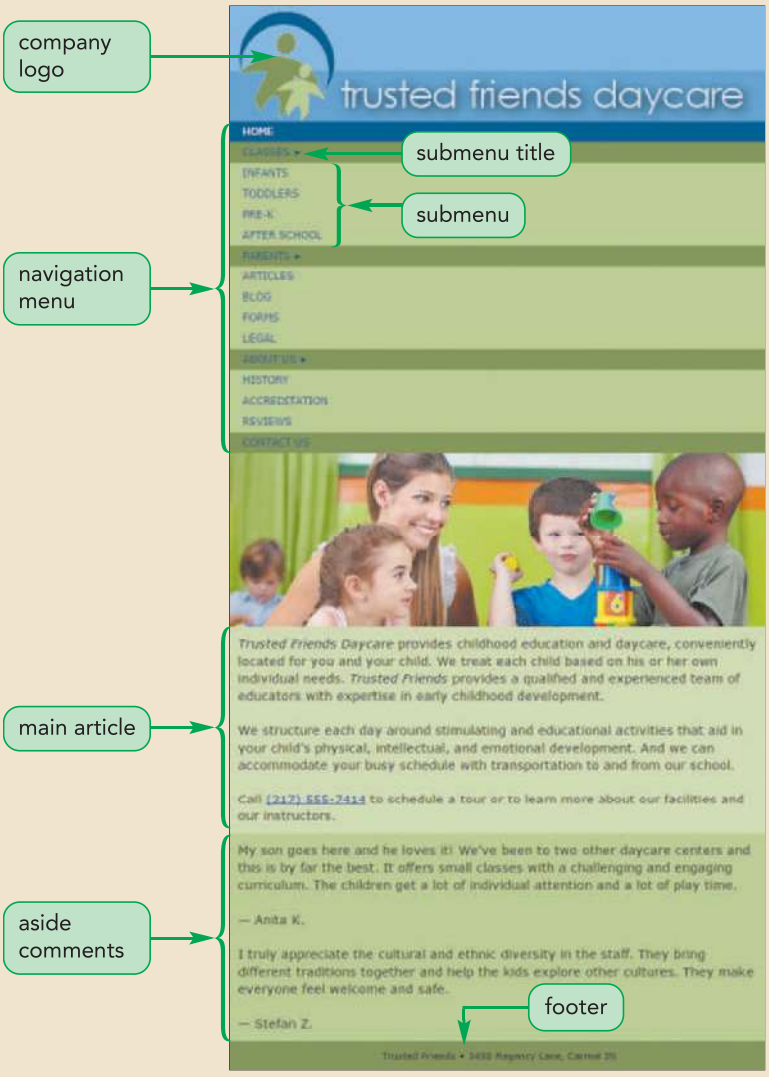
sets the width of the layout viewport to the width of the device

