**8**

**How to use Responsive**

**Web Design**

In the first seven chapters of this book, you've learned how to create and format web pages that will be displayed on desktop computers. Because more and more websites are being accessed on mobile devices, though, it's important to know how to develop pages that adapt to smaller screen sizes. To do that, you can use the principles of Responsive Web Design.

|  |  |
| --- | --- |
| **Introduction to Responsive Web Design** | **282** |
| The three components of a responsive design | 282 |
| How to test a responsive design | 284 |
| **How to implement a fluid design** | **286** |
| Fluid layouts vs. fixed layouts | 286 |
| How to convert fixed widths to fluid widths | 288 |
| How to size fonts | 290 |
| How to scale images | 292 |
| A web page with a fluid design | 294 |
| **How to use CSS3 media queries** | **296** |
| How to control the mobile viewport | 296 |
| How to code media queries | 298 |
| Common media queries for a responsive design | 300 |
| How to build responsive menus with the SlickNav plugin | 302 |
| **A web page that uses Responsive Web Design** | **304** |
| The design of the web page | 304 |
| The HTML for the web page | 306 |
| The CSS for the web page | 308 |
| **Perspective** | **314** |

**282** *Section 1 A crash course in HTML and CSS*

**Introduction to Responsive Web**

**Design**

In chapter 1, you learned that a website that uses *Responsive Web Design (RWD)* adapts gracefully to the size of the screen where it's displayed while maintaining the overall look-and-feel of the site. Now, you'll learn about the three components that you use to implement a *responsive design.* In addition, you'll learn about some of the ways that you can test a website that uses a responsive design.

**The three components of a responsive design**

Figure 8-1 describes the three components of a responsive design. To start, a responsive design *uses fluid layouts* rather than fixed layouts. That way, the

width of the web page and its structural elements adjust to the size of the screen. In this figure, for example, you can see that the web page always fills 100% of the screen width. In addition, if you reduce the size of the browser window on the desktop, the width of the main content and the sidebar always occupy the same percent of the window.

Although you can use a fluid layout to adjust the widths of the structural elements on a page, you can't use them to adjust the appearance of the page. To do that, you use *media queries.* Media queries let you change the appearance of a page based on conditions, such as the width of the screen. In this figure, for example, you can see that the page looks different on mobile devices than it does on the desktop. That makes it easier to use on these devices.

Finally, a responsive design should use images whose sizes change along with the size of their containing elements. For example, the page shown here uses a *scalable image* for the image in the main content of the page. That way, as the size of the element that contains the main content changes, the size of the image changes.

*Chapter 8 How to use Responsive Web Design* **283**

**A website that uses Responsive Web Design**

Ai Gift Adoption Cards by E- x

C [gifts.worldwildlife.org/gift-center/gifts/Gift-Adoption-Cards](http://gifts.worldwildlife.org/gift-center/gifts/Gift-Adoption-Cards)

•



•

**WWF**

|  |  |
| --- | --- |
| Gift Center  Make a donation in support of WWFs global conservation efforts and choose a thank-you gi | *Need Assistance,*  *Call* 1-800-CALL-WWF *or Chat Lhie*  Free Global Shipping ti 0 items - $0 |

Gift Adoption Cards

GIFT OPTIONS

Species Adoptions Buckets

Apparel and More Frontline Hero Dolls The Art of Wildlife **NEW!** Gift Adoption Cards

Adoption of the Month Club Monthly Symbolic Adoptions Group Giving Clubs

Make a $55 donation in honor of a friend, family



the perfect gift in an orite virtual card design Then, choose your

Gift Center

iike a donation In support of WWFs global inset-v.11ton effocu and choose a thank-you gift

**Free Global Shipping**

0 items - SO

**SEE ALL GIFT OPTIONS** •■•

:.C.. A —1 —

VI

•••• ATV UI **9:24PM 0 41%O**

[gifts.worldwildlife.org](http://gifts.worldwildlife.org)

**WWF**

**DONATE > ADOPT >**

ur gift recipient will be donation for any $55 a great last-minute gift

*d and want to check your*

•   
ww0

•

|  |  |
| --- | --- |
| **DONATE** | **ADOPT >** |

Gift Center

Make a donation in support of WVO's gkbal conservavon efforts **and choose a thank•you pft.**

**The three components of Responsive Web Design**

**Figure 8-1 The three components of Responsive Web Design**

* You use a *fluid layout* to adjust the width of a web page and its structural elements to the size of the screen.
* You use *media queries* to adjust the appearance of a web page to different device sizes.
* You use *scalable images* so the size of an image is scaled to the size of the element that contains it.

**Description**

* Websites that are developed using *Responsive Web Design* are designed to adapt gracefully to the size of the screen on which the site is being displayed.

**284** *Section 1 A crash course in HTML and CSS*

**How to test a responsive design**

As you develop a website using a responsive design, you'll want to test it in devices of various sizes to be sure it works as expected. Figure 8-2 describes three ways you can do that.

The best way to test a responsive design is to deploy the website to a server and then test it on as many devices as possible. Because so many different devices with different screen sizes are available, though, that may not be practical. So instead, you may want to use a web-based tool such as ProtoFluid that lets you view a web page in the screen sizes used by many different devices.

Another option is to use the device emulators and browser simulators that are available for many of the most popular mobile devices and browsers. To do that, you typically need to download the emulator or simulator from the manufacturer's website so you can run it on your desktop. In a few cases,

though, you can run an emulator or simulator online. But before you can do that, you must first deploy the website to a server so it can be accessed online.

A simpler way to test a responsive design is to use the developer tools that are provided by most modern browsers. To access these tools, you display a page in the browser and then right-click on the page and select Inspect Element. You can also press F12 to access these tools in Chrome, **IE,** and Firefox.

When you access the developer tools, they're typically displayed at the bottom of the browser window as shown in this figure. Then, you can use techniques that are specific to each browser to display the page in various devices. In Chrome, for example, you click the Toggle Device Mode icon in the toolbar as shown here and then select a device from the drop-down list at the top of the window.

Although the developer tools are useful as you're developing a website, they also have their shortcomings. In **IE,** for example, the options you have to choose from aren't intuitive. And in Firefox, you can't select specific devices. Instead, you have to select from a list of device sizes. Because of that, you should also

