Modernizing Your CSS Skills

with examples and
hands-on exercises

WEBUCATOR

Version: 1.0.1

The Author

Brian Hoke

Brian Hoke is Principal of Bentley Hoke, a web consultancy in Syracuse, New York. The firm serves the professional services, education, government, nonprofit, and retail sectors with a variety of development, design, and marketing services. Core technologies for the firm include PHP and Wordpress, JavaScript and jQuery, Ruby on Rails, and HTML5/CSS3. Previously, Brian served as Director of Technology, Chair of the Computer and Information Science Department, and Dean of Students at Manlius Pebble Hill School, an independent day school in DeWitt, NY. Before that, Brian taught at Insitut auf dem Rosenberg, an international boarding school in St. Gallen, Switzerland. Brian holds degrees from Hamilton and Dartmouth colleges.

Class Files

Download the class files used in this manual at https://static.webucator.com/media/public/materials/classfiles/CSS301-1.0.1.zip.

Errata

Corrections to errors in the manual can be found at https://www.webucator.com/books/errata/.

Table of Contents

LESSON 1
The Power of CSS

Topics Covered

☑ How we can use CSS to render visual and experience aspects of web pages for which we previously turned to images, JavaScript, or other techniques.

☑ Some of the challenges we face using CSS given the current state of browser support.

☑ An overview of CSS Level 3 and CSS Level 4.

Introduction

CSS Level 3 and Level 4 offers us the ability to craft visual styles and user experiences previously unavailable to us with earlier version of cascading style sheets, and lets us avoid the use of static images, JavaScript, and other techniques for rendering fonts, colors, animations, and other aspects of the web experience.

1.1. The Power of CSS

❖ 1.1.1. Doing More with Less

CSS Level 3 - the third iteration of the Cascading Style Sheet standard - offers web designers and developers a broad range of powerful techniques. We can use CSS Level 3 to:

- Render text with nonstandard fonts.
- Animate elements on the page.
- Display background gradients.
- Selectively alter styles depending on properties (e.g., screen width) of the user's device.
- And lots more.

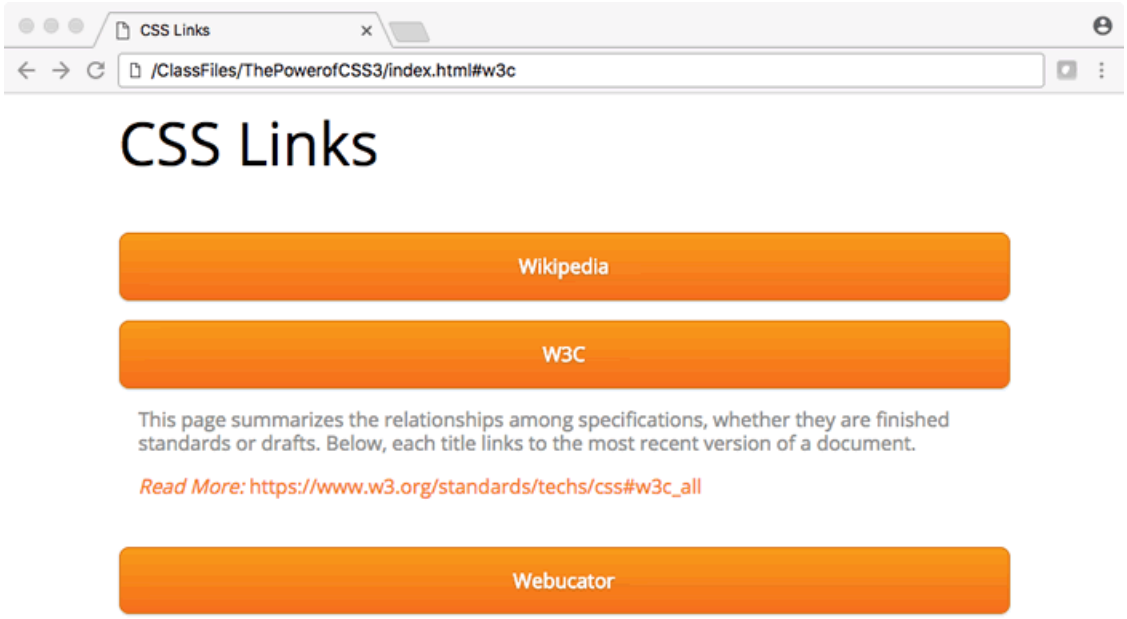
In the past, we would rely on images (to display text, say, in a nonstandard font, or to show a gradient background for an element), Flash or JavaScript (to animate an element on the

screen or respond to user events), or other techniques to enhance the look and behavior of the content we present to our users. Some of those techniques still have their place, but CSS Level 3 offers us a broader, more mature set of tools to bring to bear on the challenge of building the very best sites for our intended audiences.

Let's look at an example of a simple web page that manifests some of the power of CSS Level 3.

❖ 1.1.2. A First Example

Consider a page designed to present links to a few web resources on CSS - open `Class Files/the-power-of-css3/index.html` in a text editor to view the code and a browser to view the page. The page offers users button-type headers for each of the categories; clicking on the button expands the content below it (a snippet of text and a link) in accordion fashion. Here's a screenshot of the page, with the second ("W3C") element open:



The font for the title, buttons, and content is nonstandard; that is, it's not Arial, Georgia, or any of the other CSS Web Safe Fonts we can be sure most users will have on their systems. The buttons should fill the available width of the screen, with some padding on

either side. Clicking any of the three buttons should expand the content beneath and close up the non-clicked buttons' content. We'd like the page to render nicely - to stay true to both the look and the interaction specs - on any browser, of course, as well as on mobile devices like an Android phone or iPhone and on tablets like an iPad.

A few years ago, we might have gone about developing some elements on the page as follows:

- Use an image for the "CSS Links" title at top (to display the text in our nonstandard font).
- Use an image for the buttons, to display with rounded corners, gradient background, and drop shadow on the button text.
- Use JavaScript - the jQuery library, perhaps - to handle user events ("onclick") to selectively show and hide the content underneath each button.

Of course, these techniques come with some drawbacks:

- Using an image to display the "CSS Links" text is cumbersome - less easy to change as we update our page in the future - and won't be as readily accessible to search engine spiders, content syndication feeds, or users who employ assistive technologies (like screen readers).
- Apart from rendering all of the text as an image, we have no easy way to use a nonstandard font for the body text.
- Using an image for the button offers the same drawbacks as rendering text as an image - and makes it less easy for us to make the buttons fill the width of the allotted space in a responsive manner.
- JavaScript certainly works to fashion the accordion functionality - but it'd be nice if we didn't need to introduce another technology here, if possible.
- Sniffing the http request can tell us something about the user's device - and we can present a different style sheet in response - but this, too, is cumbersome: the "user-agent" string in the request can be difficult to decode, and how to handle future devices, that might not yet exist?

Unsurprisingly (this is a CSS course, after all!), we can accomplish all of our goals using CSS alone. While we'll spend lots of time later in the course on each of these areas (and many more) in detail, let's take a quick look now at some code to see how we can leverage some key CSS Level 3 capabilities.