page does not automatically zoom out when the page is initially opened by the browser

```
<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



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 2×2 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

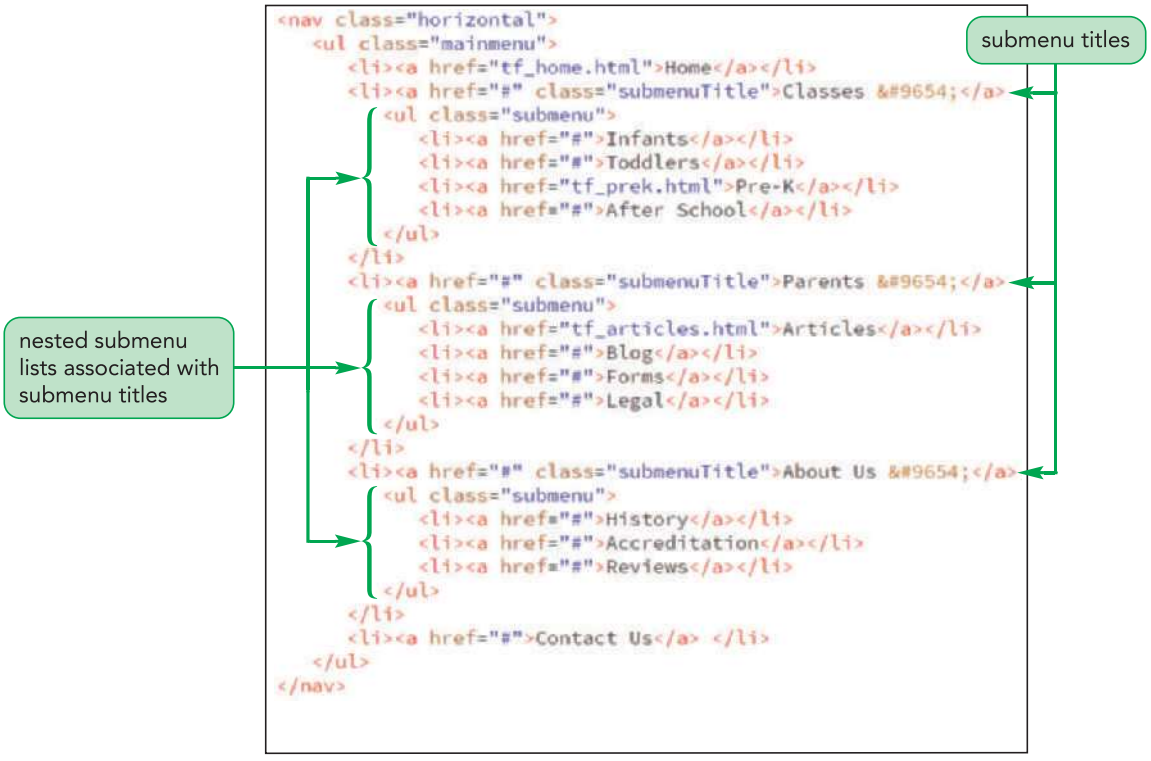
A mobile website design should reflect how users interact with their mobile devices. Because your users will be working with a small handheld touchscreen device, one key component in your design is to have the most important information up-front and easily accessible, which means your home page on a 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

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.

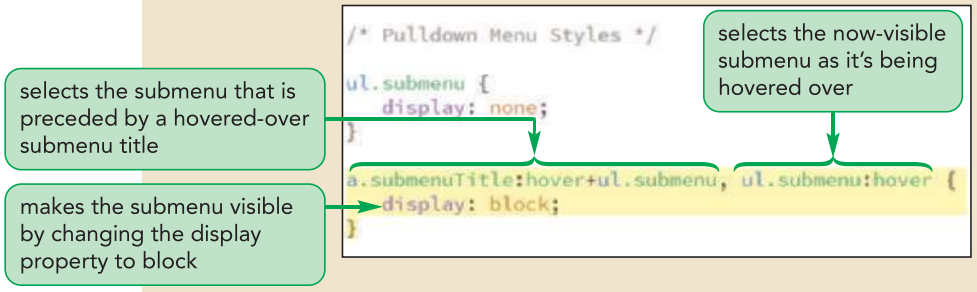
To redisplay the navigation submenus:

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.

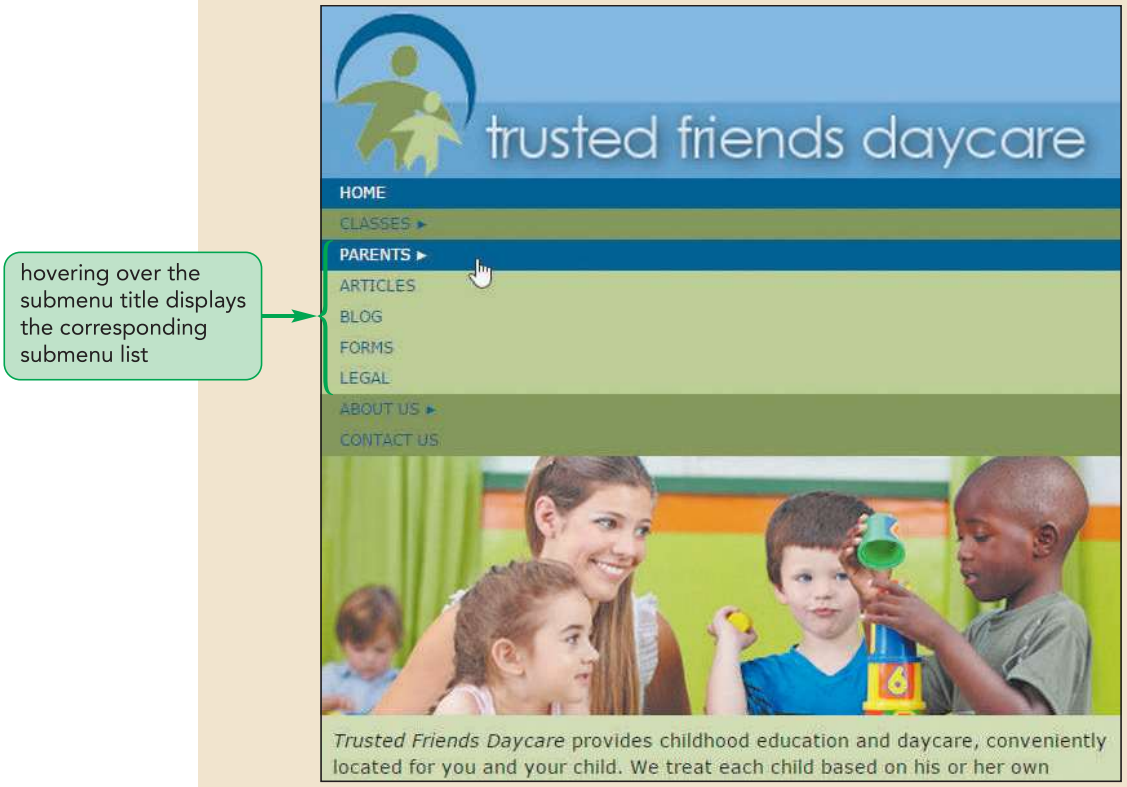
Figure 5–13 Displaying the hidden submenus



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 (*developer.android.com/sdk*)
iOS SDK	Software development kit for iPhone, iPad, and other iOS devices (*developer.apple.com*)
Mobile Phone Emulator	Online emulation for a variety of mobile devices (*www.mobilephoneemulator.com*)
Mobile Test Me	Online emulation for a variety of mobile devices (*mobiletest.me*)
Opera Mobile SDK	Developer tools for the Opera Mobile browser (*www.opera.com/developer*)

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:

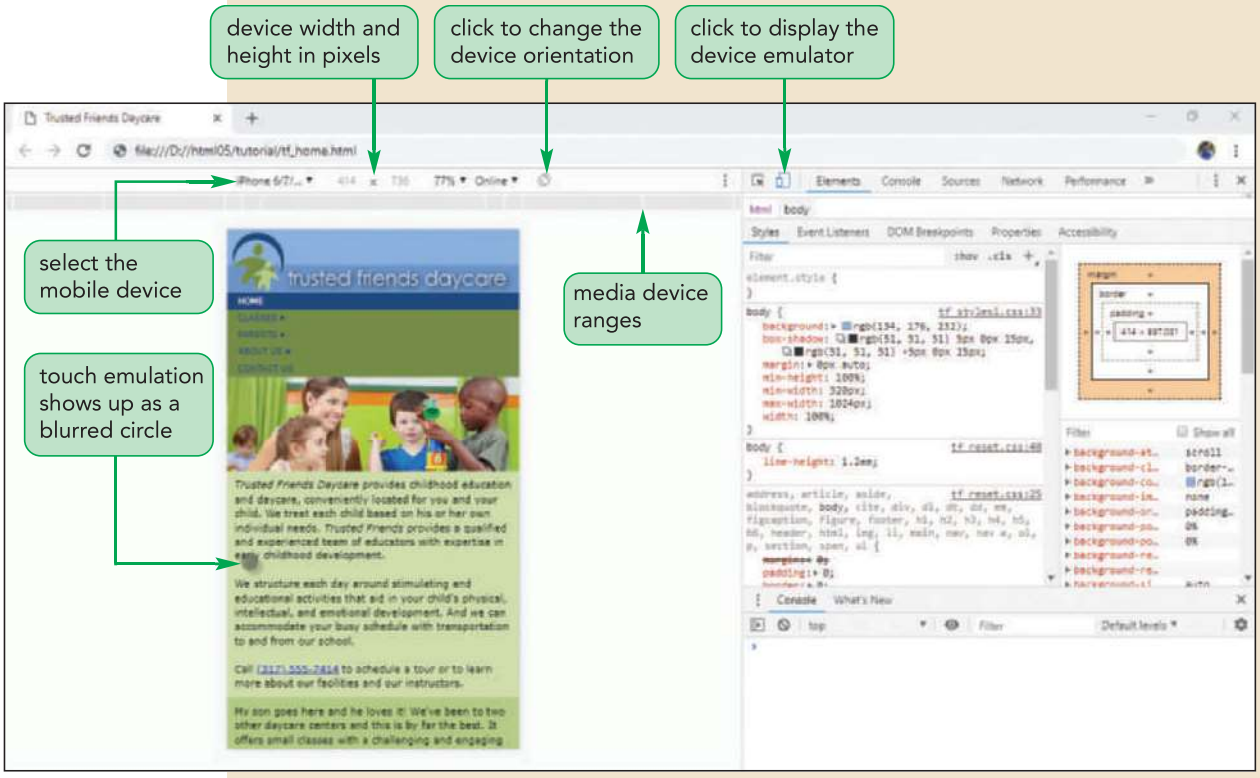
1. Return to the **tf_home.html** file in the Google Chrome browser and press **F12** to open the developer tools pane.
2. If necessary, click the **device** icon  located at the top of the developer pane to display the device toolbar.
3. Select a device of your choosing from the drop-down list of devices 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



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.

The styles rules for a media query must always be placed within curly braces to define the extent of the query.

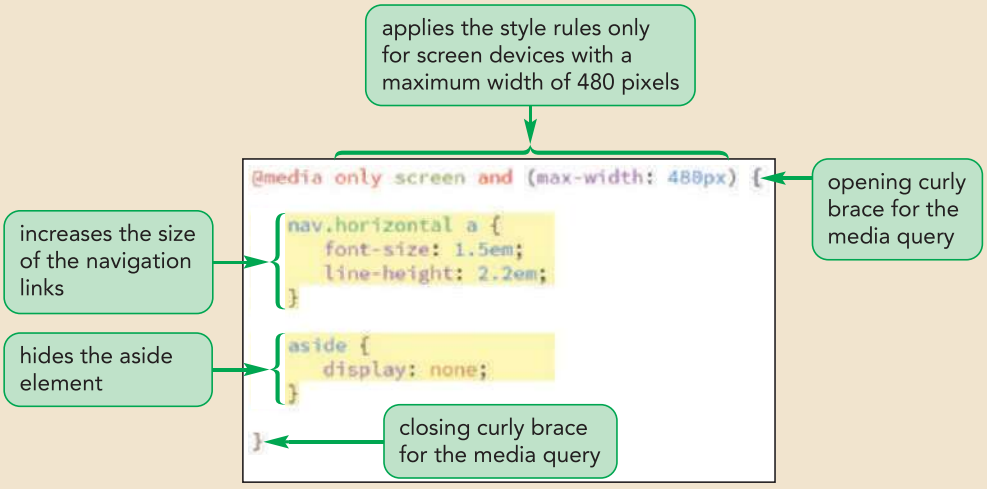
```
nav.horizontal a {
   font-size: 1.5em;
   line-height: 2.2em;
}
```

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.

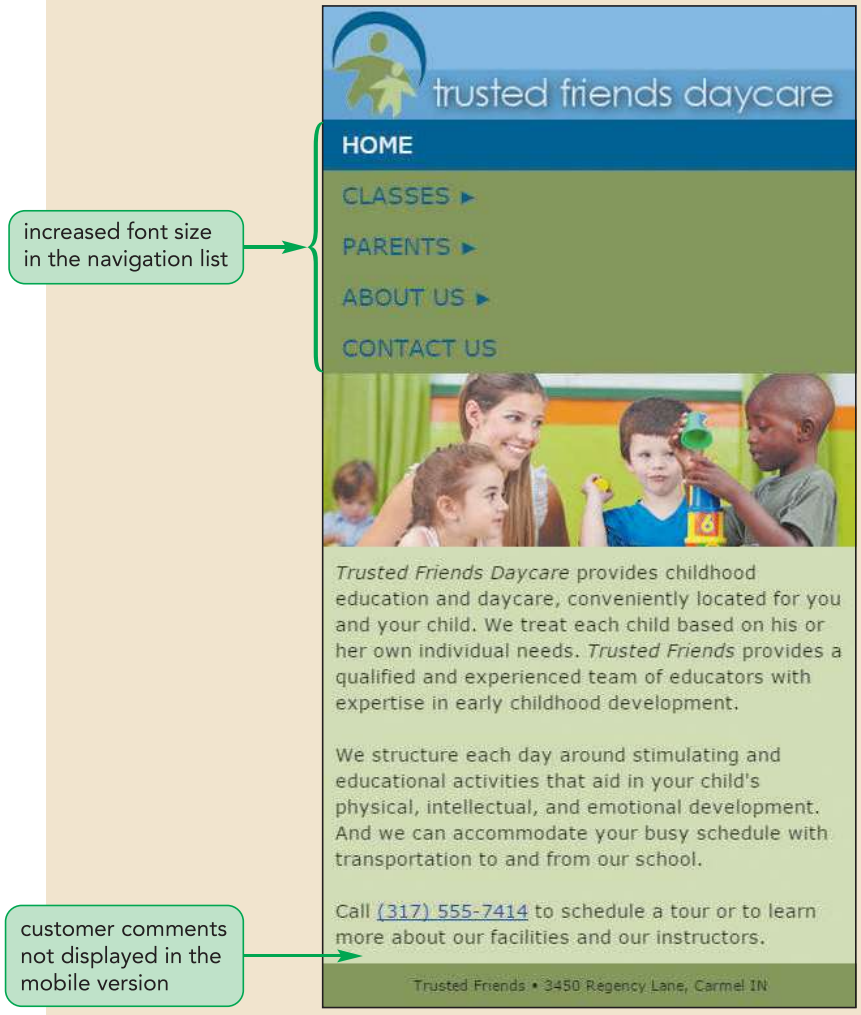
Figure 5–17 Hiding the aside element for mobile devices



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.

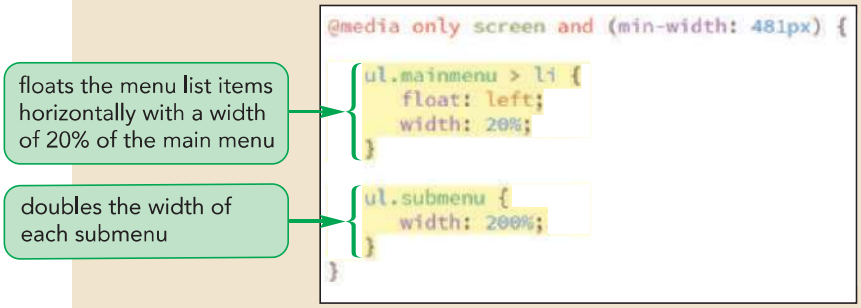
```
ul.mainmenu > li {
float: left;
width: 20%;
}
```

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%;
}
```