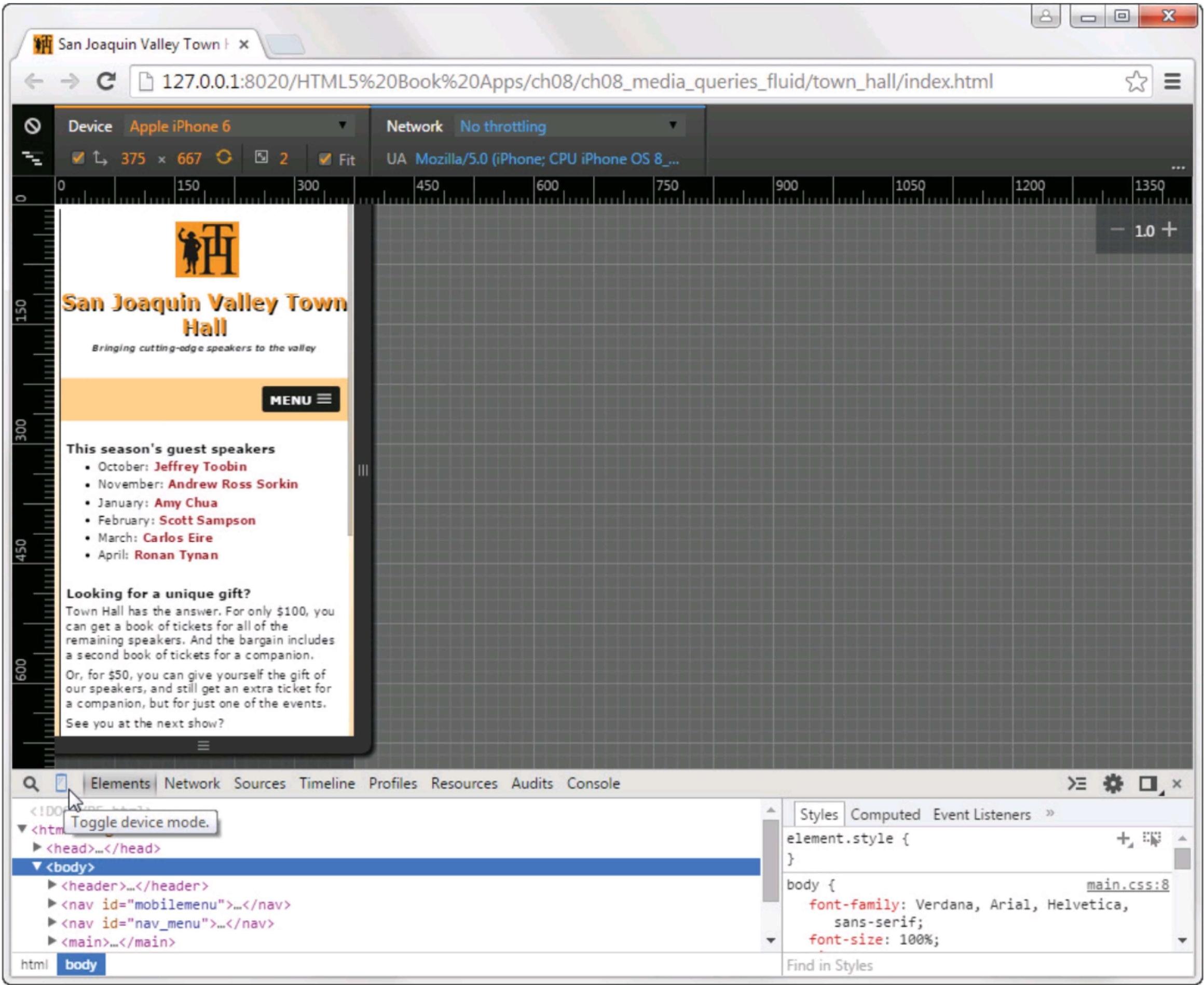
test your website on all common devices after you've deployed it.

*Chapter 8 How to use Responsive Web Design* **285**

**Figure 8-2 How to test a** responsive design

**A web page displayed using the developer tools in Chrome**

**Three ways to test a responsive design**



*s.* ***11111)I,*** *cue trip-Woe speakers* to me eater

ioaquill **'Miley Town** Hall

**Q 0 Network Sources Timeline Pion es Resources Audits Console**

**<htirLioggle device mode.**

* **<head,....</head>**
* **<body>**
* **<header >....</header>**
* **<nay id="mobilemenu">—< /nay>**
* **<nay id="nav\_menu">.-</nav>**
* **<main>—</rnain>**

**>3**

**] Styles ] Computed Event Listeners element .style**

**body ( main.css:8**

**font-family: Verdana, Arial, Helvetica, sans-serif;**

**font-size: 100%;**

html

**body**

**Find in Styles**

L-11 l....4=3Mir

**'Pi San Joaquin Valley Town r x**

C 127.0.0.1:8020/HTML5%20Book%20Apps/ch08/ch08\_media\_queries\_fluid/town\_ha'

**This season's guest speakers**

* Ottobe- Jeffrey Tooln
* **November. Andrew Ross** Sorkin
* **January:** Amy Chua
* **February: Scott** Sampson
* **March: Carlos** Eire
* April: Ronan **Tynan**

**Looking for a unique gift?**

**Town Ha.. has to ans..,er. For only 0100, you can get a book of tickets *for* all of the remaining speakers. And the bargain includes a second book of tickets for a companion.**

On **for S50, you can give yourself the gift of our speakers, and still get an extra ticket** for **a companion. but for just one of the events.**

**See you at the next show?**

* Deploy your website to a server and then test it on each device or use a web-based tool like ProtoFluid to test it.
* Use device emulators and browser simulators for the various devices and browsers you want to test.
* Use the developer tools provided by most modern browsers.

**How to use a browser's developer tools to test a responsive design**

* Display the page in a browser and then press F12 or right-click on the page and select Inspect Element to display the developer tools.
* In Chrome or Opera, click the Toggle Device Mode icon near the left side of the devel­oper tools toolbar. Then, select the device you want to emulate from the drop-down list at the top of the page. You can also drag the edges of the screen to create a custom size.
* In IE, click the Emulation tab and then select from the options that are available.
* In Firefox, click the Responsive Design Mode icon near the right side of the developer tools toolbar. Then, select a device size from the drop-down list at the top of the page, or drag the edges of the screen to create a custom size.

**286** *Section 1 A crash course in HTML and CSS*

**How to implement a fluid design**

In chapter 6, you learned the basics of creating a fluid layout. Now, I'll review those skills and present some additional skills for developing *a fluid design* that uses a fluid layout, relative font sizes, and scalable images.

**Fluid layouts vs. fixed layouts**

To help you understand how fluid layouts work, figure 8-3 compares them to fixed layouts. Here, you can see the layout of a page with a fixed width of 960 pixels. This page includes a header and a footer that occupy the entire width of

the page, along with the main content for the page, which is 600 pixels wide, and a sidebar, which is 360 pixels wide.

When you use fixed widths like this, the width of the page and its structural elements stay the same regardless of the screen size. Because of that, you'll want to use fluid layouts when you develop a responsive design. That's true even if you use media queries, as you'll see later in this chapter.

In contrast to a page that uses a fixed layout, a page that uses a fluid layout adjusts to the size of the screen automatically. For example, the illustration in this figure indicates that the width of the page using a fluid layout is 90%. In other words, the page will fill all but 10% of the screen. Then, the widths of the elements within the page always add up to 100%.

For example, the width of the element that contains the main content is 62.5%, and the width of the sidebar is 37.5%. Note that these percents are relative to the size of the page, not the size of the screen. So in this case, the width of the element that contains the main content is 56.25% of the screen width (.625 x .90), and the width of the sidebar is 33.75% of the screen width (.375 x .90).