First, let's look at the markup for our page. Open up `ClassFiles/the-power-of-css3/Demos/index.html` in a code editor:

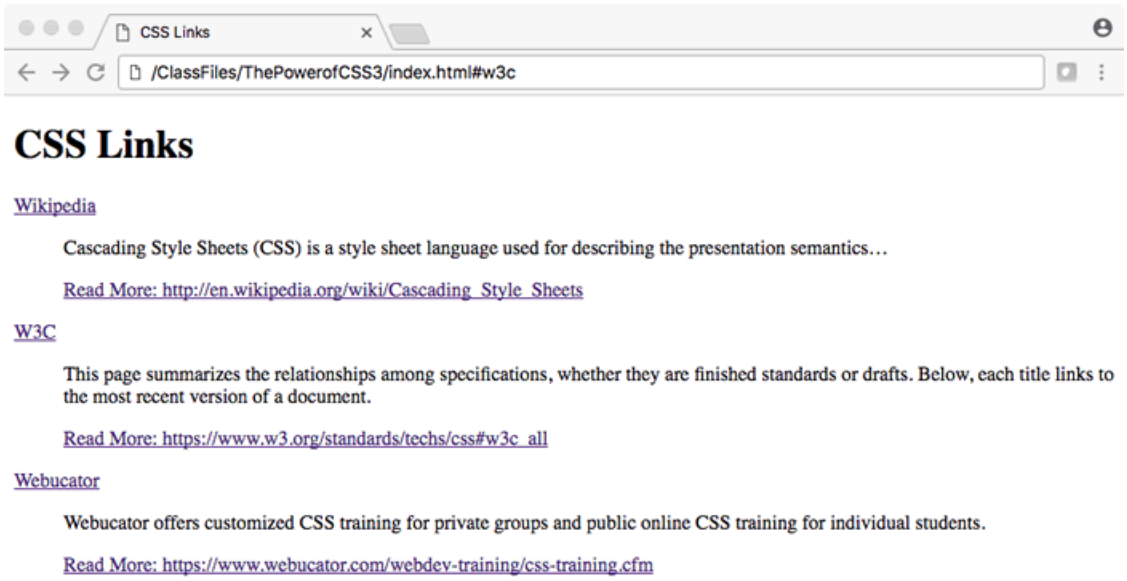
Demo 1.1: the-power-of-css3/Demos/index.html

```
1.    <!DOCTYPE html>
      -------Lines 2 through 10 Omitted-------
11.   <body>
12.    <div id="main">
13.     <h1>CSS Links</h1>
14.     <dl>
15.      <dt><a href="#wikipedia">Wikipedia</a></dt>
16.      <dd id="wikipedia">
17.       <p>Cascading Style Sheets (CSS) is a style sheet language used for describing
                the presentation semantics&hellip;</p>
18.       <p><a href="http://en.wikipedia.org/wiki/Cascading_Style_Sheets"><span>Read
                More:</span> http://en.wikipedia.org/wiki/Cascad ↵↵
               ing_Style_Sheets</a></p>
19.      </dd>
20.      <dt><a href="#w3c">W3C</a></dt>
21.      <dd id="w3c">
22.       <p>This page summarizes the relationships among specifications, whether they
                are finished standards or drafts. Below, each title links to the most
                recent version of a document.</p>
23.       <p><a href="https://www.w3.org/standards/techs/css#w3c_all"><span>Read
               More:</span> https://www.w3.org/standards/techs/css#w3c_all</a></p>
24.      </dd>
25.      <dt><a href="#webucator">Webucator</a></dt>
      -------Lines 26 through 29 Omitted-------
30.     </dl>
31.    </div>
32.   </body>
33.   </html>
```


Code Explanation

We're using a simple HTML5 shell - note that the first link is `<!DOCTYPE html>`. An `<h1>` tag is our title ("CSS Links"). An html definition list (`<dl>`) wraps the three links - each button-header is a term (`<dt>`) and each associated content (i.e., the content that shows/hides) is a description (`<dd>`).

Here's a view of the page before we apply any styles:



We'll now apply some CSS. First we link, via the `<link>` tag, some CSS to reset all styles to remove browser inconsistencies; we'll use `Eric Meyer's popular code (http://meyerweb.com/eric/tools/css/reset/)` here. Open up the file `ClassFiles/the-power-of-css3/Demos/reset.css` to view the CSS code.

Next, we'll apply styles to make the page look and behave as detailed above; open up `ClassFiles/the-power-of-css3/style.css` to examine the CSS. First, we can use `Google Web Fonts (http://www.google.com/webfonts)` to bring in a specific font (`Open Sans (http://www.google.com/webfonts/specimen/Open+Sans)`) with the line of code

```
<link href="http://fonts.googleapis.com/css?family=Open+Sans" rel="stylesheet">
```

in the `<head>` of `ClassFiles/the-power-of-css3/Demos/index.html`. We apply

```
font-family: 'Open Sans', sans-serif;
```

to the `<body>` (in `ClassFiles/the-power-of-css3/Demos/style.css`) to render all text in this font.

We'll now style the definition list, the core of our page. The `<dl>` tag gets a style of `width:100%` which, given that this element sits inside of the `<div>` with `id #main` and `div#main` is styled to have a `max-width` of 720 pixels, ensures that the definition list fills the center of the screen up to the maximum width allocated to it.

Each of the three `<dt>` elements - the button-headers - is styled as follows:

```
dl dt {
  text-align: center;
  padding: .5em 2em .55em;
  text-shadow: 0 1px 1px rgba(0,0,0,.3);

  /* rounded corners for the buttons */
  border-radius: .5em;

  /* slight dropshadow */
  box-shadow: 0 1px 2px rgba(0,0,0,.2);

  border: solid 1px #da7c0c;

  background: #f78d1d;
  /* CSS gradient background for browsers that support it */
  background: linear-gradient(#faa51a, #f47a20);

  margin-bottom: 1em;
}
```


We use the CSS Level 3 `text-shadow` and `box-shadow` properties to add drop shadowing to the text label and the buttons, respectively, and the CSS Level 3 `border-radius` property to round the corners of the buttons.

We apply a gradient background - with lighter orange on top and darker orange on bottom - with `linear-gradient`. Older browsers will see just a flat orange background with no gradient.

Next, a style for the definition - the initially hidden content for each item:

```
dl dd {
  color: #999;
  overflow: hidden;

  /* transition when height changes */
  transition: height 0.25s ease;
}
```

Here we apply a CSS Level 3 `transition`: the height (the change for which we'll handle a little farther down in the style sheet) should change over the course of 1/4 of a second (`0.25s`), rather than happening instantly. We specify the timing function ("`ease`") so that the transition starts slowly, speeds up, then slows near the end.

The following CSS allows browsers to break long strings of text with no spaces into multiple lines, rather than extending (as would be the case without these rules) past the right edge of the displayed area:

```
dl dd p a {
  color: #f78d1d;
  text-decoration: none;

  /* ensure that long strings (like URLs) without spaces will break over multiple lines,
 if needed */
  word-break: break-all;
  word-break: break-word;
  hyphens: auto;
}
```


This will be especially important when our page is viewed on a mobile device, with a narrower screen.

The next two CSS rules effect the open- or closed-ness of the content:

```
dl dd:not(:target) {
  height: 0;
}

dl dd:target {
  height: 7em;
}
```

CSS allows us to style an element referenced in a fragment on the end of the page's URL. Note that, when a user clicks on one of the links in a button on the page, the user visits a URL like `ClassFiles/the-power-of-css3/index.html#wikipedia`; we exploit the CSS Level 3 `:target` pseudo-selector to style the element corrresponding to the fragment (like "wikipedia") at the end. Those elements which don't correspond to the target (`:not(:target)`) get a height of 0; the element that does match the `:target`, if any, gets a height of `auto`.

The next CSS rule adjusts the page slightly when viewed on a device with a small screen:

```
@media only screen and (max-device-width: 480px) {
  dl dd:target {
    height: 10em;
  }
}
```

The font size for the "open" item was a little small when viewed on my phone; this *media query* says "make the font size `1.2em` when viewed on a device whose screen width is less than 480px".

Nothing up our sleeves here - no images or JavaScript were harmed (or used) in the creation of this page.

❖ 1.1.3. Challenges

Sadly, our solution won't work on all browsers - older versions of Internet Explorer prior to 9 won't display correctly, for instance. A key theme throughout this course will be when to (and how to) use CSS Level 3 (and Level 4) techniques appropriately:

- Is the desired effect something that *has* to work for all users, regardless of browser in use?
- If so, can we code the effect in such a manner that those with older/less-capable browsers will see an appropriate fall-back style or interaction? That is, can we ensure that our code degrades gracefully?
- Can we make assumptions about our audience that tells us whether, and when, to employ a given CSS Level 3 technique?

We'll review these concepts in detail as we move through the course. Of course, using these newer CSS features depends upon the target browser support for your website or application.

1.2. CSS Level 3 and CSS Level 4

❖ 1.2.1. CSS Level 3

CSS Level 3 (often referred to as "CSS3"), like earlier versions of CSS, allows for separation of content from presentation: well-formed, semantic markup (whether it be XHTML, HTML5, or even XML) connotes the meaning of the content: an `<h1>` implies a page title, while an HTML5 `<nav>` tag signifies a navigation element. With CSS, we define the content's visual presentation: the colors and fonts in which text is rendered, for example, or the manner in which various elements are positioned. We can present different styles for different media (like screen or print) and, with CSS Level 2, craft styles in response to the screen width or other properties of the user's device.