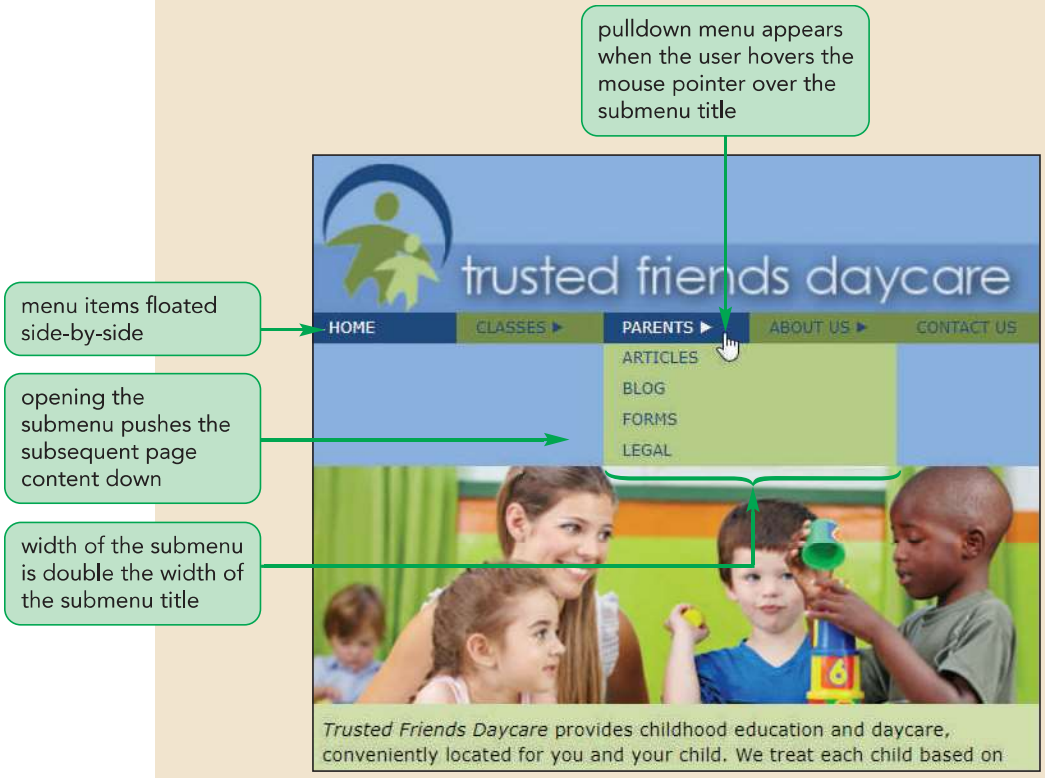
Figure 5–19 highlights the style rule within the media query for tablet devices.

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 Pulldown 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:

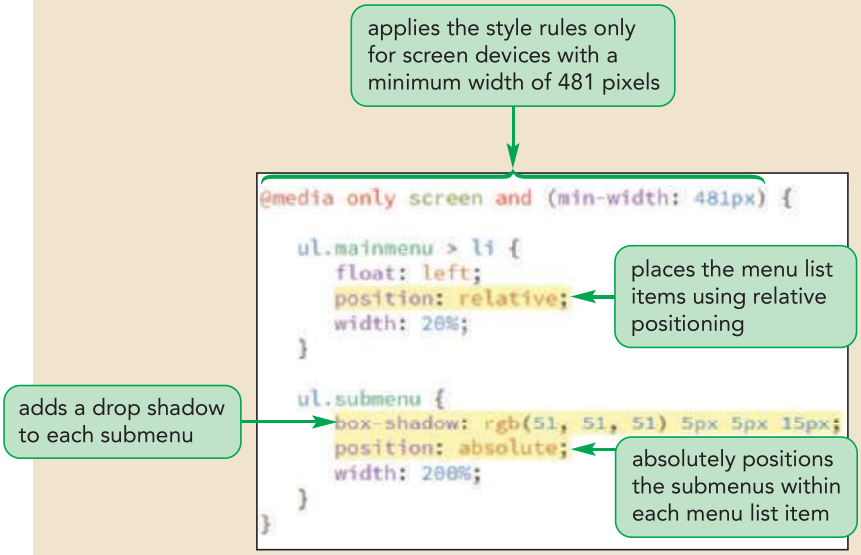
```
position: relative;
```

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



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

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:

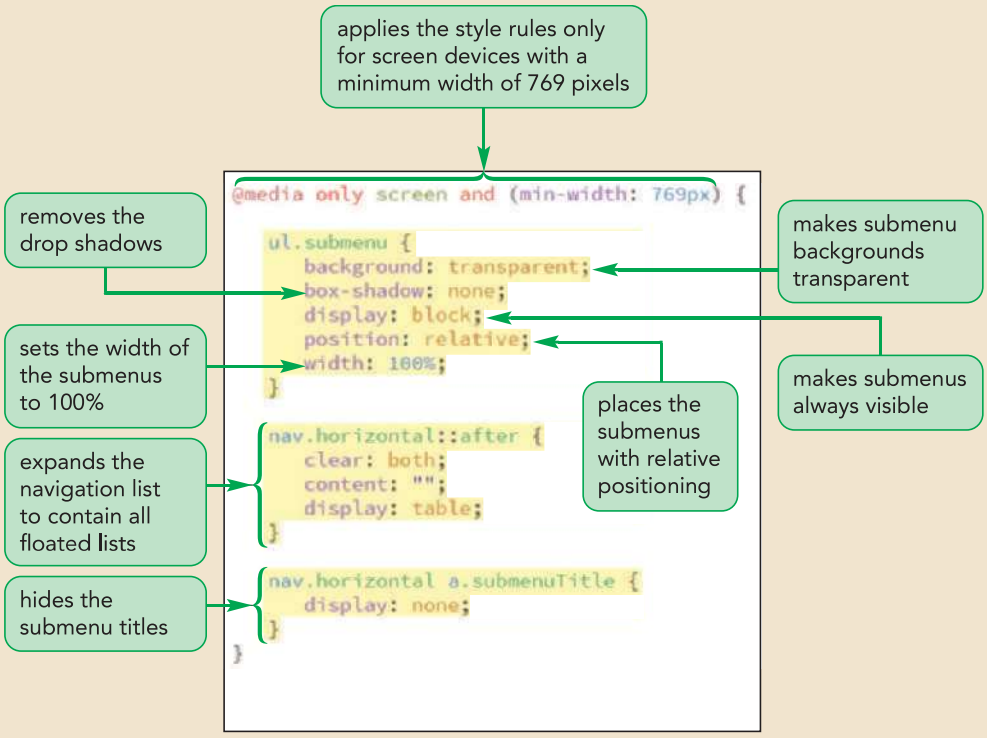
```
nav.horizontal::after {
   clear: both;
   content: "";
   display: table;
}
```

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 home page

navigation list expands to contain floated content

submenus are laid out horizontally and are always visible

article and aside elements are arranged in two columns

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.

REVIEW

Session 5.1 Quick Check

1. Which of the following is *not* a part of responsive design theory?
   a. flexible layouts
   b. pulldown menus
   c. image rescaling
   d. media queries
2. Which attribute do you add to a `link` element for aural browsers?
   a. `media = "aural"`
   b. `type = "aural"`
   c. `media = "speech"`
   d. `type = "speech"`
3. What `@rule` do you use for braille device?
   a. `@media braille`
   b. `@braille true`
   c. `@type braille`
   d. `@media nonscreen`
4. What `@rule` loads style rules for screen devices up to a maximum width of 780 pixels?
   a. `@screen: 780px`
   b. `@media screen and (width: 780px)`
   c. `@screen and (width <= 780px)`
   d. `@media screen and (max-width: 780px)`
5. What attribute would you add to a link element for screen devices whose width ranges from 480 pixels up to 780 pixels (inclusive)?
   a. `media="screen" min-width="480px" min-width="480px"`
   b. `media="screen and (width=480px - 780px)"`
   c. `minScreenWidth = "480px" maxScreenWidth = "780px"`
   d. `media="screen and (min-width: 480px and max-width: 780px)"`
6. In general, what media rules should be listed first in your media queries if you want to support mobile, tablet, laptop, and desktop devices?
   a. mobile
   b. tablet
   c. laptop
   d. desktop
7. Which viewport displays the web page content that fits within mobile screen?
   a. layout
   b. visual
   c. webpage
   d. browser
8. Which viewport contains the entire content of the page, some of which may be hidden from the user?
   a. layout
   b. visual
   c. webpage
   d. 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.

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.

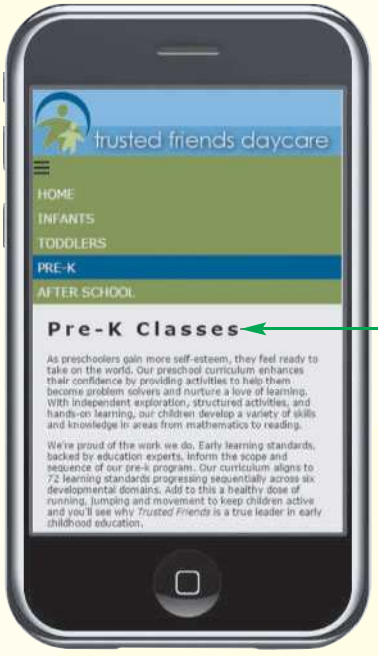
The **flex-shrink** value specifies how fast the item shrinks below its basis size relative to other items in the flexbox.

The **flex-grow** value specifies how fast the item grows above its basis size relative to other items in the flexbox.