*Chapter 8 How to use Responsive Web Design* **287**

Figure 8-3 Fluid layouts vs. fixed layouts

**A comparison of fixed and fluid widths in a two-column layout**

Fixed width: 960px   
Fluid width: 90%

|  |  |  |  |
| --- | --- | --- | --- |
|  | Fixed width: 960px  Fluid width: 100% | |  |
| Fixed width:  Fluid width: | 600px  62.5% | Fixed width:  Fluid width: | 360px  37.5% |
|  | Fixed width: 960px  Fluid width: 100% | |  |

**The benefits of using fluid layouts**

* Page layouts are proportional to the size of the screen, so they will fill the screen equally at all sizes.
* They are scalable, so when new screen sizes become available in the future, they will automatically adapt to those new sizes.
* They can be used without media queries in some cases, although that's not common.

**Description**

* A *fixed layout* uses absolute measurements to specify the widths of a page and its main structural elements.
* A *fluid layout* uses percentages to specify the widths of a page and its main struc­tural elements. This meets the challenge of a web page adapting gracefully to all screen sizes, since the layouts are proportional rather than fixed.

**288** *Section 1 A crash course in HTML and CSS*

**How to convert fixed widths to fluid widths**

If you've already developed a web page that uses fixed widths, you'll need to know how to convert the widths in pixels to widths in percents. To do that, you can use the formula shown at the top of figure 8-4. Here, *target* is the width of the element in pixels that you want to convert, and *context* is the width of its containing element in pixels.

The example in this figure illustrates how this formula works. First, the HTML includes the structural elements for a page like the one shown in the previous figure. Then, the CSS shows how the widths of those elements would be converted to use a fluid layout. Note in this example that unlike the example in the previous figure, the header, main, aside, and footer elements all include left and right padding. If you add this padding to the width of the element, though, you'll see that it adds up to the width shown in the previous figure.

For example, the fixed widths of the header and footer elements are 930 pixels plus left and right padding of 15 pixels, for a total of 960 pixels. The fixed width of the main element is 570 pixels plus left and right padding of 15 pixels, for a total of 600 pixels. And the fixed width of the aside element is 330 pixels plus left and right padding of 15 pixels, for a total of 360 pixels.

Because each of these elements is contained within the body element, you calculate its width as a percent by dividing its width in pixels by the width of the body in pixels. Here, the width of each element and its left and right padding is divided by 960 pixels as indicated by the comments on the properties that set the widths and padding for these elements. Remember, though, that the context is always the containing element. So, for example, if the main element in this figure contained additional structural elements, the widths of those elements would be calculated based on the width of the main element, not on the width of the body element.

As you review these calculations, notice that they include all the decimal places from the result. Although you might be tempted to round these results, we don't recommend that. That's because you want these values to reflect the widths of the elements as accurately as possible.

Now, take a look at the two properties that are highlighted for the body element in this figure. As described in the previous figure, the width of the body is set to 90%, which means that the page will always take up 90% of the width of the screen. Although you can set the width to whatever you want, it's typically set between 90% and 100%.

In addition to the width property, the max-width property is set to a fixed width of 1024 pixels. Because of that, the page won't expand beyond this width even if the browser window can accommodate a wider page. This is useful because most fluid pages don't look good when displayed beyond a given width, and many monitors provide for much wider widths.

Although you can specify widths, padding, and margins as percents, you should realize that you can't specify the width of a border as a percent. Because of that, you may sometimes have to adjust the widths of one or more elements within a page if the page contains left or right borders.

*Chapter 8 How to use Responsive Web Design* **289**

**A formula for converting pixels to percents**

**target context x 100 = result**

**The HTML for a page with padding and borders**

**<body>**

**<header><p>This is the text for the header.</p></header> <main><p>This is the text for the main element.</p></main> <aside><p>This is the text for the aside.</p></aside> <footer><p>This is the text for the footer.</p></footer>**

**</body>**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **The CSS for a fluid layout**  **body {**  **width: 90%;**  **max-width: 1024px;**  **margin: 0 auto;**  **border: 2px solid black; }** | **/\*  /\*** | **changed from 960px \*/ maximum width of page \*/** | | | | |
| **header {** |  |  |  |  |  |  |
| **width: 96.875%;** | **/\*** | **930** | **960** | **x** | **100** | **\*/** |
| **padding: 15px 1.5625%;** | **/\*** | **15** | **960** | **x** | **100** | **\*/** |
| **border-bottom: 2px solid black;** |  |  |  |  |  |  |
| **main {** |  |  |  |  |  |  |
| **width: 59.375%;** | **/\*** | **570** | **960** | **x** | **100** | **\*/** |
| **padding: 15px 1.5625%;** | **/\*** | **15** | **960** | **x** | **100** | **\*/** |
| **float: left; }** |  |  |  |  |  |  |
| **aside {** |  |  |  |  |  |  |
| **width: 34.375%;** | **/\*** | **330** | **960** | **x** | **100** | **\*/** |
| **padding: 15px 1.5625%;** | **/\*** | **15** | **960** | **x** | **100** | **\*/** |
| **float: right; }** |  |  |  |  |  |  |
| **footer {** |  |  |  |  |  |  |
| **clear: both;** |  |  |  |  |  |  |
| **width: 96.875%;** | **/\*** | **930** | **960** | **x** | **100** | **\*/** |
| **padding: 15px 1.5625%;** | **/\*** | **15** | **960** | **x** | **100** | **\*/** |

**border-top: 2px solid black;** }

**Description**

* To convert the width of an element from pixels to a percent, you divide the width of that element (the target) by the width of its parent element (the context) and then multiply that value by 100.
* The width of the outermost element determines how much of the screen the page occupies. This width is typically set between 90% and 100%.
* In addition to converting the widths of the structural elements on a page to percents, you should convert the left and right margins and padding to percents. However, you can't specify the width of a border using a percent.
* To limit the width of an element, you can use the max-width property. This property is typically used to limit the width of the entire page.

**Figure 8-4 How to convert fixed widths to fluid widths**

**290** *Section 1 A crash course in HTML and CSS*

**How to size fonts**

In chapter 4 of this book, we recommended that you use relative measure­ments for your font sizes. When you do that, users can vary the sizes by using their browsers.

This is important for responsive design too. That way, the size of the font that's used by an element is relative to the size of the font used by the parent element. So if you want to change the font size that's used by a parent element and all of its child elements, you just need to change the font size for the parent element.

In this book, we've specified relative font sizes as percents, but you can also use ems. If you're working on a website that uses pixels instead, you can easily convert them to percents or ems. Figure 8-5 shows you how.

This figure starts by presenting the HTML for a simple web page. As you can see, this page has the same structure as the one shown in the previous figure, but it contains some additional content.

The CSS that follows shows how the font sizes for this page are calculated in ems. Before I explain how this works, notice that the base font size in the body element is specified as 100% rather than lem. Although both of these values result in the same font size, we recommend you use a percent for the base font size because an issue can arise in older versions of IE if you use ems.

To calculate a font size in ems, you simply divide the font-size in pixels by 16 since 16 is the default font size for most browsers. For example, the hl

element in this figure has been converted from 28 pixels to 1.75 ems, and the h2 element has been converted from 24 pixels to 1.5em. If you want to use percents instead of pixels, you can just multiply the result by 100.