Unlike previous versions of CSS, CSS Level 3 has been released as a series of modules. Whereas, for example, the `CSS2 specification (http://www.w3.org/TR/CSS21/)` completely defines that version of CSS2, CSS3 is organized into separate modules. Progress on each module, representing either an extension to features from CSS2 or new capabilities, proceeds independently. The W3C's `CSS Current Status (https://www.w3.org/standards/techs/css)` lists the status of the various modules. Ultimately, support among popular browsers for each module dictates the degree to which you can safely use a given module and expect that visitors to your pages will see the style you intend.

CSS Level 3 includes the following modules:

Selected CSS Level 3 Modules

Module	Description
W3C Link	
Selectors Level 3	Selectors are patterns that match against elements in a tree, and as such form one of several technologies that can be used to select nodes in an XML document.
`Selectors Level 3 (https://www.w3.org/TR/css3-selectors/)`	
CSS Backgrounds and Borders Module Level 3	Borders consisting of images, boxes with multiple backgrounds, boxes with rounded corners and boxes with shadows
`CSS Backgrounds and Borders Module Level 3 (https://www.w3.org/TR/css-backgrounds-3/)`	
CSS Color Module Level 3	Color-related properties and values to color the text, backgrounds, borders, and other parts of elements in a document; color values and properties for foreground color and group opacity.
`CSS Color Module Level 3 (https://www.w3.org/TR/css-color-3/)`	
CSS Fonts Module Level 3	Font properties and how font resources are loaded dynamically.
`CSS Fonts Module Level 3 (https://www.w3.org/TR/css-fonts-3/)`	
Media Queries	A media query consists of a media type and zero or more expressions that check for the conditions of particular media features. Among the media features that can be used in media queries are `width`, `height`, and `color`. With media queries, presentations can be tailored to a specific range of output devices without changing the content itself.
`Media Queries (https://www.w3.org/TR/css3-mediaqueries/)`	
CSS Multi-column Layout Module Level 1	Multi-column layouts: content can be flowed into multiple columns with a gap and a rule between them.
`CSS Multi-column Layout Module Level 1 (https://www.w3.org/TR/css-multicol-1/)`	

We will cover each of these modules in more detail later in this class.

1.3. CSS Level 4

CSS Level 4 has no single specification; like CSS Level 3, CSS Level 4 is a collection of "level 4" modules, each of which is developed (and, potentially, adopted as a standard) on its own timeline. These modules are very much, as of this writing, in the development stage. Some of the more intriguing modules include:

❖ 1.3.1. Image Values

This module enables the use of the `image()` function in CSS, allowing us to construct a style like the following:

```
div.darkbackground {
  color: #fff;
  background: image("black.png", #000);
}
```


In which the color `#000` (black) is used as background if the image does not load.

CSS already has a mechanism allowing for multiple backgrounds:

```
div.darkbackground {
  color: #fff;
  background: url("black.png") #000;
}
```

However, if the image specific by the `url()` function *does* load and had some transparency in it, then the background color (`#000`) would still show, behind the transparency. The CSS `image()` function would allow us to include a fallback background color if and only if the specified background image did not load.

Learn more: `W3C CSS Image Values and Replaced Content Module Level 4 Specification (https://www.w3.org/TR/css-images-4/)`

❖ 1.3.2. Background and Borders

The "CSS Backgrounds and Borders Module Level 4" is a draft; as the W3C's CSS Working Group `page on the module (https://drafts.csswg.org/css-backgrounds-4/)` states, "This spec is not yet ready for implementation. It exists in this repository to record the ideas and promote discussion." The draft includes proposed extensions of how we can render borders and backgrounds in CSS. For instance, a new `corner-shape` property would allow us to specify a variety of shapes for corners of elements defined via the (existing) `border-radius` property. A style such as

```
div {
  border-radius: 5px;
  corner-shape: notch;
}
```

with a `corner-shape` of `notch` would style an element where the border radii define a concave rectangular notch at the corner. In addition to `notch`, other values would be `round`, `bevel`, and `scoop`.

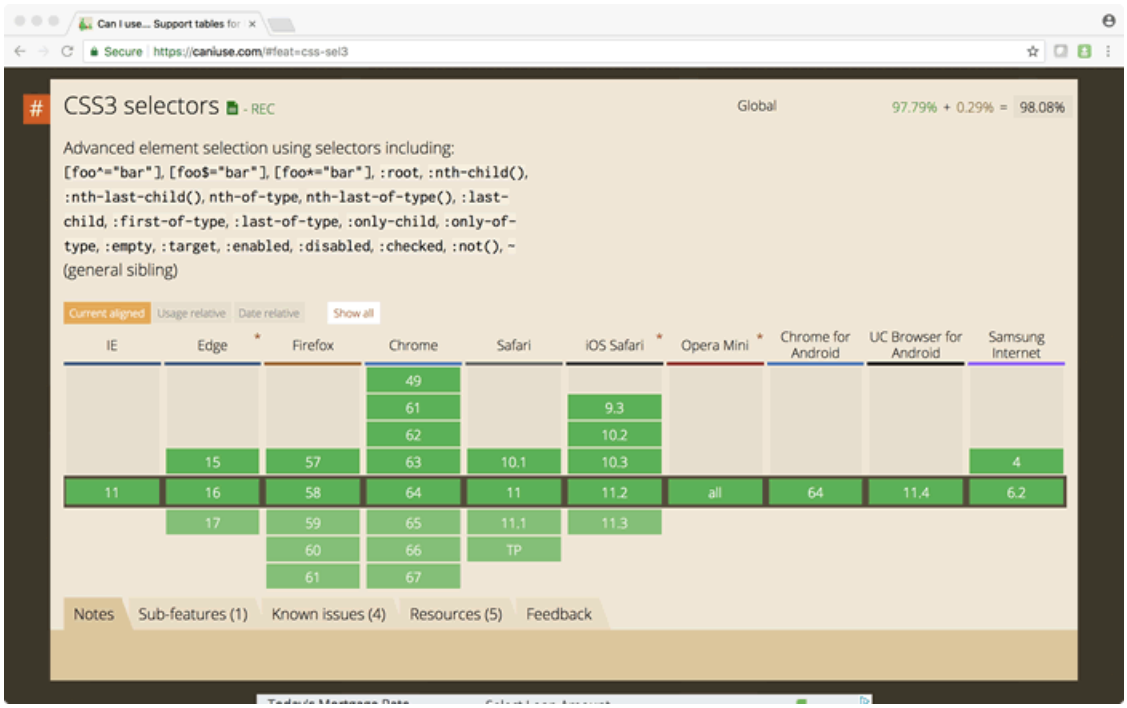
As mentioned above, this module - and other CSS Level 4 modules - are very much in the draft stage; browser support will be limited.

Learn more: `CSS Backgrounds and Borders Module Level 4 (https://drafts.csswg.org/css-backgrounds-4/)`

❖ 1.3.3. Browser Support

Various browsers offer differing levels of support for different CSS Level 3 and CSS Level 4 modules. Older version of Internet Explorer tend to provide spotty support for many of the newer modules; more recent versions of browsers (including both IE and many mobile browsers) offer better support.

Of course, careful attention to the intended audience for your sites, to the current state of browser support for CSS features you might use, and to fallback strategies for older browsers ensures that the most users get the most meaningful experience from the pages you build. Online resources such as `CanIUse.com (http://caniuse.com/)` offer a helpful list of feature support, for CSS as well as other technologies such as HTML5. For instance, `http://caniuse.com/#feat=css-sel3` lists how well (or not) browsers such as Internet Explorer, Firefox, Chrome, Safari, Opera, and others support the use of CSS selectors such as `:nth-child`, `:first-of-type`, etc.:



Conclusion

In this lesson, you have learned:

- How CSS offers designers and developers the ability to present complex styles and user experiences.
- How CSS enables us to skip the use of images, JavaScript, and other techniques.
- An overview of CSS Level 3 and CSS Level 4.