The **flex-basis** value provides the basis or initial size of the item prior to flexing.

```
body {
   display: flex;
   flex-flow: row wrap;
}

header, footer {
   width: 100%;
}

aside {
   flex: 1 1 120px;
}

section#main {
   flex: 3 1 361px;
}

section#topics {
   display: flex;
   flex-flow: row wrap;
}

section#topics article {
   flex: 1 1 200px;
}
```

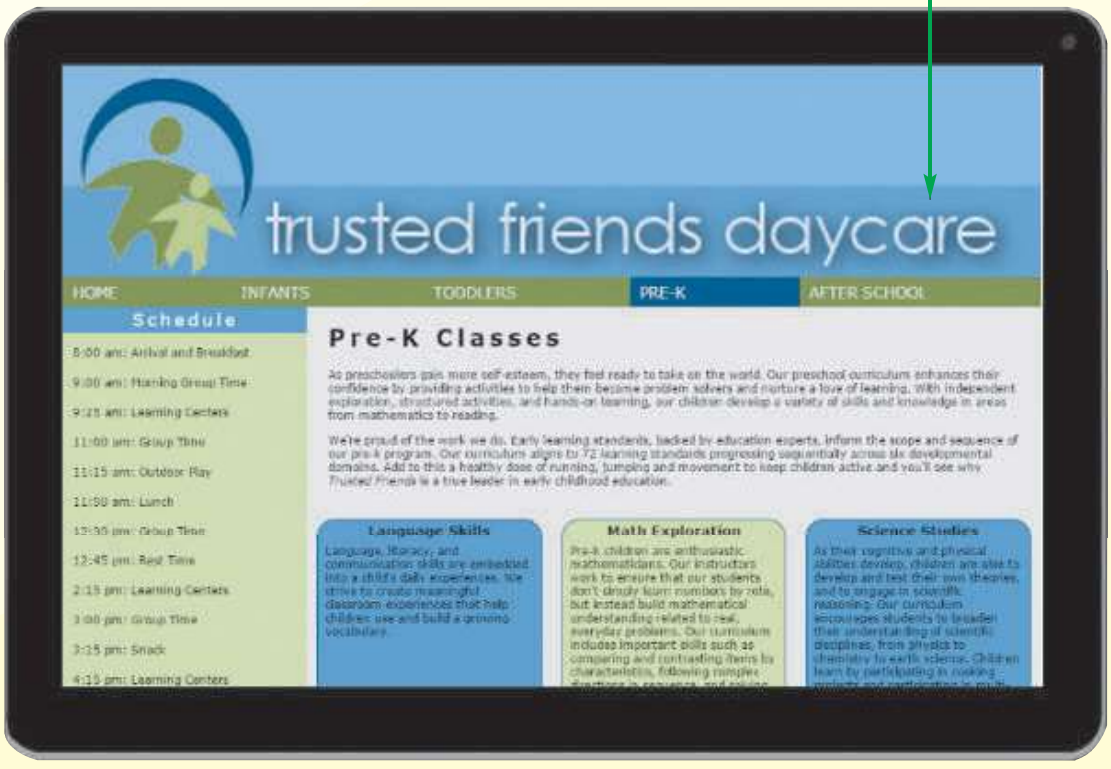
Flexible Layouts



BenBois/openclipart

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/openclipart

Introducing Flexible Boxes

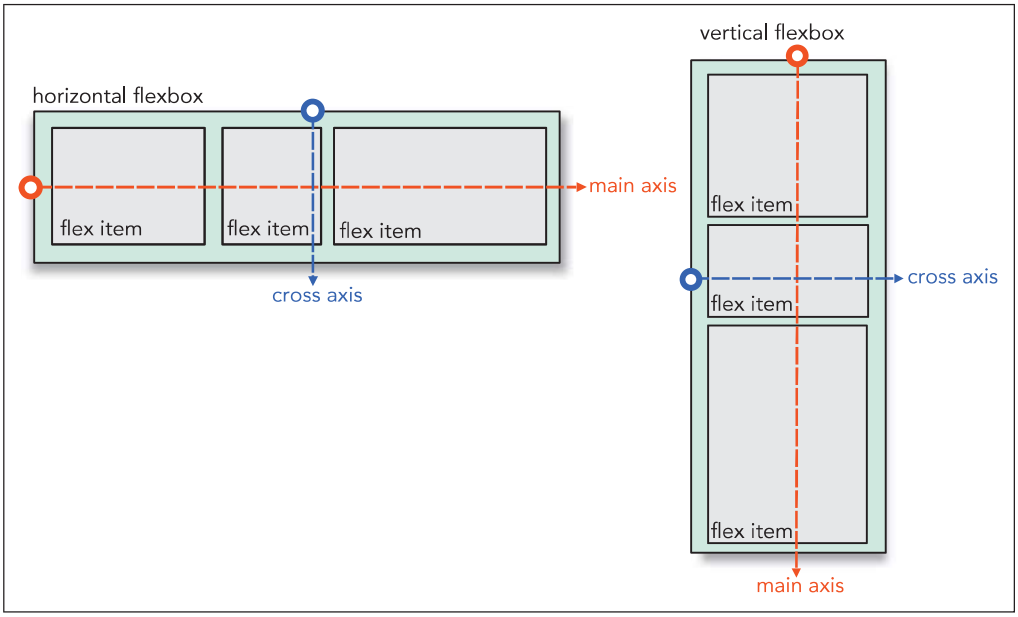
So far our layouts have been limited to a grid system involving floating elements contained within a fixed or fluid grid of rows and columns. One of the challenges of this approach under responsive design is that you need to establish a different grid layout for each class of screen size. It would be 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;
```

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

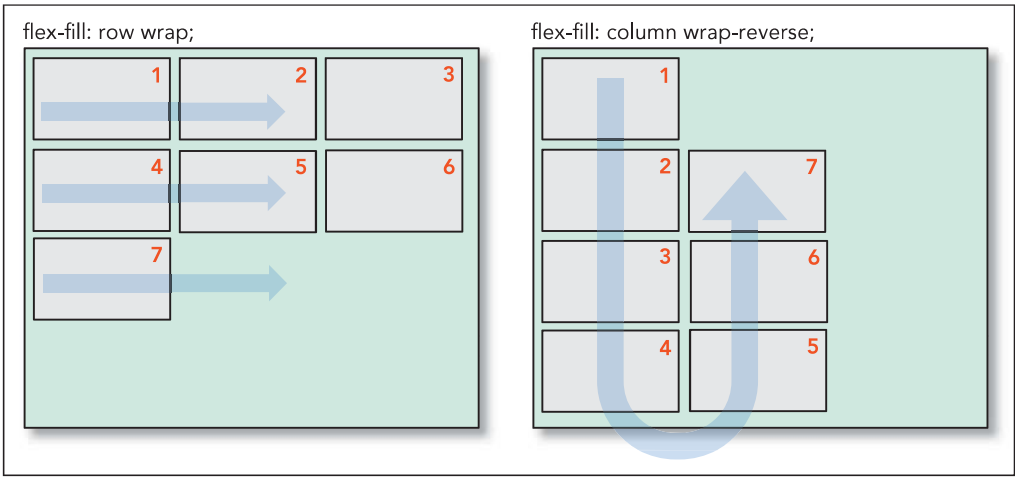
```
flex-flow: direction wrap;
```

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.

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.

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:

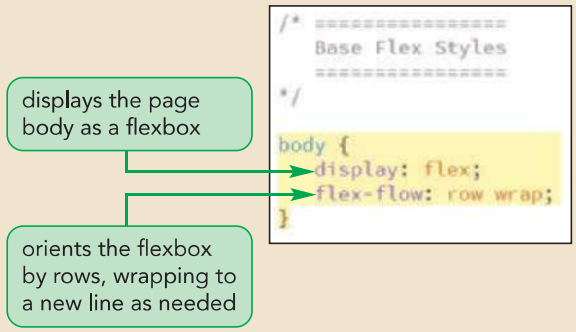
1. Use your editor to open the **tf_prek_txt.html** and **tf_flex_txt.css** files from the html05 ▸ tutorial folder. Enter ***your name*** and ***the date*** in the comment section of each file and save them as **tf_prek.html** and **tf_flex.css** respectively.
2. Return to the **tf_prek.html** file 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.

Setting the Flex Basis

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.

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

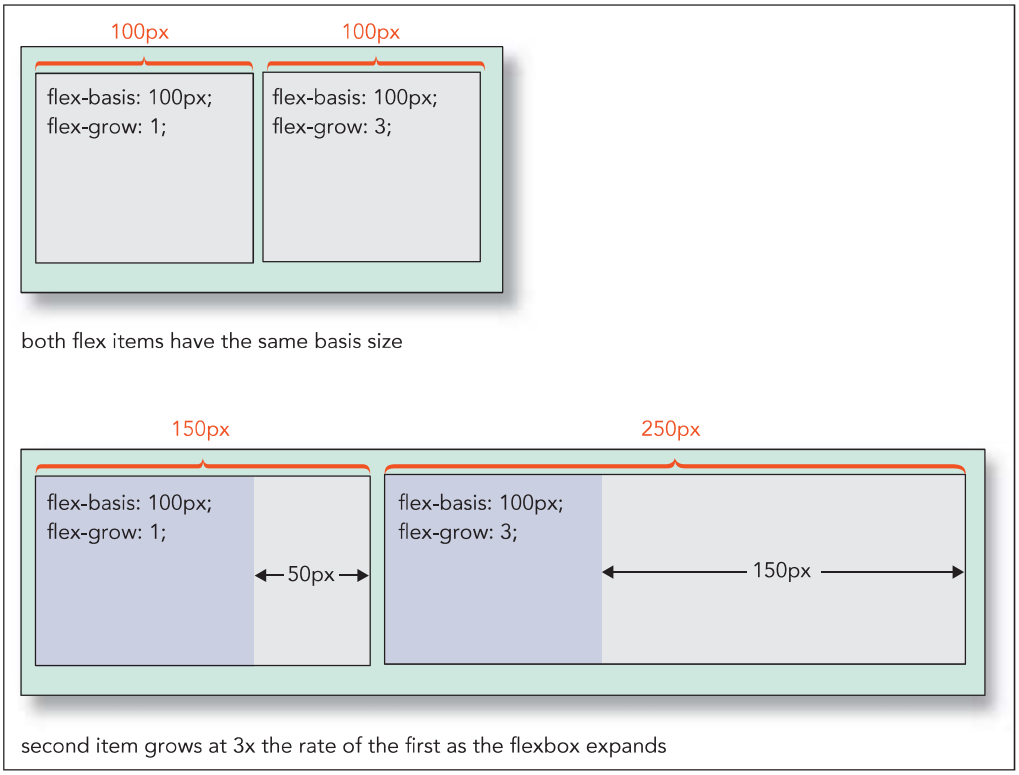
Once the basis size of the item has been defined, the browser will attempt to expand the item into its flexbox. The rate at which a flex item grows from its basis size is determined by the following `flex-grow` property