**296** *Section 1 A crash course in HTML and CSS*

**How to use CSS3 media queries**

In the topics that follow, you'll learn how to use media queries to control the appearance of a page in various screen sizes. You'll also learn how to build responsive menus using a jQuery plugin called SlickNay. But first, you'll learn how to control the mobile viewport so it works in conjunction with media queries.

**How to control the mobile viewport**

When you develop a website that uses Responsive Web Design to provide for various screen sizes, you'll want to be sure it configures the *viewport* appro­priately for mobile devices. To do that, you can use the meta element that's presented in figure 8-8.

To start, you should know that the viewport on a mobile browser works differently from the viewport on a desktop browser. On a desktop browser, the viewport is the visible area of the web page. The user can change the size of the viewport by changing the size of the browser window.

In contrast, the viewport on a mobile device can be larger or smaller than the visible area. In this figure, for example, you can see that the first web page is displayed so the entire width of the page is visible. Some mobile browsers reduce a page like this automatically if no meta element is included. In contrast, other mobile browsers don't reduce the page at all, so it extends beyond the visible area of the screen as shown in the second web page in this figure.

When you use media queries, you want to be sure that the web page isn't reduced, since the media queries will adjust the appearance of the page based on the screen size. In other words, you want the page to look like the second one in this figure. To do that, you add a meta element like the one shown in this figure.

Here, the name attribute is set to "viewport" to indicate that the element applies to the viewport. Then, the content attribute specifies the properties for the viewport. The first property, width, indicates that the width of the viewport should be set to the width of the screen in CSS pixels at a zoom factor of 100%. This makes sense if you realize that a CSS pixel isn't always the same as a device pixel. On mobile devices that have high resolutions, for example, one CSS pixel can be equal to two or more device pixels. In addition, if a user zooms into a page, the number of CSS pixels per device pixel increases. At a zoom factor of 200%, for example, each CSS pixel is two device pixels wide and high.

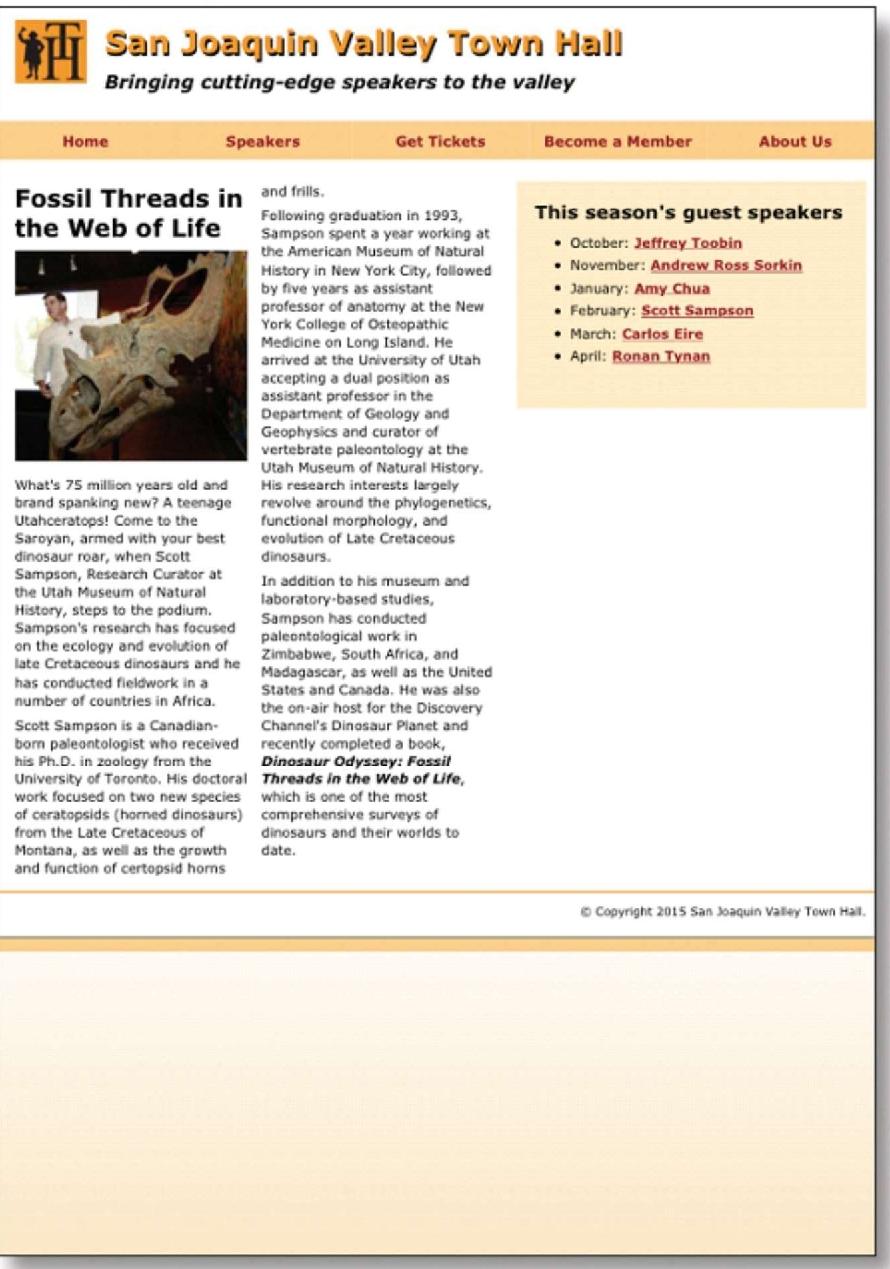
The second property, initial-scale, determines the initial zoom factor, or *scale,* for the viewport. In this case, this scale is set to 1, which represents the default width for the viewport. This is what keeps the browser from scaling the page automatically.

In addition to the width and initial-scale properties, you may want to set some of the other properties that are presented in this figure. Specifically, when you use RWD, you may want to set the user-scalable property to "no" so the user can't zoom in or out of the display. Or, if you want to let the user zoom in or

out, you can set the minimum-scale and maximum-scale properties to limit how much the user can zoom.

*Chapter 8 How to use Responsive Web Design* **297**

**A web page on a mobile device without and with a meta viewport element**



Moms

Speekere

Betrickets

Become a member

cljg Szin luzitjuin Hain

*Bringing cutting-edge speakers to the valley*

Research 0:aser

an m of raMeral s.

emes ftew to ...MM.' Samos. Ms conducted   
Sampim's niW. has focused

17:e fr.b■Vr="[IndlNeed

was1115.

1i

ecreere

***Tar\* :11r. VI:of We,***

MN,

This season's guest speakers

,McPaCsoloefim

* Apra: Ronsm\_Teneo



***Slain JOIMPElL***

***Bringing cutting-***

**Home**

**Speak(**

**Fossil Threads in the Web of Life**

anc

Fol Sar the His by pro Yor Me

a rr   
a cc   
ass

**Content properties for viewport metadata**

Figure 8-8 How to control the mobile viewport

**A meta element that sets viewport properties**

**Property Description**

The logical width of the viewport specified in pixels. You can also use the device-width keyword to indicate that the viewport should be the width of the screen in CSS pixels at a scale of 100%.