LESSON 2

Selectors and Pseudo Classes

Topics Covered

☑ How to use CSS wildcard selectors.

☑ How to use CSS `::before` and `::after` selectors.

☑ How to use CSS pseudo-classes.

☑ How to use CSS pseudo-elements.

Introduction

CSS Level 3 introduced new selectors, new pseudo-classes, and new pseudo-elements, giving web designers and developers more powerful ways in which to target elements for styling.

2.1. Selectors, Pseudo-Classes, and Pseudo-Elements

❖ 2.1.1. Attribute Selectors

CSS Level 3 introduces four new attribute selectors, allowing us to more precisely target elements whose attributes begin, end, or contain a specified string:

New Selectors in CSS3

Selector	Example(s)	Notes
`[attr^=val]`	`img[alt^="abc"]`	Every <img> whose alt attribute value starts with “abc”
`[attr$=val]`	`img[alt$="xyz"]`	Every <img> whose alt attribute value ends with “xyz”
`[attr*=val]`	`img[alt*="def"]`	Every <img> whose src attribute value contains string “def”

These selectors can be useful if the markup to which we are applying styles do not have convenient ids or classes, or if some aspect of the code lends itself to querying against a

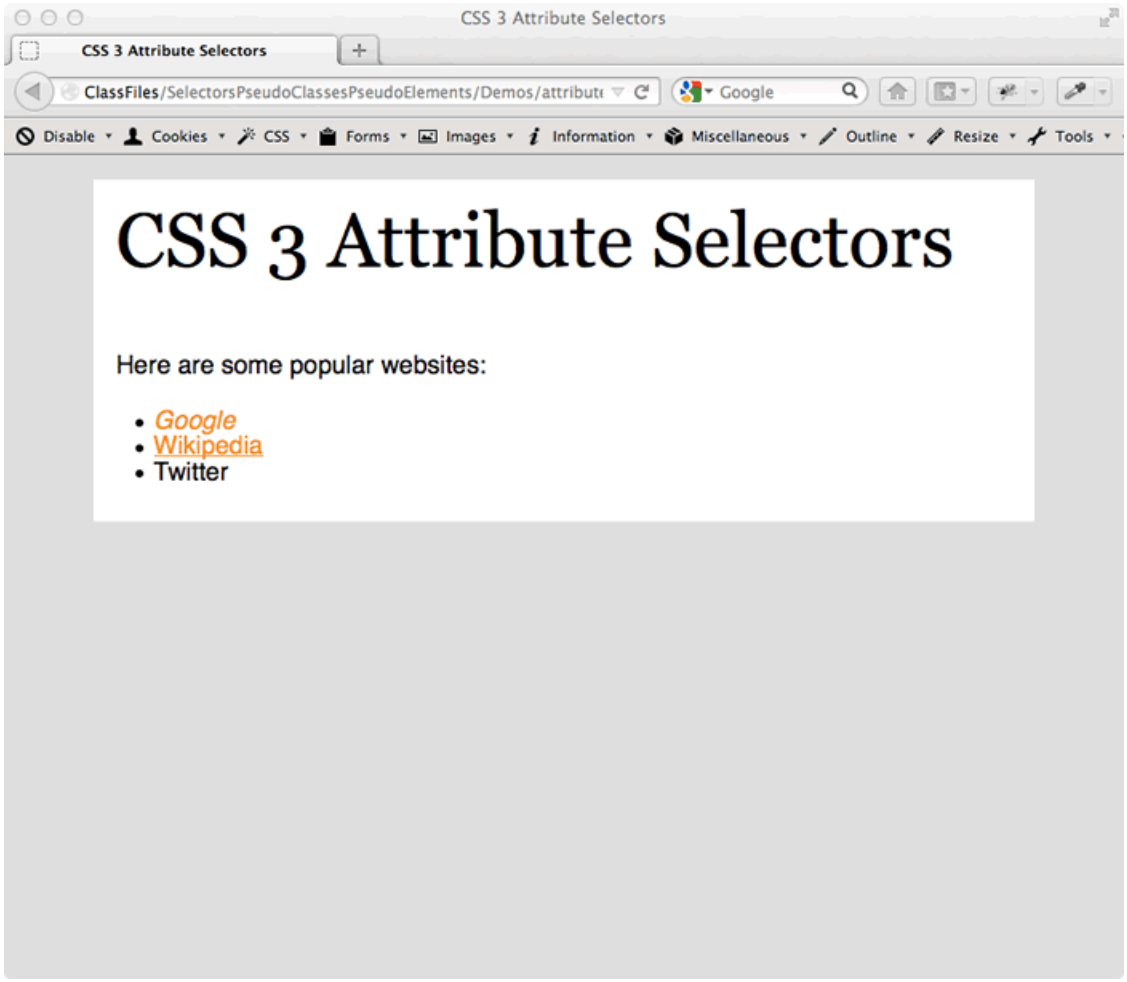
substring of an attribute value. Let's look at an example in which we target the `href` attribute of some simple links. Open `ClassFiles/selectors-pseudoclasses/Demos/attributeselectors.html` in a browser and in a file editor to check the code.

Demo 2.1: selectors-pseudoclasses/Demos/attributeselectors.html

```
1.  <!DOCTYPE html>
2.  <html>
3.  <head>
4.   <meta charset="utf-8">
5.   <title>CSS3 Attribute Selectors</title>
6.   <meta name="viewport" content="width=device-width">
7.   <link rel="stylesheet" href="../Shared/reset.css">
8.   <link rel="stylesheet" href="../Shared/style.css">
9.   <style>
10.   a[href^="https"] {
11.    font-style: italic;
12.   }
13.   a[href$="/"] {
14.    color:#000;
15.   }
16.   a[href*="wiki"] {
17.    text-decoration: underline;
18.   }
19.  </style>
20. </head>
21. <body>
22.  <div id="main">
23.   <h1>CSS3 Attribute Selectors</h1>
24.   <p>Here are some popular websites:</p>
25.   <ul>
26.    <li><a href="https://www.google.com">Google</a></li>
27.    <li><a href="http://en.wikipedia.org/wiki/Main_Page">Wikipedia</a></li>
28.    <li><a href="http://twitter.com/">Twitter</a></li>
29.   </ul>
30.  </div>
31. </body>
32. </html>
```


Code Explanation

The code above renders the following:



We include a reset style sheet (`reset.css`) and a general style sheet (`style.css`) to render some general styles for the page. We include the specific styles for this page in the `<head>` of the page - we'll do this throughout the course, to make easier scanning the relevant styles, unless the demo requires many lines of CSS code.

The first CSS rule (`a[href^="https"]`) targets any link whose `href` attribute starts with "https"; the Google link is thus italicized.

The second CSS rule (`a[href$="/"]`) targets any link whose `href` attribute ends with a forward slash. The effect of this rule is color the Twitter link in black.

The third rule (`a[href*="wiki"]`) targets any link with an `href` attribute containing (anywhere in the string) the text "wiki". This rule catches the Wikipedia link, which is thus underlined.

We can also use attribute selectors to target class or id names, regardless of the element to which they are applied:

```
[class*="ew"] {
        font-weight:bold;
      }
```

This would make bold the text within elements such as `<div class="new">`, `<p class="news">`, etc.

Exercise 1: Styling a Document Using Attribute Selectors

10 to 15 minutes

In this exercise, you will use CSS3 attribute selectors to style text.