```
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;
}
```

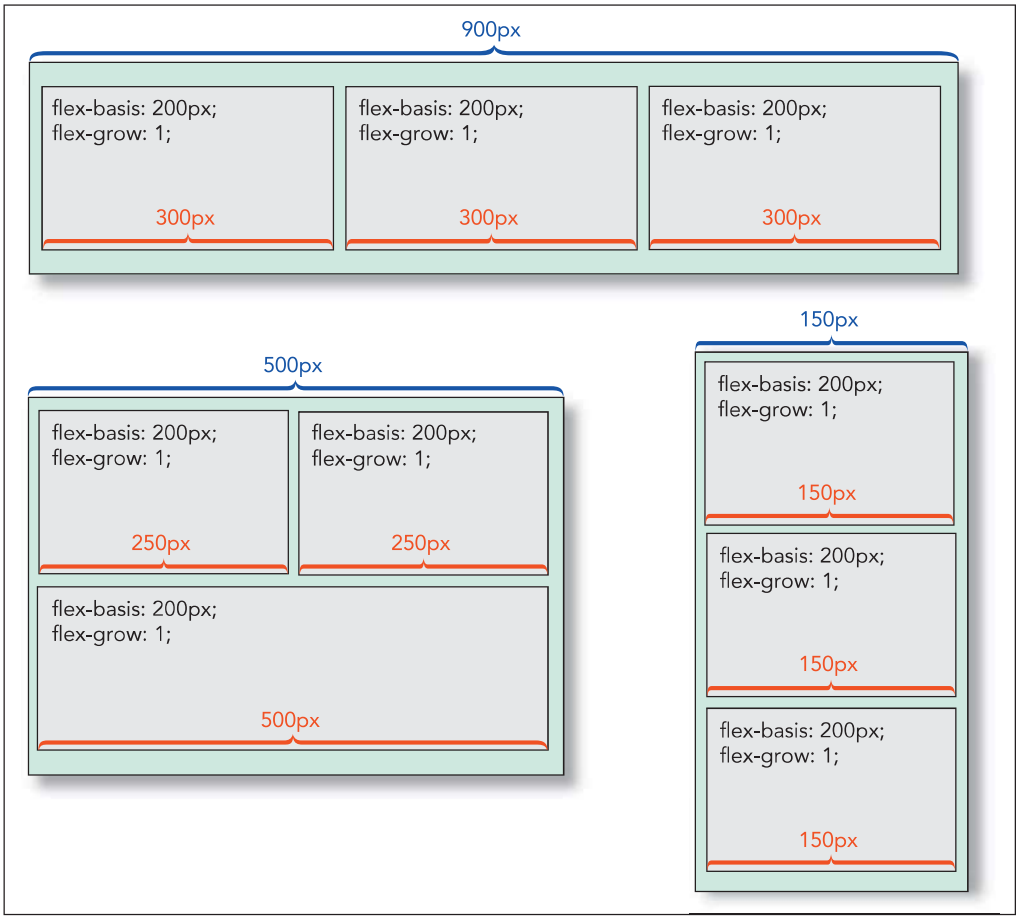
If there are four items in this navigation list, each will be 25% of the total list size and if at a later date a fifth item is added, those items will then be allotted 20% of the total size. Thus, unlike a grid layout, there is no need to revise the percentages to accommodate new entries in the navigation 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

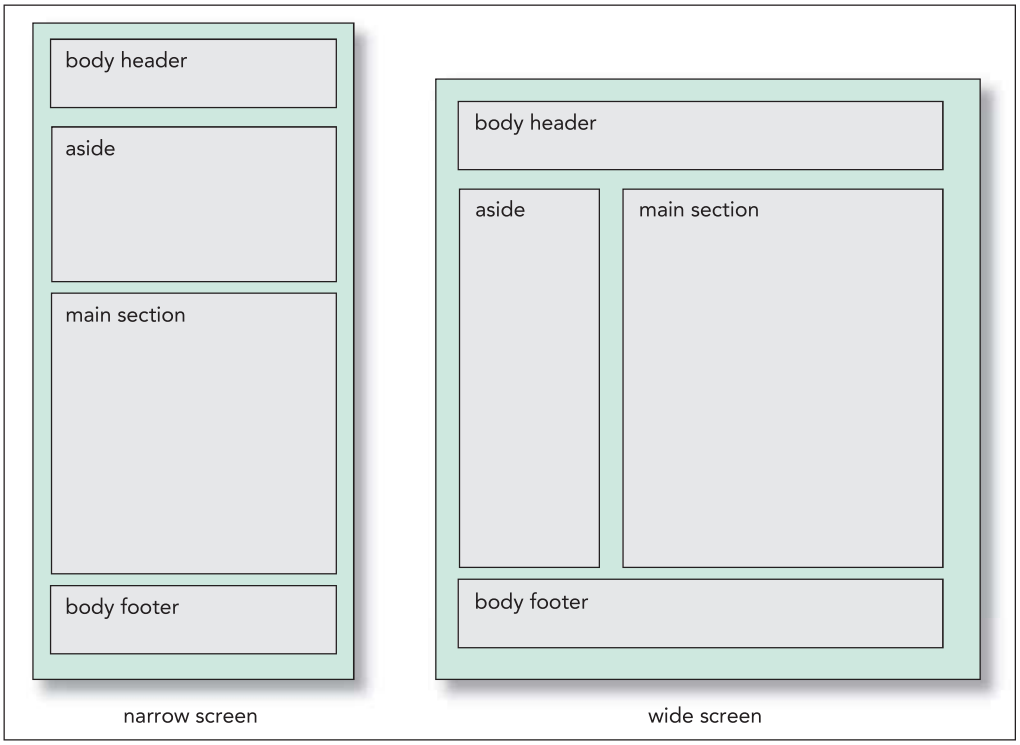
  ```
  flex: grow shrink basis;
  ```

  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.

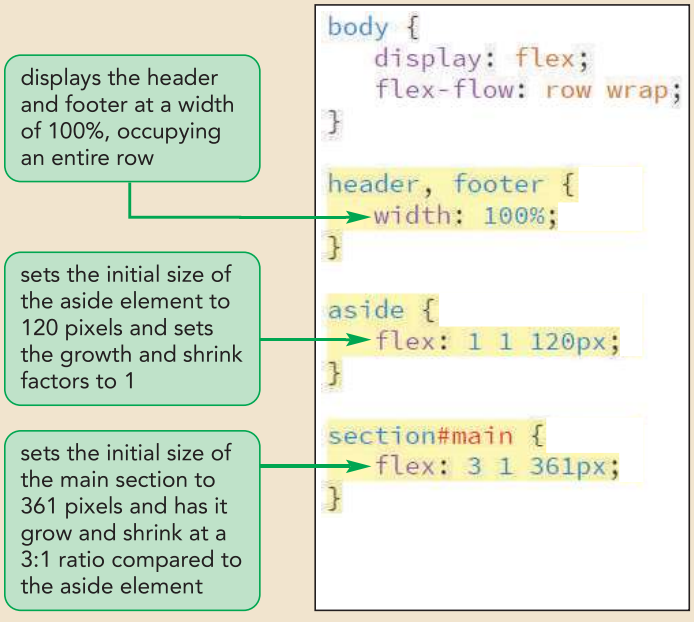
To define the flex layout:

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;
}
```

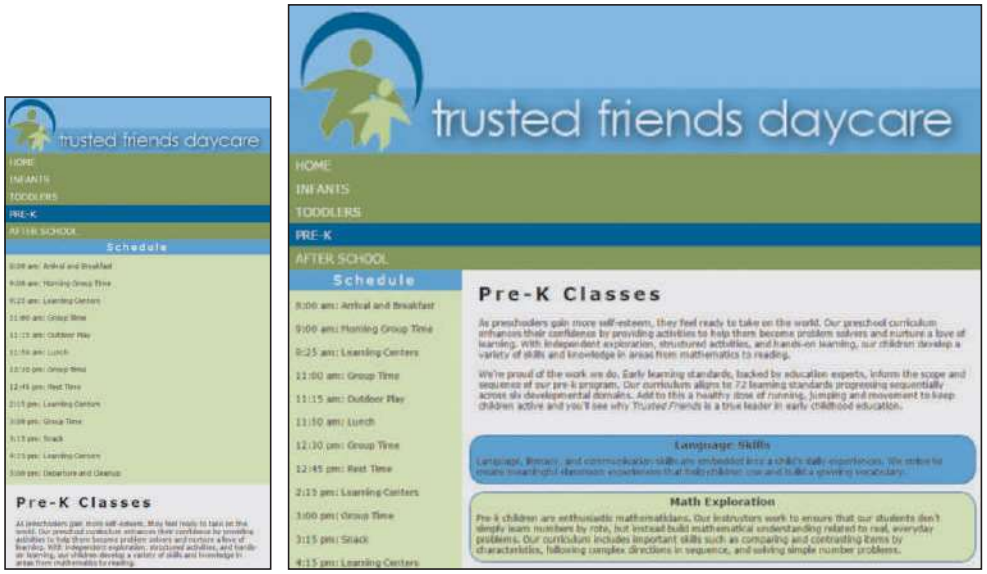
Figure 5–32 highlights the newly added style rules to define the flex item sizes.

Figure 5–32 Set the flex properties of the flex items in the page body



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.