The logical height of the viewport specified in pixels. You can also use the device-height keyword to indicate that the viewport should be the height of the screen in CSS pixels at a scale of 100%.

A number that indicates the initial zoom factor that's used to display the page. A number that indicates the minimum zoom factor for the page.

A number that indicates the maximum zoom factor for the page.

Indicates whether the user can zoom in and out of the viewport. Possible values are yes and no.

**width**

**height**

**initial-scale minimum-scale maximum-scale user-scalable**

**<meta name="viewport" content="width=device-width, initial-scale=l">**

**Description**

* The *viewport* on a mobile device determines the content that's displayed on the   
  screen. It can be larger or smaller than the actual visible area of the screen.
* You use a meta element to control the viewport settings for a device. You add this element within the head element of a page.
* When you use media queries, you should set the width property of the content attribute to "device-width", and you should set the initial-scale property to 1. You can also prevent or limit scaling with the user-scalable, minimum-scale, and maximum-scale properties.

**298** *Section 1 A crash course in HTML and CSS*

**How to code media queries**

*Media queries* are a CSS3 feature that allows you to write conditional expressions directly within your CSS code. These conditional expressions can be used to query various properties of a device, such as the screen size. Figure 8-9 shows how to code media queries for a responsive design.

At the top of this figure, you can see the basic syntax of a media query. It starts with an *@media selector,* followed by a media type. When you're devel­oping a responsive design, you'll set the media type to "screen" so the media query will only be used if the page is displayed on a screen. You can also precede the media type with the only keyword. Then, if a browser doesn't support media queries, it will only check the media type.

After the media type, you code one or more conditional expressions, where each expression specifies the value of a property. The table in this figure lists some of the common properties for the screen media type. Note that all of these properties can also be prefixed with min- or max-, as shown in the media queries in this figure.

In the first media query, for example, the conditional expression uses the max-width property to check that the width of the viewport is 767 pixels or less. The second media query in this figure includes two conditional expressions. The first one checks that the width of the viewport is 480 pixels or more, and the second one checks that the width of the viewport is 767 pixels or less.

Within each media query, you code the CSS that adjusts the appearance of the web page so it's appropriate for the screen size that's specified by the query. For example, I could use a media query to change the font sizes that are used on a page or to change the layout of a page. You'll see the CSS for a web page that uses media queries like this later in this chapter.

This figure also lists all the desktop browsers that support media queries. This includes all the current versions of all the major browsers. Because of that, you can use any of these browsers to test that the media queries used by a web page work correctly. To do that, you just change the width of the browser window to the width specified by a media query. Then, the styles defined by that media query will be applied. Even if you do that, though, keep in mind that you'll still want to thoroughly test your media queries by using one of the techniques that were presented earlier in this chapter.

Before I go on, you should realize that all the desktop browsers except   
for Internet Explorer are updated automatically when a new version becomes

available. Because of that, you don't need to check what version you have before testing media queries. If you want to use Internet Explorer for testing, though, you need to be sure that you have at least version 9.

*Chapter 8 How to use Responsive Web Design* **299**

Figure 8-9 How to code media queries

**The basic syntax of a media query**

**@media [only]** media-type **[and** (expression-1)] **[and** (expression-2)]... { ...styles go here

}

**Common properties for the screen media type**

**Property Description**

|  |
| --- |
| **width** The width of the viewport.  **height** The height of the viewport.  **device-width** The width of the display area of the device. This is the same  as width if the viewport is set as shown in figure 8-8.  **device-height** The height of the display area of the device.  **orientation** Landscape or portrait. |

**A media query that checks that the viewport width is 767 pixels or less**

**@media only screen and (max-width: 767px) {...}**

**A media query that checks that the viewport width is between 480 and 767 pixels**

**@media only screen and (min-width: 480px) and (max-width: 767px) {...}**

**Desktop browsers that support media queries**

* Internet Explorer 9 and later
* Firefox 3.6 and later
* Safari 4 and later
* Opera 10 and later
* Chrome 5 and later

**Description**

* A *media query* is defined by a CSS3 *@media selector.* This selector specifies the media type for the query and, for the screen media type, one or more conditional expressions. If all of the conditions are true, the styles within the media query are applied to the page.
* Each conditional expression can check one of the properties listed above. These properties can also be prefixed with min- or max-.
* The screen size at which a media query is used to change the appearance of a page can be referred to as a *breakpoint.*
* Media queries are supported by all the current versions of all the major desktop browsers. That makes media queries easy to test in those browsers.
* Media queries are also supported by all the current versions of all mobile browsers.
* If you include the only keyword, older browsers that don't support media queries will check the media type but not the conditional expressions.

**300** *Section 1 A crash course in HTML and CSS*

**Common media queries for a responsive design**

Figure 8-10 presents some common media queries that are used to imple­ment a responsive design. Before I describe these media queries, you should know that there are two standard techniques for developing the queries you need. First, you can start with the desktop design and then develop media queries that will be applied successively to the next smallest screen size. Second, you can start with the design for the smallest screen size and then develop media queries that will be applied successively to the next largest screen size. This design technique is often referred to as "mobile first design".

If you're developing a new website, the technique you choose may depend on personal preferences. If you're used to designing pages for the desktop, for example, you may want to do that first. But even if you do, you'll want to keep the mobile design in mind from the start. That will make it easier to add the media queries you need for smaller devices later on.

Another consideration is whether your website will be accessed primarily by users on mobile phones or by users on larger devices like tablets and computers. If it will be used primarily by mobile phone users, it makes sense to develop the mobile design first. Then, you can write the media queries for implementing the site on larger devices later.

If you're developing media queries using a desktop down design, your media queries should look something like those shown in the first example in this figure. These media queries check the max-width property, so they apply the styles they contain if a device has a width that's less than or equal to the value that's specified. Here, the first media query is for full-size tablets in portrait orientation (768 to 959 pixels), the second media query is for mobile phones

in landscape orientation (480 to 767 pixels), and the third media query is for mobile phones in portrait orientation (479 pixels or less).

In addition to these media queries, you could include a media query for the desktop and tablets in landscape orientation (960 pixels or larger), but these styles are typically coded outside the media queries. You could also include a media query for all mobile phones. Then, that query would contain styles that apply to any mobile phone, and the other two queries for phones would contain styles specific to phones in landscape or portrait orientation.

The second example in this figure shows the media queries that are commonly used for a mobile up design. These queries check for smaller screen widths first by using the min-width property. Because of that, the styles in these media queries are applied if a device has a width that's greater than or equal to the value that's specified. Just as in the first example, you could also code the styles for the smallest device in a media query, but that's not necessary.

It's important to keep in mind that the media queries must be coded in the sequence shown here. That way, the styles in each media query can override the styles in the previous media query if the condition on that media query is satis­fied. If you're using desktop down design, for example, and the screen width is between 480 and 767 pixels, the desktop styles will be applied first, followed by the tablet portrait styles, and then the mobile landscape styles. This will make more sense when you see the example at the end of this chapter.