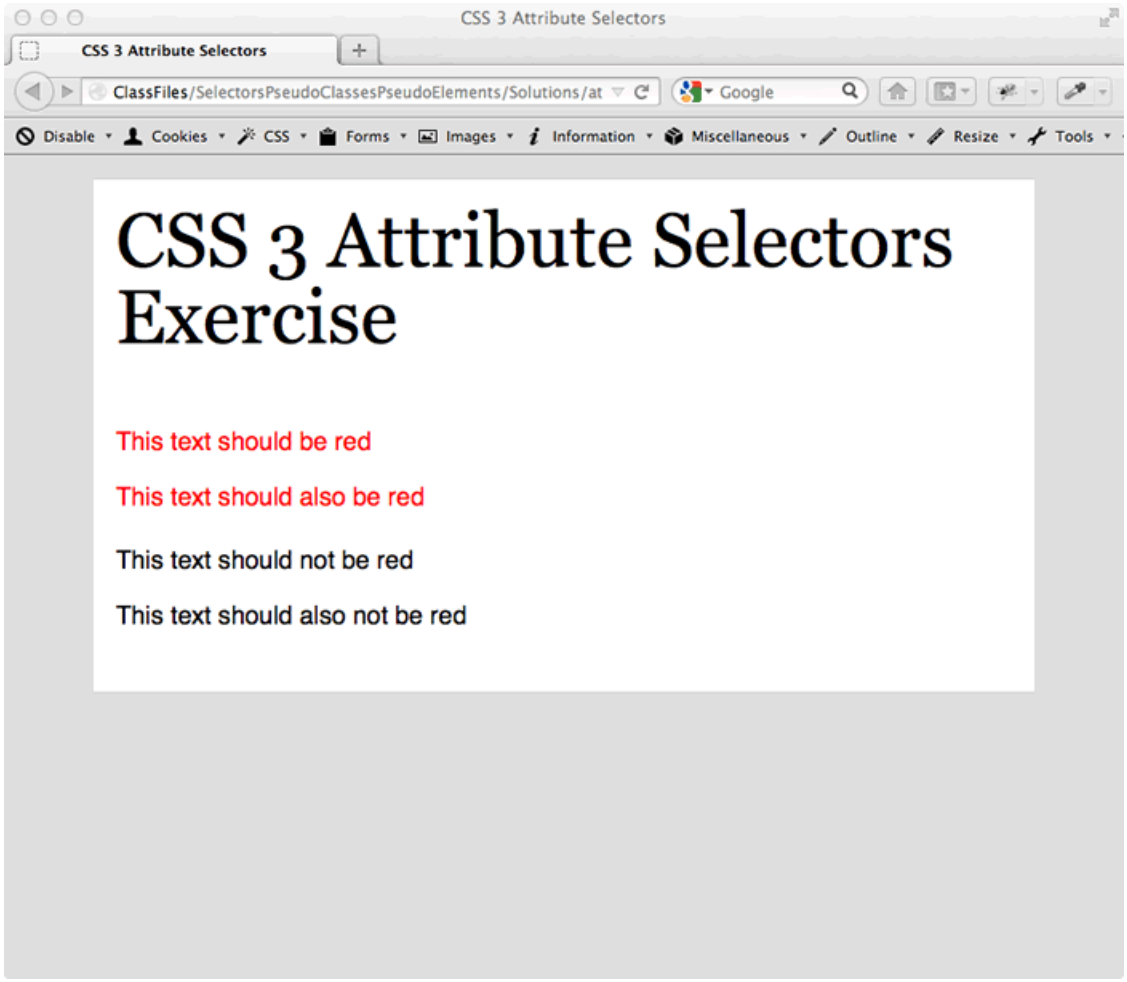
1. Open `ClassFiles/selectors-pseudoclasses/Exercises/attributeselectors.html` in a text editor.
2. Add a CSS rule where indicated (inside the `<style></style>` tags) to color the text red inside of any element with a class name containing the string “special”.
3. Note that there are four elements: the paragraph of class “special” and `div` of class “especial” should be red; the other paragraph and `div` should not be red.
4. Check your work in a browser.

Solution: selectors-pseudoclasses/Solutions/attributeselectors.html

```
1.   <!DOCTYPE html>
2.   <html>
3.   <head>
4.    <meta charset="utf-8">
5.    <title>CSS3 Attribute Selectors</title>
6.    <meta name="viewport" content="width=device-width">
7.    <link rel="stylesheet" href="../Shared/reset.css">
8.    <link rel="stylesheet" href="../Shared/style.css">
9.    <style>
10.    [class*="special"] {
11.     color:red;
12.    }
13.   </style>
14.  </head>
15.  <body>
16.   <div id="main">
17.    <h1>CSS3 Attribute Selectors Exercise</h1>
18.    <p class="special">This text should be red</p>
19.    <div class="especial">This text should also be red</div>
20.    <p class="normal">This text should not be red</p>
21.    <div>This text should also not be red</div>
22.   </div>
23.  </body>
24.  </html>
```


Code Explanation

The solution should look as follows:



We target class names containing “special” with the selector `[class*="special"]`.

Exercise 2: Using CSS Target

15 to 20 minutes

In this exercise, you will use the CSS `target` pseudo-class to style elements on a page in response to the target fragment on the page's URI.

1. Open `ClassFiles/selectors-pseudoclasses/Exercises/targetselectors.html` in a text editor.
2. The page contents and style display a simple navigation at top to allow users to jump lower on the page to the desired item; note that the navigation is marked up as an unordered list of links inside an HTML5 `<nav>` element and each item below is an `<article>`.
3. Add CSS rules where indicated (inside the `<style></style>` tags) to highlight the selected item - that is, the item corresponding to the target on the URL, after the user clicks on a link.
4. Change the border color, background color, and link color for the highlighted item.
5. Check your work in a browser.

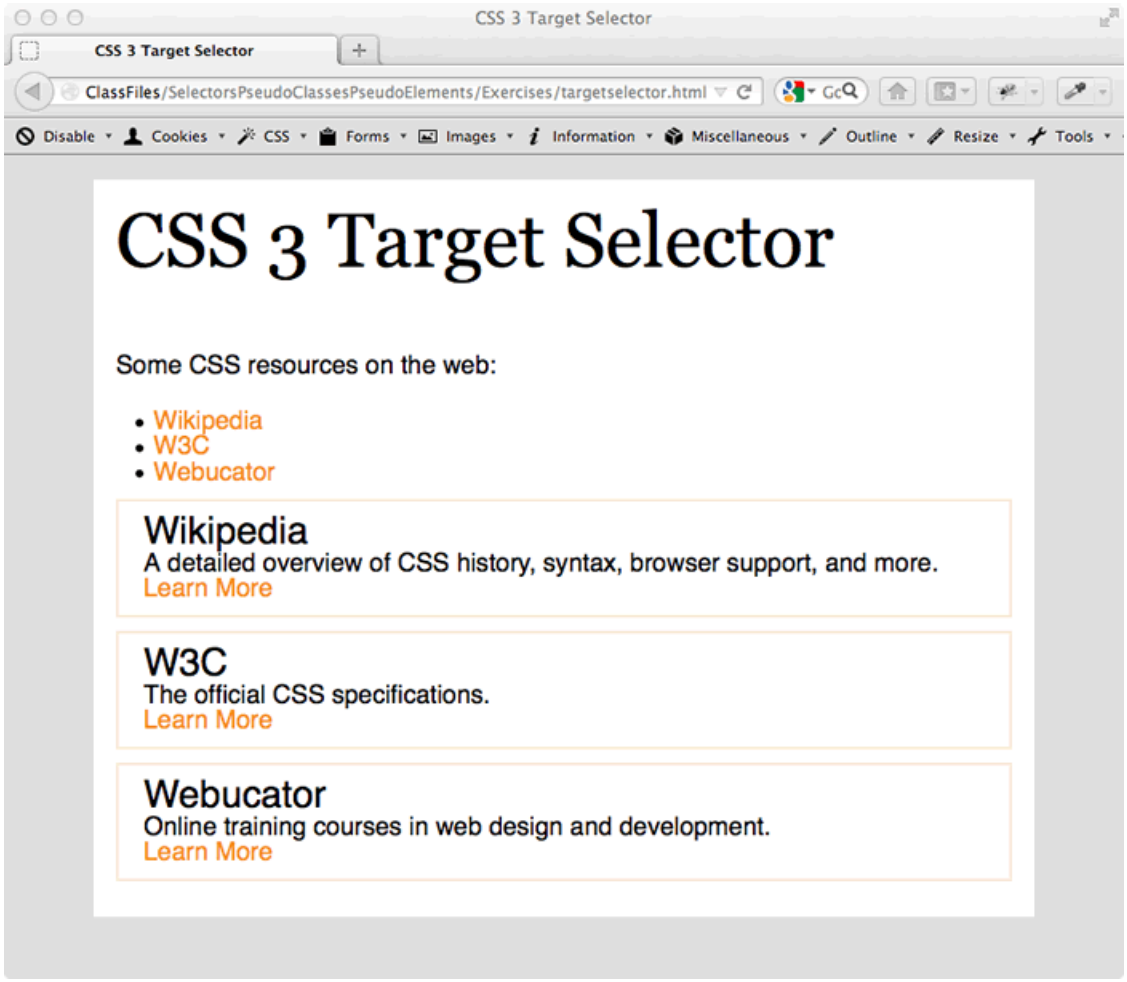
Solution: selectors-pseudoclasses/Solutions/targetselector.html