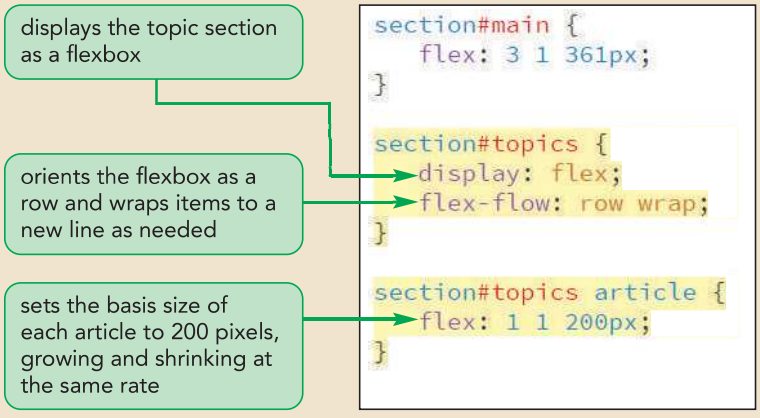
To lay out the topic articles and navigation list:

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;
   }

}
```

displays the unordered list as a flexbox

orients the flexbox in the row direction with no wrapping

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

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:

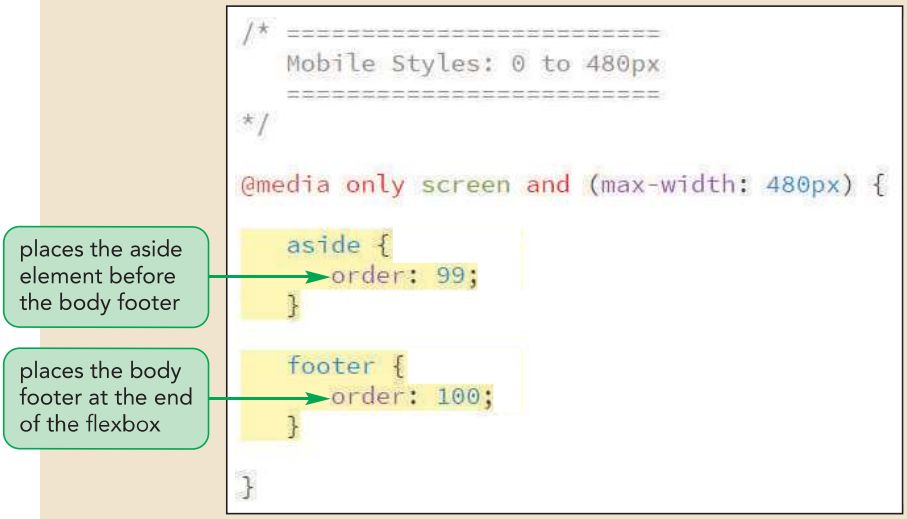
1. Return to the **tf_flex.css** file in your editor and go to the Mobile Devices media query.
2. Add the following style rules, indented to offset them from the braces in the media query:

```
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



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

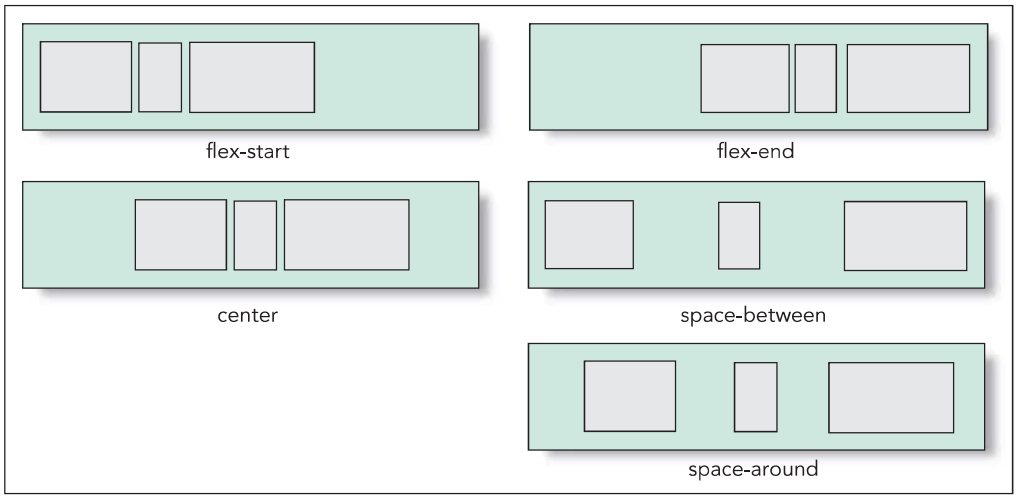
```
justify-content: placement;
```

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:

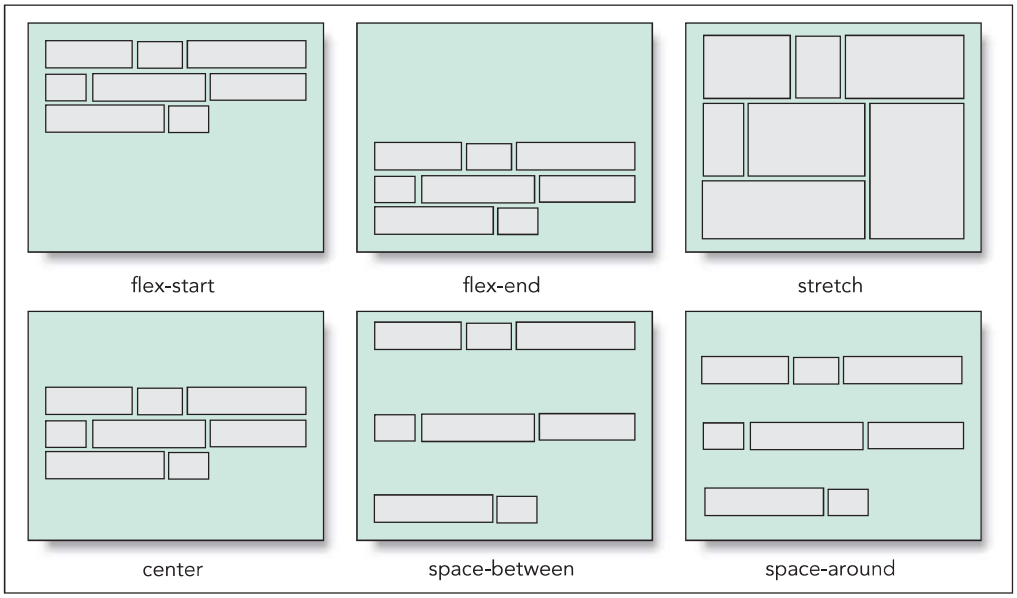
```
align-content: value;
```

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

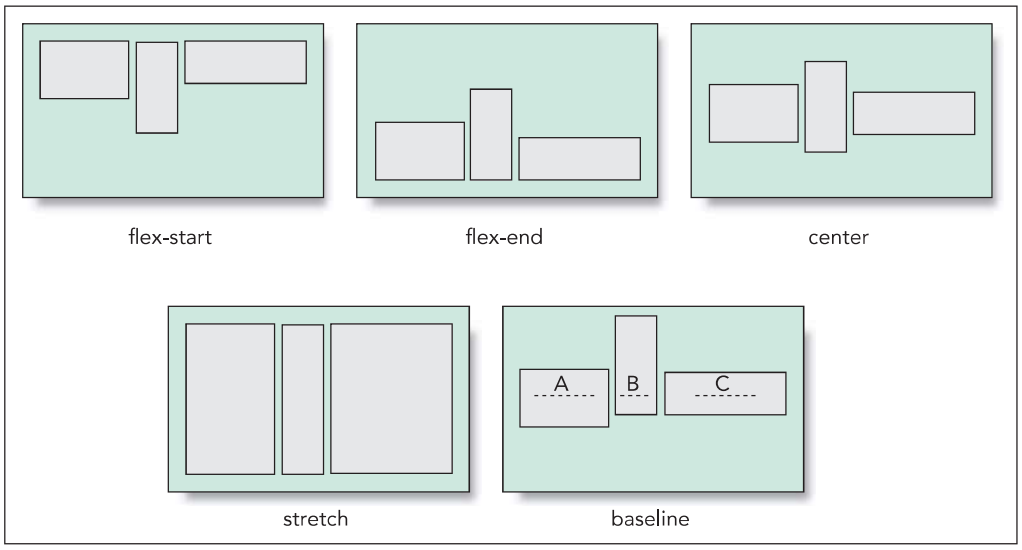
```
align-items: value;
```

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.

INSIGHT

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="#">
   <img src="tf_navicon.png" alt="" />
</a>
```

Figure 5–41 highlights the code to create the navicon.

Figure 5–41 Inserting the navicon