*Chapter 8 How to use Responsive Web Design* **301**

**Figure 8-10 Common media queries for a responsive design**

**Common media queries for a desktop down design**

**/\* tablet landscape and desktop layout (960px or more) \*/ ...tablet landscape and desktop styles go here**

**/\* tablet portrait layout (768px to 959px) \*/ @media only screen and (max-width: 959px) { ...tablet portrait styles go here**

**/\* mobile landscape layout (480px to 767px) \*/ @media only screen and (max-width: 767px) { ...mobile landscape styles go here**

}

**/\* mobile portrait layout (479px or less) \*/ @media only screen and (max-width: 479px) { ...mobile portrait styles go here**

**Common media queries for a mobile up design**

**/\* mobile portrait layout (479px or less) \*/ ...mobile portrait styles go here**

**/\* mobile landscape layout (480px to 767px) \*/ @media only screen and (min-width: 480px) { ...mobile landscape styles go here**

**/\* tablet portrait layout (768px to 959px) \*/ @media only screen and (min-width: 768px) ...tablet portrait styles go here**

**/\* tablet landscape and desktop layout (960px or more) \*/ @media only screen and (min-width: 960px) {**

**...tablet landscape and desktop styles go here**

}

**Description**

* One standard technique for developing the media queries for a responsive design is to start with the queries for larger devices and then work your way down to smaller devices.
* Another standard technique for developing media queries is to develop the design for the smallest mobile devices first and then work your way up to larger devices.
* When you use the techniques shown above, each media query inherits the styles that precede it in the style sheet, including styles that are coded outside of media queries. Then, the media queries can override the inherited styles.
* You can also code the starting set of styles in a media query. For a desktop down design, you would code the styles in a media query that specifies a minimum width of 960 pixels. For a mobile up design, you would code the styles in a media query that specifies a maximum width of 479 pixels.

**302** *Section 1 A crash course in HTML and CSS*

**How to build responsive menus with the SlickNav plugin**

When you develop a responsive design, you need to provide for menus that can be accessed easily on small mobile devices. One way to do that is to use a jQuery plugin called SlickNay. Figure 8-11 shows how this plugin works.

To use the SlickNav plugin, you start by downloading its Zip file from <http://slicknay.com> and then unzipping this file. Next, you copy the slicknay.css and jquery.slicknay.min.js files to your website. Then, you include a link element for the .css file, which includes the styles for the plugin, and you include a script element for the .js file, which includes the JavaScript for the plugin.

In addition to the link and script elements for the SlickNav plugin, you must include a script element for the jQuery core library, since this library is used

by the SlickNav plugin. Note that this element must be coded before the script element for the plugin. Also note that you can retrieve the jQuery library from the Content Delivery Network (CDN) on the jQuery website by coding the script element as shown here. That way, you don't have to download this library.

Finally, you code a script element that calls the slicknav method within the jQuery ready event handler. To do that, you use a jQuery selector that refers

to the id of the standard navigation menu, followed by the method name. Note that this menu must be coded using an unordered list as shown in the previous chapter. You can also code one or more parameters for the slicknav method. Here, the prependTo parameter is used to add the mobile menu to the nav element that has an id of "mobile\_menu". Without this parameter, the menu would be added at the beginning of the body element. For a complete list of parameters, see the SlickNav website.

When you use the SlickNav plugin, you must provide a way to display the mobile menu on smaller screens and hide it on larger screens. To do that, you use media queries. If you're using a desktop down design, for example, you'll want to include a rule set in the desktop styles that hides the mobile menu by setting its display property to "none". Then, when you want to display this menu, you set the display property for this menu to "block". In the example in this figure, this is done by using an id selector.

In addition to hiding and displaying the mobile menu, you can change its appearance using classes that are defined in the slicknay.css file. In the next figure, for example, you'll see a mobile menu with a different background color than the one shown here. To find out what classes are available, just display the slicknay.css file.

To use the SlickNav menu, you simply click on the MENU button to display the first level of the menu. Then, you can click an item in the menu to display another page. Or, if a menu item has a submenu, you can click on it to display that submenu. To hide any menus that are displayed, just click the MENU button again.

*Chapter 8 How to use Responsive Web Design* **303**

**A multi-tier menu that uses SlickNav**



**Code for the SlickNav plugin in the head element**

**<link rel="stylesheet" href="styles/slicknay.css">**

**<script src="**[**https://code.jquery.com/jquery-2.1.3.min.js**](https://code.jquery.com/jquery-2.1.3.min.js)**"></script> <script src="js/jquery.slicknay.min.js"></script>**

**<script type="text/javascript"›**

**$(document).ready(function(){ $(1#nav\_menu1).slicknav({prependTo:"#mobile\_menu"}); ));**

**</script>**

**HTML for the SlickNav menu and the navigation menu**

**<nav id="mobile menu"></nav>**

**<nav id="nav menufl>**

**<ul>**

**<li><a href="index.html">Home</a></li>**

**</ul> </nav>**

**CSS for the SlickNav menu**

**/\* hide the mobile menu \*/ #mobile**\_**menu { display: none;** }

**/\* display the mobile menu \*/ #mobile\_menu { display: block; }**

**Description**

* To use the SlickNav plugin, download the Zip file from <http://slicknay.com>. Then, unzip this file and add the slicknay.css and jquery.slicknay.min.js files to your website.
* Before you can use the SlickNav plugin on a web page, you must include a link element for the slicknay.css file and a script element for the jquery.slicknay.min.js file. You must also include a script element for the jQuery core library, and this element must be coded before the script element for the jquery.slicknay.min.js file.
* To use the SlickNav plugin, add a script element with the jQuery code shown above that refers to the element that contains the unordered list for the menu and the element for the mobile menu. Then, use the CSS display property to hide and display the mobile menu depending on the size of the screen.

**Figure 8-11 How to build responsive menus with the SlickNav plugin**

**304** *Section 1 A crash course in HTML and CSS*

**A web page that uses Responsive Web**

**Design**

To complete this chapter, the next three figures present a web page that uses a responsive design. This is the same web page you saw in figure 8-7 that used a fluid layout, but this page also uses media queries.

**The design of the web page**

Figure 8-12 shows the design of this page on the desktop and in both landscape and portrait orientation on an iPhone 6. The design of this page in portrait orientation on a tablet is identical to the design for the desktop, except that some of the fonts are smaller.