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
 <title>CSS3 Target Selector</title>
 <meta name="viewport" content="width=device-width">
 <link rel="stylesheet" href="../Shared/reset.css">
 <link rel="stylesheet" href="../Shared/style.css">
 <style>
  article {
   padding: 0.5em 1em;
   margin: 0.5em 0;
   border: 2px solid AntiqueWhite;
  }
  article h2 {
   font-size:1.5em;
  }
  article p {
   margin:0;
  }
  article:target {
   color: #fff;
   background:DarkGreen;
   border: 2px solid #000;
  }
  article:target a {
   text-decoration: underline;
  }
 </style>
</head>
<body>
 <div id="main">
  <h1>CSS3 Target Selector</h1>
  <p>Some CSS resources on the web:</p>
  <nav>
   <ul>
    <li><a href="#wiki">Wikipedia</a></li>
    <li><a href="#w3c">W3C</a></li>
    <li><a href="#webucator">Webucator</a></li>
   </ul>
  </nav>
  <article id="wiki">
   <h2>Wikipedia</h2>
   <p>A detailed overview of CSS history, syntax, browser support, and more.</p>
```

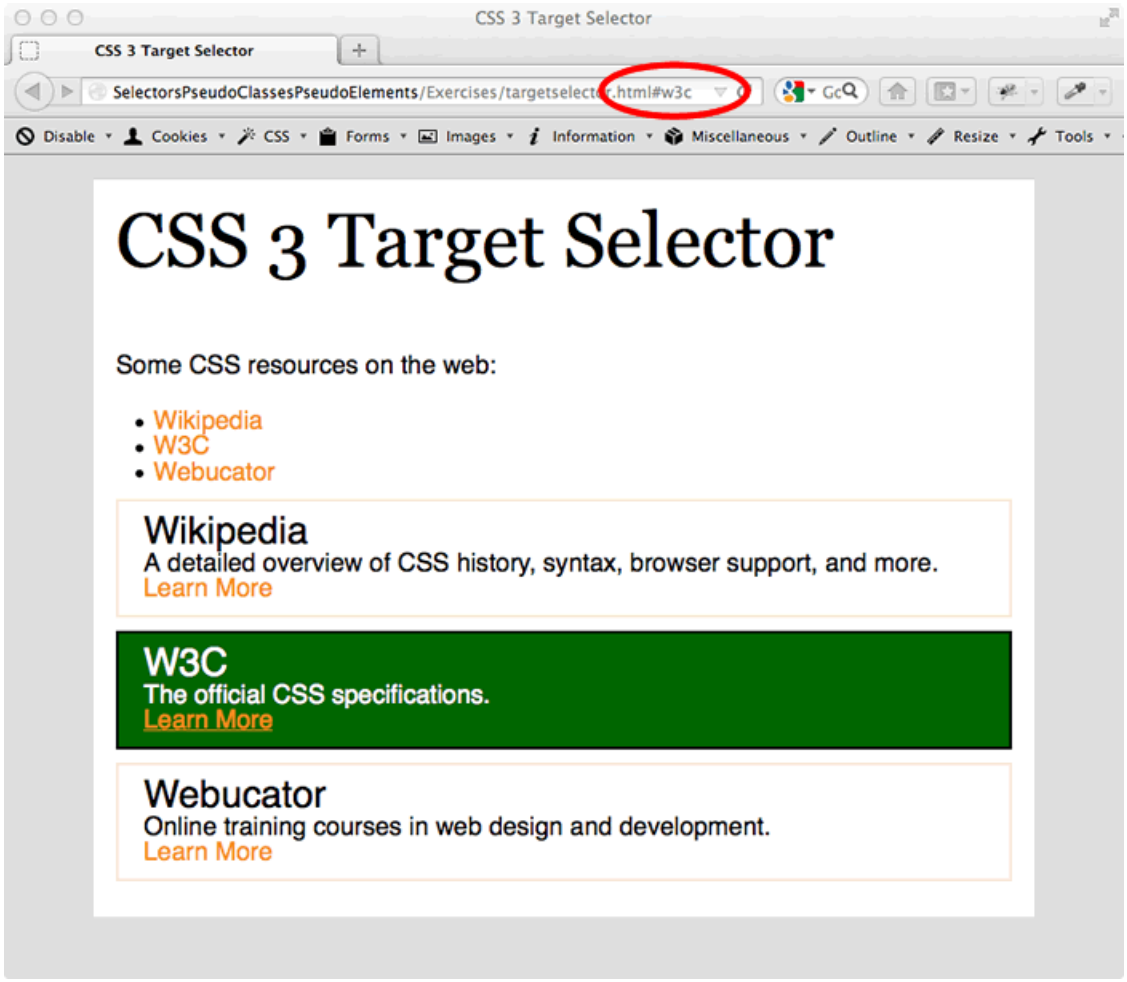
```
45.      <p><a href="http://en.wikipedia.org/wiki/Cascading_Style_Sheets">Learn
               More</a></p>
46.     </article>
47.     <article id="w3c">
48.      <h2>W3C</h2>
49.      <p>The official CSS specifications.</p>
50.      <p><a href="http://www.w3.org/Style/CSS/Overview.en.html">Learn More</a></p>
51.     </article>
52.     <article id="webucator">
53.      <h2>Webucator</h2>
54.      <p>Online training courses in web design and development.</p>
55.      <p><a href="http://www.webucator.com/webdesign/css.cfm">Learn More</a></p>
56.     </article>
57.    </div>
58.   </body>
59.   </html>
```


Code Explanation

Before adding styles to address the highlighting - or before the user clicks on a link - looks as such:



If the user were to click the second (“W3C”) link, the page should look like this:



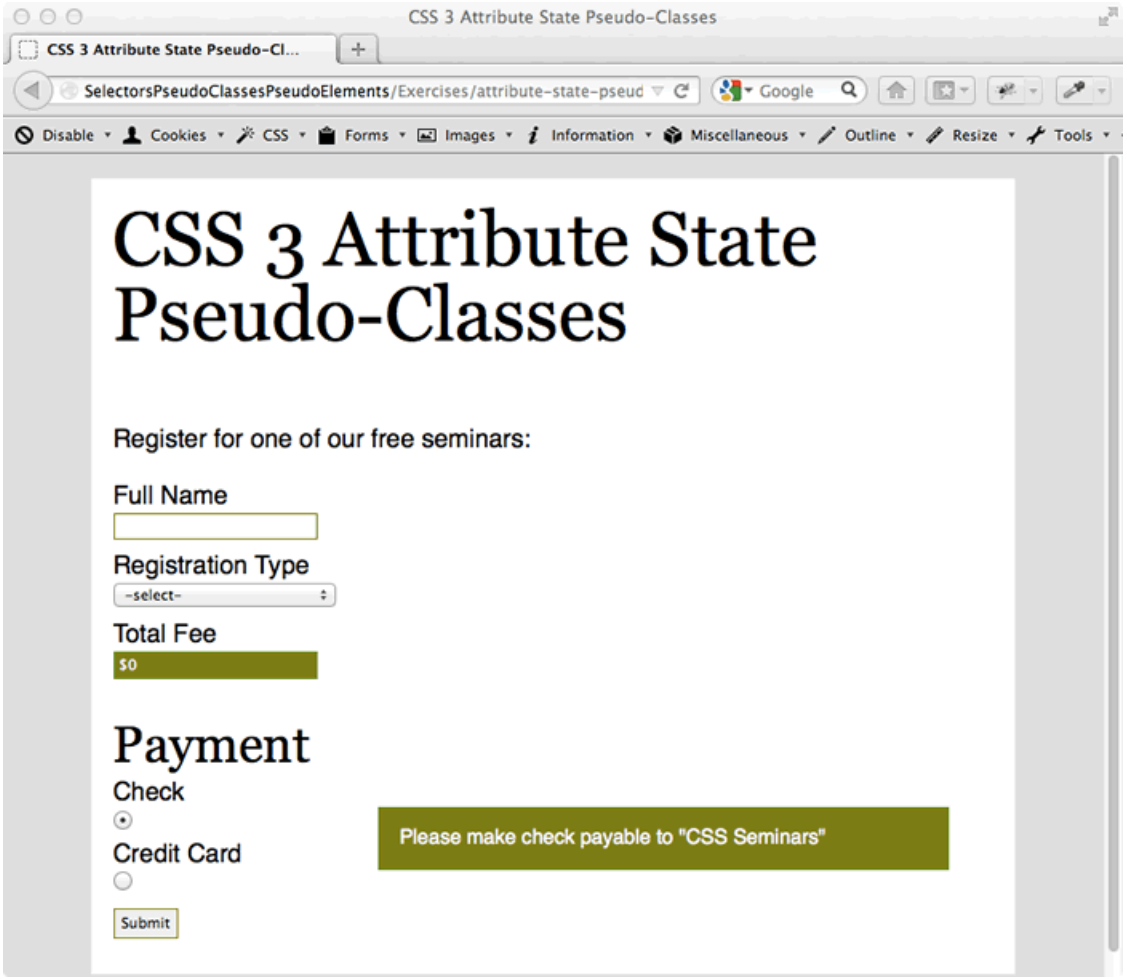
We use the `:target` pseudo-class to style the `<article>` corresponding to the fragment, if one exists, on the end of the URI for the page. A fragment of, for example, "#w3c", would match the `id` of the second `<article>` - and thus the border color, background color, and text color of that item would change.

Exercise 3: Using CSS Attribute-State Pseudo-Classes

15 to 20 minutes

In this exercise, you will style a form to highlight the difference between enabled and disabled form elements and to show/hide informational text based on the checked state of an element.

1. Open `ClassFiles/selectors-pseudoclasses/Exercises/attribute-state-pseudo-classes.html` in a text editor.
2. The form on the page represents a short conference registration form, in which the user enters his or her full name, selects the type of registration ("full" or "sessions only"), and chooses the manner ("check" or "credit card") in which they will make payment.
3. Note that a short bit of JavaScript (by way of the jQuery library) populates the read-only "Total Fee" field in response to the user's selection; this is done for you.
4. Style the page so that the enabled elements are visually distinct from the disabled elements.
5. Add two `<div>` elements, one after each of the radio buttons for payment type. Style the page to hide or display payment information ("make check out to...", "Paypal details appear on next screen") based on the user's choice of payment type. Float the `<div>`s to the right of the payment type radio buttons.
6. Your completed page should look something like this:



7. Check your work in a browser.

Solution: selectors-pseudoclasses/Solutions/attribute-state-pseudo-classes.html

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
 <title>CSS3 Attribute State Pseudo-Classes</title>
 <meta name="viewport" content="width=device-width">
 <link rel="stylesheet" href="../Shared/reset.css">
 <link rel="stylesheet" href="../Shared/style.css">
 <style>
  input:enabled {
   border:1px solid Olive;
  }
  input:disabled {
   border:1px solid OliveDrab;
   background:Olive;
   color:White;
  }
  div.paymentinfo {
   border:1px solid OliveDrab;
   background:Olive;
   color:White;
   font-size:.8em;
   padding:1em;
   display:none;
   float:right;
   width:60%;
   margin-right:5%;
  }
  input:checked + div {
   display:block;
  }
 </style>
 <script src="../Shared/jquery.min.js"></script>
 <script>
  $(document).ready(function(){
   $('#regtype').change(function() {
    regtype = $(this).val();
    if (regtype == 'all') {
     $('#fee').val('$100');
    } else if (regtype == 'sessions') {
     $('#fee').val('$50');
    } else {
     $('#fee').val('$0');
```

```
44.       }
45.      })
46.     });
47.    </script>
48.   </head>
49.   <body>
50.    <div id="main">
51.     <h1>CSS3 Attribute State Pseudo-Classes</h1>
52.     <p>Register for one of our free seminars:</p>
53.     <form action="#" method="get">
54.      <label for="fullname">Full Name</label> <input type="text" name="fullname"
           id="fullname">
55.      <label for="regtype">Registration Type</label>
56.      <select name="regtype" id="regtype">
57.       <option value="">-select-</option>
58.       <option value="all">Sessions &amp;amp; Dinner ($100)</option>
59.       <option value="sessions">Sessions Only ($50)</option>
60.      </select>
61.      <label for="fee">Total Fee</label> <input type="text" name="fee" id="fee"
           disabled="disabled" value="$0">
62.      <h2>Payment</h2>
63.      <label for="check">Check</label> <input type="radio" name="payment" id="pay ↵↵
           ment-check" value="check">
64.      <div id="paymentinfo-check" class="paymentinfo">Please make check payable to
           "CSS Seminars"</div>
65.      <label for="creditcard">Credit Card</label> <input type="radio" name="payment"
           id="payment-creditcard" value="creditcard">
66.      <div id="paymentinfo-creditcard" class="paymentinfo">Secure payment details
           will appear on the next screen</div>
67.      <br>
68.      <input type="submit" value="Submit">
69.     </form>
70.    </div>
71.   </body>
72.   </html>
```