```
<nav class="horizontal">
   <a id="navicon" href="#"><img src="tf_navicon.png" alt="" /></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>
```

navicon image

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:

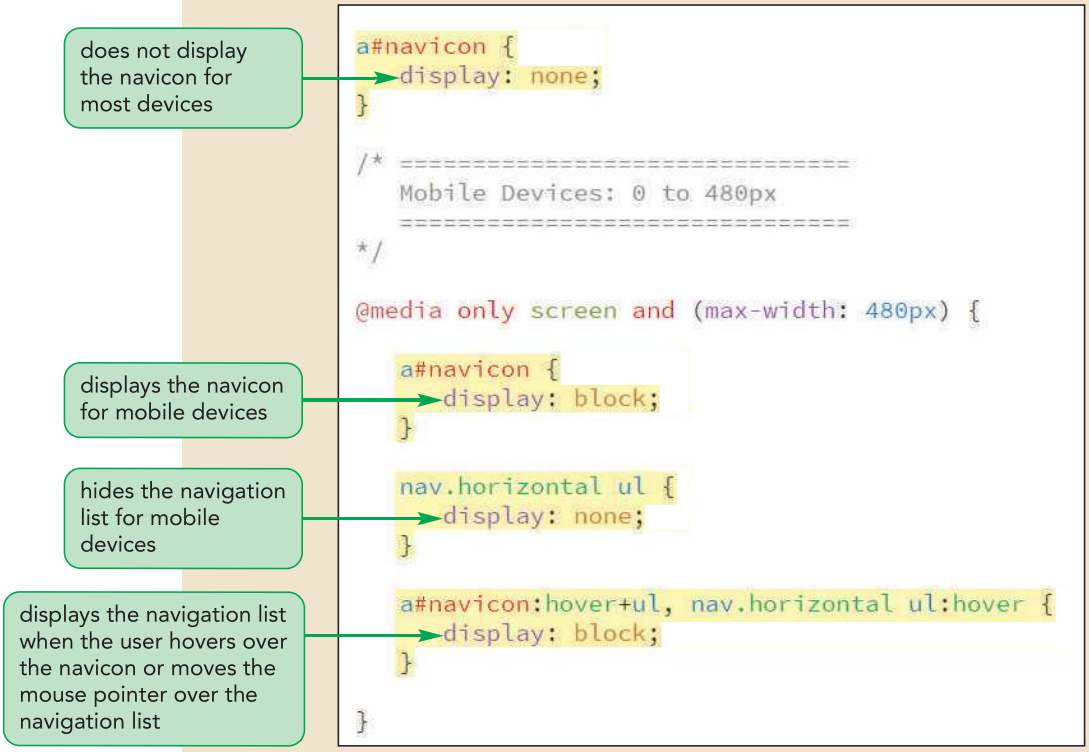
```
nav.horizontal ul {
   display: none;
}
```

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



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/openclipart

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.

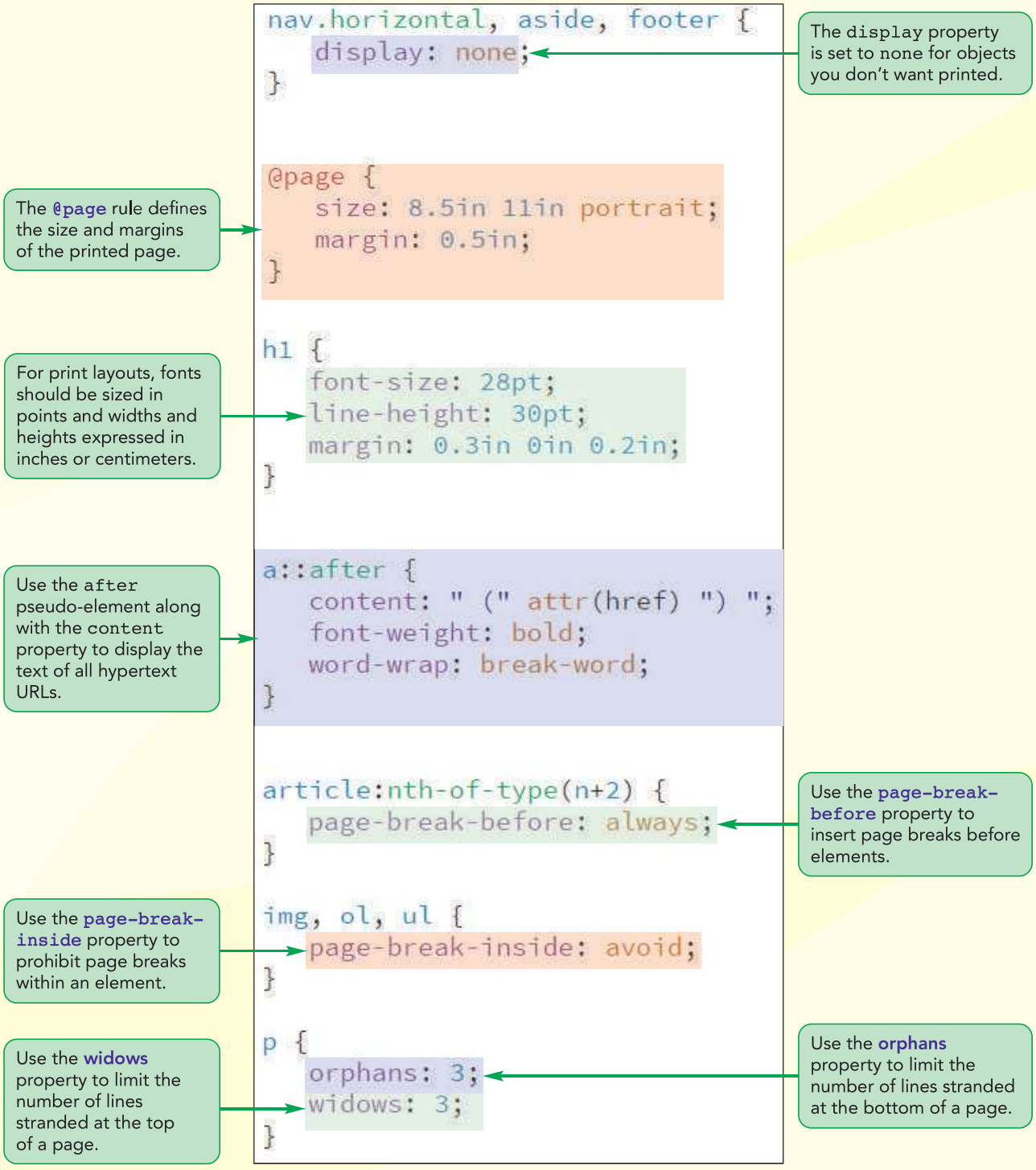
You've completed your work on the design of the Pre-K Classes page for Trusted Friends Daycare. In the next session, you'll explore other uses of media queries by designing a page for printed output. You may close your files now.

REVIEW

Session 5.2 Quick Check

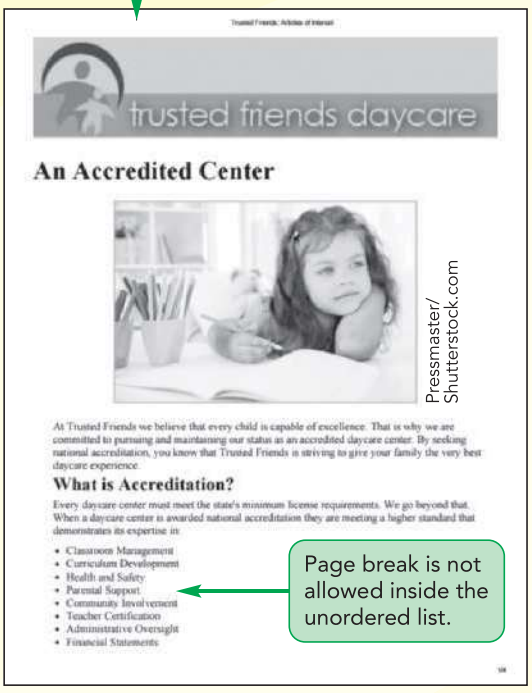
1. Which of the following is *not* a style to display an element as a flexbox?
   a. `display: -chrome-flex;`
   b. `display: -webkit-flex;`
   c. `display: -webkit-box`
   d. `display: -ms-flexbox;`
2. To display items within a flexbox in a column filled from the bottom upward, use:
   a. `flex-direction: column up;`
   b. `flex-direction: column-bottom;`
   c. `flex-direction: column-reverse;`
   d. `flex-direction: column-to-top;`
3. To set the initial size of a flexbox item to 250 pixels, use:
   a. `flex-size: 250px;`
   b. `flex-basis: 250px;`
   c. `flex: 250px;`
   d. `flex-from: 250px;`
4. To set the growth rate of a flexbox item to a rate of 4, use:
   a. `flex-rate: 4;`
   b. `flex: 4x;`
   c. `flex-growth: 4;`
   d. `flex-grow: 4;`
5. Which of the following sets the `div` element to be equal in size regardless of the size of the flexbox container?
   a. `div {flex: equal;}`
   b. `div {flex: 1 1 100px;}`
   c. `div {flex: 1 1 0px;}`
   d. `div {flex: 0 0 0px}`
6. To reorder of the placement of a flex item within its flexbox, use:
   a. `flex-reorder`
   b. `flex-move`
   c. `flex-basis;`
   d. `order`
7. To center flex items along the flexbox's main axis, use:
   a. `justify-content: center;`
   b. `align-content: center;`
   c. `flex-position: center;`
   d. `flex-main: center;`
8. To center flex items along the flexbox's cross axis, use:
   a. `justify-content: center;`
   b. `align-content: center;`
   c. `flex-position: center;`
   d. `flex-main: center;`

Session 5.3 Visual Overview:



Print Styles

Page size is set at 8.5 inches by 11 inches with a 0.5 inch margin in portrait orientation.



trusted friends daycare

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

Page break is not allowed inside the unordered list.

page 1

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. But 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?

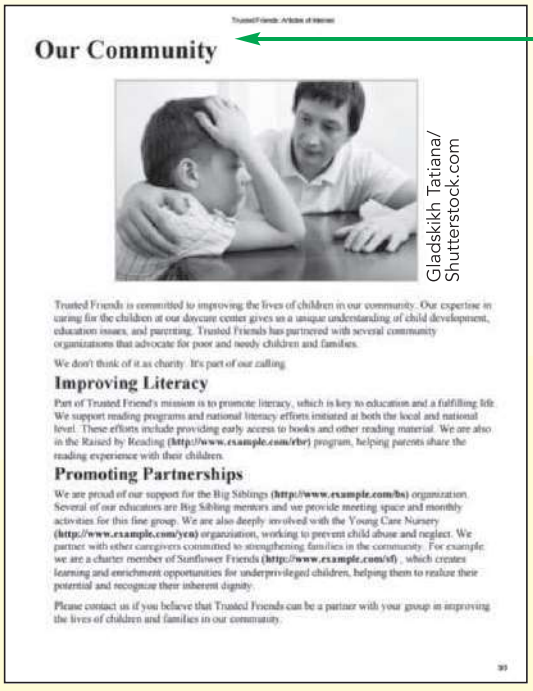
There are several national organizations that provide accreditation services. Who a center chooses for oversight is important. Trusted Friends pursues national accreditation from three of the most respected national early childhood accreditation agencies:

- 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.

Hypertext URLs are displayed in bold after the hypertext link.

page 2



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 Friend's 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/bs)** 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.

Page break is inserted before the article element, starting it on a new page.

page 3

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

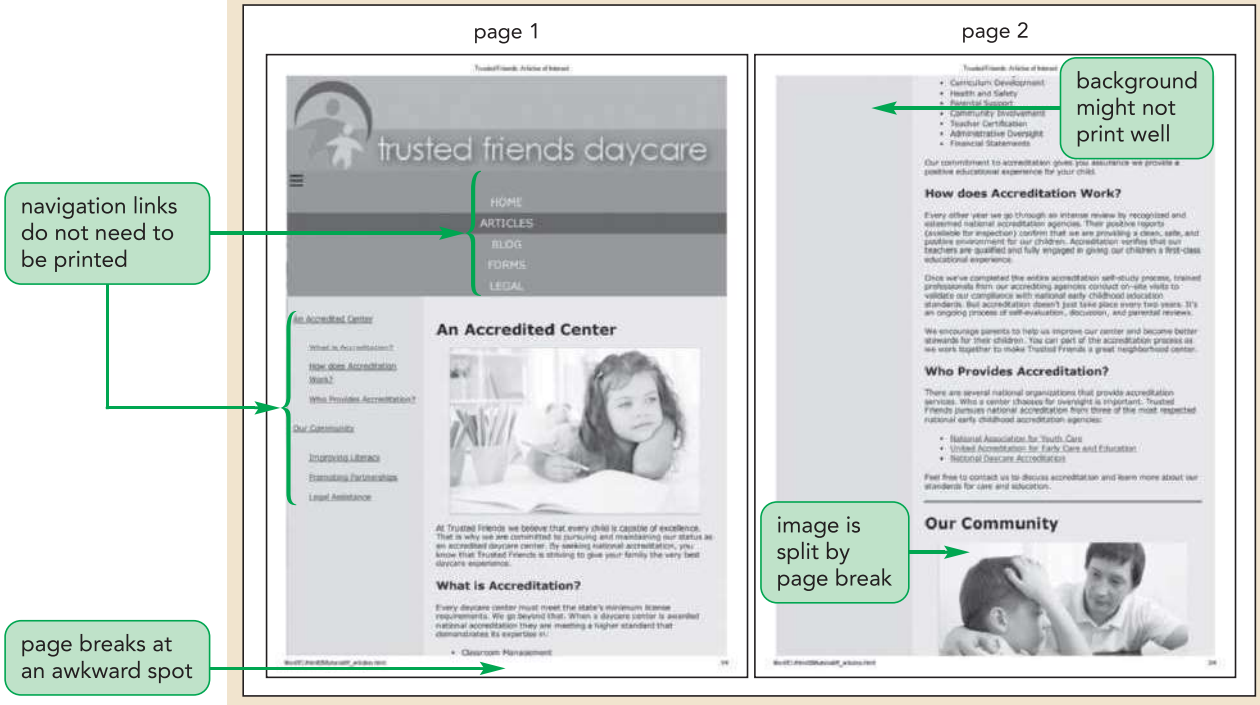
Marjorie has created a page containing articles of interest for parents at Trusted Friends Daycare. She has already written the page content and the style sheets for mobile, tablet, and desktop devices. Open the articles document now.

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.

To avoid mixing screen styles with print styles, identify styles common to both devices with the media type *all*.

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.

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 all devices

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.

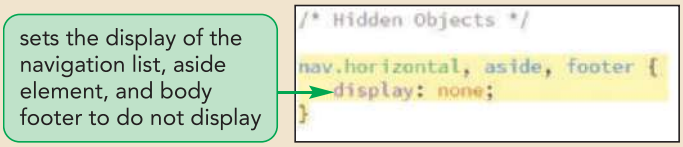
To hide elements in the print version:

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



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;
```