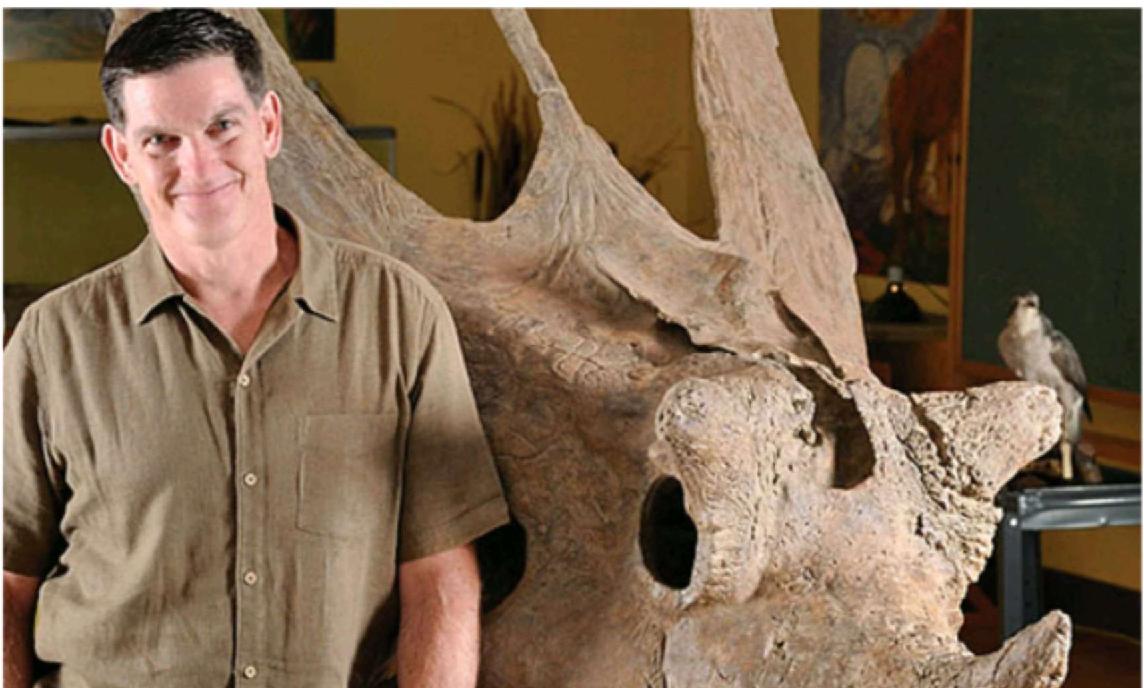
If you compare the appearance of this page in the desktop browser with its appearance in the iPhone in landscape orientation, you'll notice some major differences. To start, everything is smaller on the iPhone, which you would expect. Second, the margin and border have been removed from the page so it can use all of the available screen. Third, the logo is displayed above the text for the header and the logo and text are centered. If you scrolled down to the bottom of the page, you would see that the footer is centered too. Fourth, a SlickNav menu is displayed instead of a standard navigation menu. Fifth, the content is displayed in a single column with the article displayed before the sidebar. And sixth, the image in the article is floated to the left and it's sized so it takes up only a portion of the article width.

You'll also notice one main difference between the appearance of this page in landscape orientation and in portrait orientation on an iPhone. That is, the image in the article is changed back so it once again occupies the full width of the article.

*Chapter 8 How to use Responsive Web Design* **305**

**A speaker page in desktop and mobile layouts**

**111 San Joaquin** Valley Town x



**Fossil Threads in the Web of Life**



ATM aG 10 13 AM

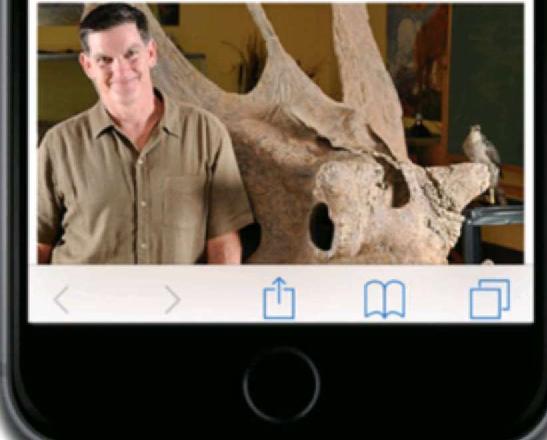
[murach.com](http://murach.com)

Lz.in juzgluir, Tr.wn

*Bringing cutting-edge speakers to the   
valley*

**MENU**

Fossil Threads in the Web of Life



C 127.0.0.1:8020/HTML5%20Book%20Apps/ch08/ch08\_media\_queries\_fluid/town\_hall/sampson.html

Figure 8-12 The design of the web page

|  |
| --- |
| **rim Salm VEllay Town r J** J J  ***Bringing cutting-edge speakers to the valley*** |

**Home Speakers Get Tickets Become a Member About Us**

**This season's guest speakers**

* October: Jeffrey -roobin
* November: **Andrew Ross Sorkin**
* **January: Amy Chua**
* **February: Scott Sampson**
* **March: Carlos Eire**
* **April: Ronan Tynan**

What's 75 million years old and brand spanking new? A teenage Utahceratops! Come to the Saroyan, armed with your best dinosaur roar, when Scott Sampson, Research Curator at the Utah Museum of Natural History, steps to the podium. Sampson's research has focused on the ecology and evol

fieldwork **in a** r*r*

sari jr.)D:JUifl IzdJ7z/Tr.pm, /11111

***Bringing cutting-edge speakers to the valley***

**Fossil Threads in the Web of Life**

What's 75 million years old and   
brand spanking new? A teenage

**Description**

* This web page uses a fluid layout and scalable images like the one in figure 8-7.
* This web page also uses media queries to change the appearance of the page depending on the size of the screen where it's displayed.
* The tablet portrait layout for this page is identical to the desktop and tablet landscape layout except that the font sizes are reduced.

**310** *Section 1 A crash course in HTML and CSS*

Finally, I added the media queries that change the appearance of the page for different screens sizes. For a tablet in portrait view, I simply reduced some of the fonts so the elements fit better on the screen. For a mobile phone in landscape view, though, I made a number of changes to the styles to implement the changes to the appearance of the page that you saw in figure 8-12.

First, I changed the display property of the navigation menu to "none" to hide it, and I changed the display property of the mobile menu to "block" to display it. This property is set to "none" in the styles outside the media queries so it's not displayed on larger screens. I also changed the background-color property of the SlickNav menu using the .slicknav\_menu class. Note that I included the !important rule for this property so it will override the property set in the slicknay.css file.

Next, I changed the styles for the body so it will fill the screen. To do that, I set the width to 100%, I removed the margins, and I removed the border.

To display the header as shown in figure 8-12, I removed the floating from the image so it's displayed above the two headings. I also aligned the text within the header so the image and headings are centered. Then, I reduced the font-sizes for the headings and I adjusted the margins. To change the layout of the footer, I centered and removed the right margin from the paragraph it contains.