Code Explanation

A bit of jQuery populates the (disabled) `#fee` field in response to the user changing the registration type. We add two `<div>`s, each of `class` “paymentinfo”, to display information about each type of payment.

We style the form elements depending on their state - `:enabled` items get a white background, while `:disabled` items get a dark background and white text.

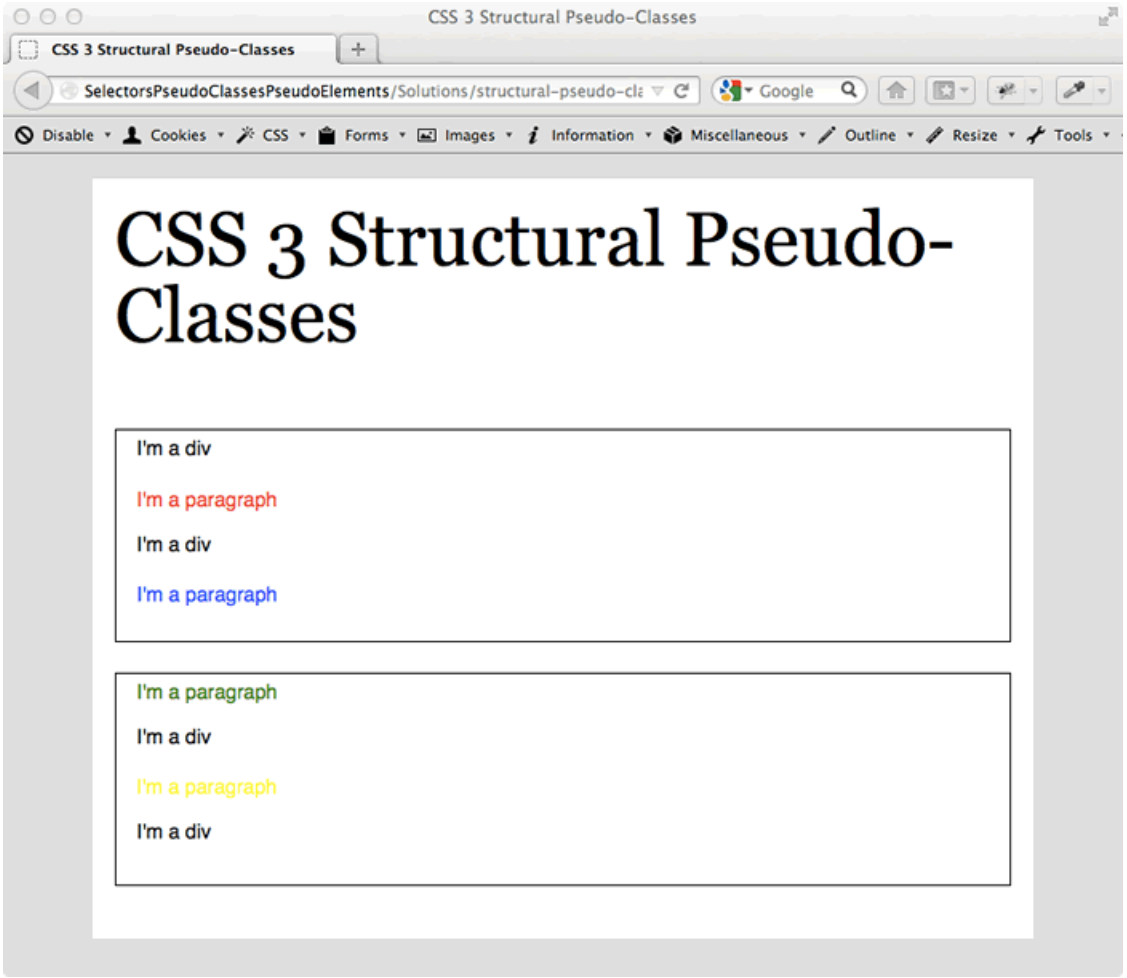
We style the two `<div>`s as initially hidden. We set the `<div>` immediately following a checked `<input>` (`input:checked + div`) to display as block, unhiding the element.