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;
}
```

TIP

Users can override the page sizes and orientations set in `@page` rule by changing the options in their print dialog box.

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.

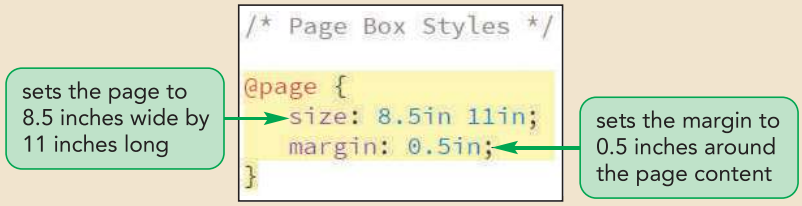
To define the printed page size:

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:

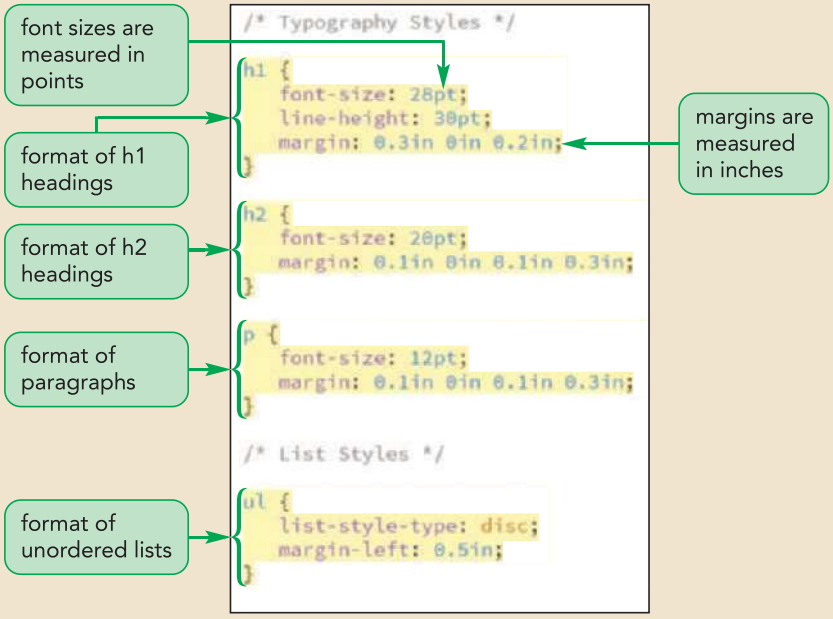
```
h1 {
   font-size: 28pt;
   line-height: 30pt;
   margin: 0.3in 0in 0.2in;
}

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



Next, you'll format the appearance of images on the page.

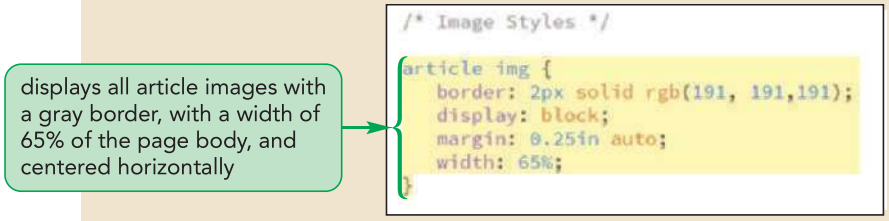
To format the printed images:

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



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

print version of the images

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.

REFERENCE

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.

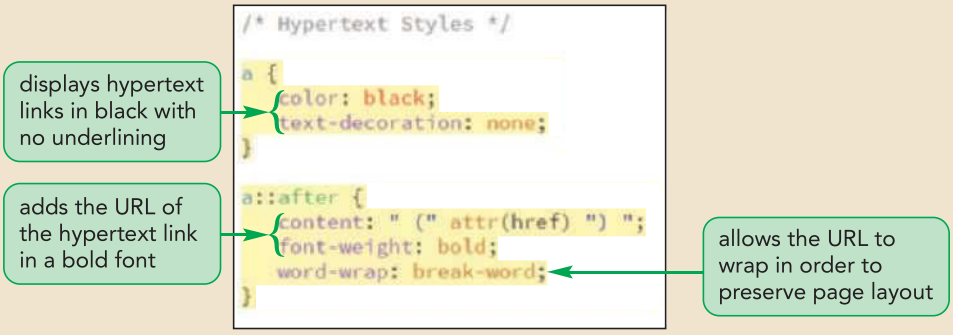
To format the hypertext links:

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



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

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.

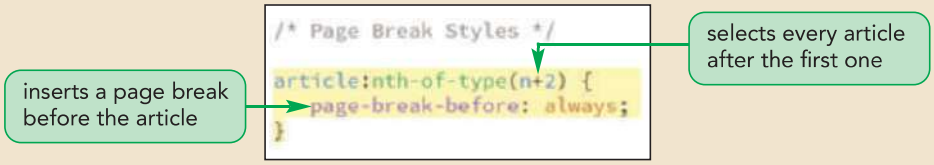
To print each article on a new page:

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.

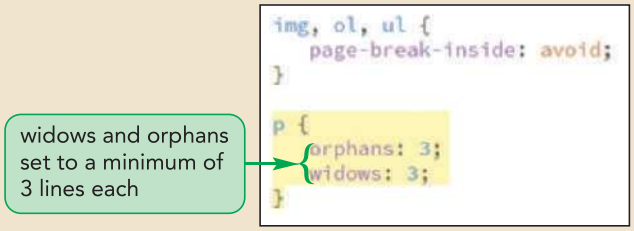
To avoid widows and orphans:

1. Within the Page Break Styles section of the **tf_print.css** file, add the following style rule.

```
p {
   orphans: 3;
   widows: 3;
}
```

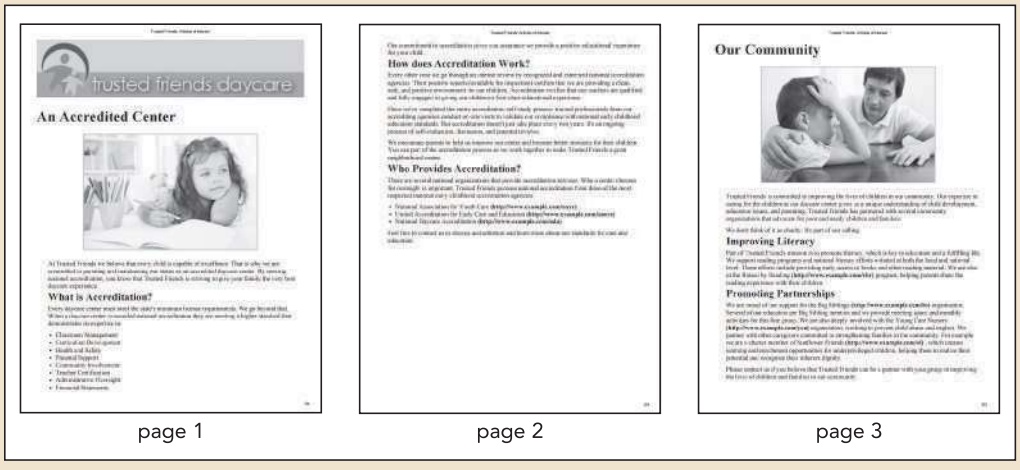
Figure 5–55 highlights the style rule for setting the size of widows and orphans.

Figure 5–55 Setting the size of widows and orphans



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



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.

REVIEW

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)`