*Chapter 8 How to use Responsive Web Design* **311**

**The CSS that controls the page layout Page 2**

/\* changes to the styles for the sidebar \*/ aside {

width: 35.35353%;

/\* 350 990 x 100 \*/

/\* 20 990 x 100 \*/

/\* 20 990 x 100 \*/

/\* 20 16 \*/

/\* base font size \*/

/\* 14 16 \*/

/\* full width of body \*/

/\* 12 16 \*/

/\* 20 990 x 100 \*/

float: left;

padding: 2.0202%;

background-color: #ffebc6; margin: 1.5em 2.0202% 0 0;

}

aside h2 { font-size: 1.25em; }

aside h3 { font-size: lem;

aside li { font-size: .875em; }

/\* changes to the styles for the footer \*/ footer { width: 100%; ... }

footer p {

font-size: .75em;

text-align: right;

margin-right: 2.0202%;

}

/\* hide the mobile menu initially \*/ #mobile\_menu {

display: none;

}

|  |  |
| --- | --- |
| **/\* tablet portrait to standard** 960 \*/  @media only screen and (max-width: 959px) {  #nav\_menu ul li a { font-size: .875em; section hl, article hl { font-size: 1.5em; } section h2, aside h2 { font-size: 1.125em;  ) | /\* 14 .4 16 \*/  /\* 24 16 \*/  /\* 18 16 \*/ |

/\* mobile landscape to tablet portrait \*/

@media only screen and (max-width: 767px) {

#nav\_menu { display: none; }

#mobile\_menu { display: block; }

.slicknav menu { background-color: #facd8a !important; } body {

width: 100%; /\* full width of screen \*/

margin: 0; /\* no margins \*/

border: none; /\* no border \*/

|  |  |
| --- | --- |
| }  header, footer p { text-align: center; } footer p { margin-right: 0; }  header h2 {  font-size: 1.625em;  margin: .4em 0 .25em 0; | /\* 26 16 \*/ |

}

header h3 {

font-size: lem; /\* base font size \*/

margin-left: 0;

}

header img { float: none;

Figure 8-14 The CSS for the web page (part 2 of 3)

**312** *Section 1 A crash course in HTML and CSS*

To display the content for the page in a single column, floating is removed from both the article and the aside. In addition, the margins and padding for these elements are adjusted so an equal amount of space is displayed at the left and right sides of the screen. Finally, the widths of these elements are set so they take up the rest of the screen. In other words, the widths of these elements are 100% minus the widths of any margins and padding for the elements.

The styles for the image within the article are also changed so the image floats to the left of the text. The width of this element is set to 50% of the width of the article, and the right margin is set to 2%. That means that the text for the article will take up the remaining 48%.

The media query for a phone in portrait view further reduces some of the fonts. In addition, it changes the image in the article so it takes up the full width of the article again. To do that, it removes the floating from the image, changes its width to 100%, and removes the right margin.

*Chapter 8 How to use Responsive Web Design* **313**

**The CSS that controls the page layout**

**Page 3**

article {

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 100 | - (4 | /\*  /\*  x | 100 - (2  20 4 990  2.0202) | x 2.0202) for padding  x 100 \*/  for padding and margins | \*/  \*/ |

width: 95.9596%;

float: none; margin-right: 2.0202%; padding-right: 0;

)

article img {

float: left; width: 50%; margin-right: 2%;

}

aside {

width: 91.9192%; /\*

float: none;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| margin: 0 2.0202% 2.0202% 2.0202%;  }  } | /\* | 20 | 990 x | 100 | \*/ |
| /\* mobile portrait to mobile landscape \*/ |  |  |  |  |  |
| @media only screen and (max-width: 479px) |  |  |  |  |  |
| header h2 { font-size: 1.375em; } | /\* | 22 4. | 16 \*/ |  |  |
| header h3 { font-size: .8125em; | /\* | 13 | 16 \*/ |  |  |
| p { font-size: .875em; | /\* | 14 | 16 \*/ |  |  |
| article hl, aside h2 |  |  |  |  |  |
| font-size: lem; } | /\* | base | font | size | \*/ |
| aside li { font-size: .875em; } | /\* | 14 | 16 \*/ |  |  |
| ) |  |  |  |  |  |
| article img { |  |  |  |  |  |

float: none;

width: 100%;

margin-right: 0;

)

aside h3 { font-size: .9375em; } /\* 15 16 \*/

footer p { font-size: .6875em; /\* 11 16 \*/

)

Figure 8-14 The CSS for the web page (part 3 of 3)

**314** *Section 1 A crash course in HTML and CSS*

**Perspective**

The use of mobile devices has increased dramatically over the past few years. Because of that, it has become important to design websites that are easy to use from these devices. Although that often means more work, this can be a critical aspect of maintaining your presence in the business world.

In this chapter, you learned how to develop websites using Responsive Web Design. With RWD, you use fluid layouts to adjust the width of a page depending on the screen size. You use media queries to adjust the appearance of a page depending on the screen size. And you use scalable images to adjust the size of the images depending on the screen size. With these features, you can develop responsive websites that have the same look-and-feel on desktops, tablets, and smart phones.

**Terms**

Responsive Web Design (RWD) scalable image

responsive design viewport

fluid design scale

fixed layout @media selector

fluid layout breakpoint

media query

**Summary**

* *Responsive Web Design* refers to a technique that's used to create websites that adapt gracefully to any screen size. A *responsive design* includes fluid layouts, media queries, and scalable images.
* To create a web page with a *fluid layout,* you set the widths of the page and its main structural elements to percents so they increase and decrease depending on the width of the screen.
* To convert the fixed width for an element to a fluid width, you divide the width of the element in pixels by the width of its containing element in pixels and then multiply the result by 100 to get a percent.
* When you develop a responsive design, you should specify font sizes in ems or percents. To convert a font size from pixels to ems, you divide the size by 16 since that's the default size for most browsers. To convert the font size to a percent, you multiply the result of the division by 100.
* To create a *scalable image,* you remove the height and width property from the img element for the image, and you set the max-width property to the percent of its containing block you want it to fill.
* If you want to limit the size of an image to its native size, you can set the width property to a percent and then set the max-width property to the native width in pixels.

*Chapter 8 How to use Responsive Web Design* **315**

* A *media query* is defined by a CSS3 *@media selector* that uses conditional expressions to determine when the styles it contains are applied. You use media queries with **RWD** to change the appearance of a page for different screen sizes.
* When you use media queries with **RWD,** you can develop the design for larger devices first and then work your way down to smaller devices. Or, you can develop the design for the smallest device first and then work your way up to larger devices.
* The *viewport* on a mobile device determines the content that's displayed on the screen. When you use media queries, the viewport should be set so the page is displayed at its full size.
* SlickNav is a jQuery plugin that converts a standard navigation menu to a menu that's easier to use on smaller devices.
* An easy way to test a responsive design is to use the developer tools that are provided by most modern browsers. You can also use device emulators and browsers simulators, or you can deploy the website and then test it on various devices or use a web-based tool like ProtoFluid.

|  |
| --- |
| **Exercise 8-1 Convert the Town Hall home page**  **to use a responsive design** |

In this exercise, you'll convert the Town Hall home page that you worked on in exercise 7-1 so it uses a fluid layout, scalable images, and media queries. When you're through, the page should look like this in mobile phone portrait and landscape orientations:



**San Joaquin Valley Town Hall**

***Celebrating our 75th Year***

**MENU**

**Our Mission**

San Joaquin Valley Town Hall is a non-profit organization that is run by an all-volunteer board of directors. Our mission is to bring nationally and internationally renowned, thought-provoking speakers who inform, educate, and entertain our audience! As one or our members told us:

***"Each year 1 give a ticket package to*** *each of our family members. I think of*



**San Joaquin Valley Town   
Hall**

***Celebrating our 75th Year***

**MENU**

Our Mission

San Joaquin Valley Town Hall is a non-profit organization that is run by an