Exercise 4: Using the Structural Attribute Pseudo-Classes

10 to 15 minutes

In this exercise, you will explore the difference between `first-child`/`last-child` and `first-of-type`/`last-of-type`.

1. Open `ClassFiles/selectors-pseudoclasses/Exercises/structural-pseudo-classes.html` in a text editor.
2. The markup for the page, which is created for you, contains two `<div>` elements, each of which contains two `<div>` and two `<p>` elements. Note the order of the contained `<div>`/`<p>` elements differs on the page.
3. Style the page so that the four “I’m a paragraph” elements are colored red, blue, green, and yellow (in that order).
4. Use only `first-child`, `last-child`, `first-of-type`, and `last-of-type` rules to effect the coloring.
5. Your completed page should look something like this:



6. Check your work in a browser.

Solution: selectors-pseudoclasses/Solutions/structural-pseudo-classes.html

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
 <title>CSS3 Structural Pseudo-Classes</title>
 <meta name="viewport" content="width=device-width">
 <link rel="stylesheet" href="../Shared/reset.css">
 <link rel="stylesheet" href="../Shared/style.css">
 <style>
  .container {
   border:1px solid black;
   font-size:.8em;
   padding:.5em 1em;
  }
  p:first-of-type {
   color:red;
  }
  p:last-of-type {
   color:yellow;
  }
  p:first-child {
   color:green;
  }
  p:last-child {
   color:blue;
  }
 </style>
</head>
<body>
 <div id="main">
  <h1>CSS3 Structural Pseudo-Classes</h1>
  <div id="container1" class="container">
   <div>I'm a div</div>
   <p>I'm a paragraph</p>
   <div>I'm a div</div>
   <p>I'm a paragraph</p>
  </div>
  <div id="container2" class="container">
   <p>I'm a paragraph</p>
   <div>I'm a div</div>
   <p>I'm a paragraph</p>
   <div>I'm a div</div>
  </div>
```

```
44.    </div>
45.  </body>
46.  </html>
```

Code Explanation

`p:first-of-type` selects (and colors in red) the second child element in the first (`#container1`) `<div>`, since it is the first child element of type `<p>`. Similarly, `p:last-of-type` colors in yellow the third child (a paragraph) of the second (`#container2`) `<div>`, since it is the last child of type `<p>`.

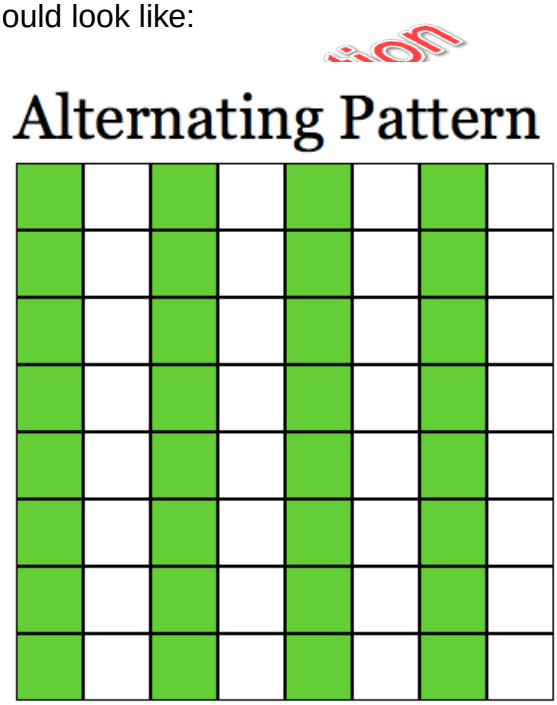
`p:first-child` selects (and colors in green) any element that is both a `<p>` and is the first child of its parent; this targets the first child of the second (`#container2`) `<div>`. Last, `p:last-child` colors in blue the last child of the first `<div>`.

Exercise 5: Using nth-of-type Pseudo-Class

20 to 30 minutes

In this exercise, you will use the nth-of-type pseudo-class to style an 8-by-8 tile of squares (`<div>`s) with alternating colors.

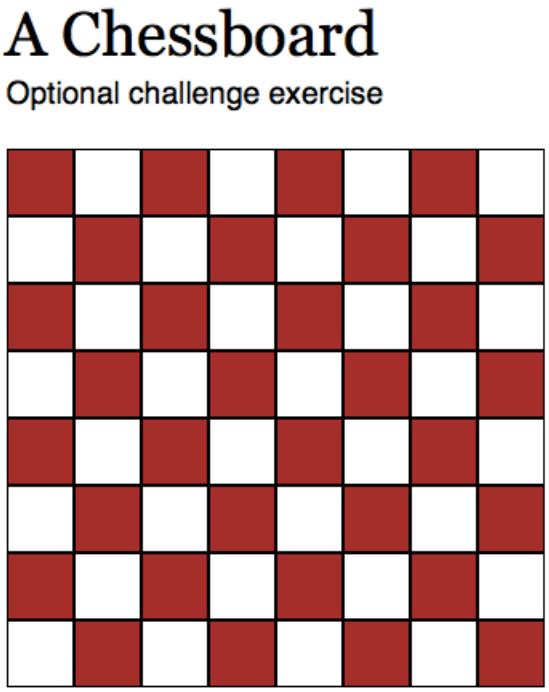
1. Open `ClassFiles/selectors-pseudoclasses/Exercises/structural-pseudo-classes-nth.html` in a text editor.
2. Note that the section of the page titled “Alternating Pattern” displays 64 `<div>`s; style these, using `nth-of-type`, to display in an 8-by-8 grid with each alternating columns of green.
3. Your solution should look like:



4. Check your work in a browser.

Challenge

1. Open `ClassFiles/selectors-pseudoclasses/Exercises/structural-pseudo-classes-nth-challenge.html`) in a text editor.
2. Note that the section titled “Chessboard” displays 64 `<div>`s; style these as a chessboard (also using `nth-of-type`, as such:



3. Check your work in a browser.

Solution: selectors-pseudoclasses/Solutions/structural-pseudo-classes-nth.html

```
1.   <!DOCTYPE html>
2.   <html>
3.   <head>
4.    <meta charset="utf-8">
5.    <title>CSS3 "nth" Structural Pseudo-Classes</title>
6.    <meta name="viewport" content="width=device-width">
7.    <link rel="stylesheet" href="../Shared/reset.css">
8.    <link rel="stylesheet" href="../Shared/style.css">
9.    <style>
10.    #alternating div {
11.         width:40px;height:40px;border:1px solid black;
12.         float:left;
13.         margin:0;
14.    }
15.
16.    #alternating div:nth-of-type(2n+1) {
17.     background:LimeGreen;
18.    }
19.
20.    #alternating div:nth-of-type(8n+1) {
21.        clear:left;
22.    }
23.   </style>
24.  </head>
25.  <body>
26.   <div id="main">
27.    <h1>CSS3 "nth" Structural Pseudo-Classes</h1>
28.    <h2>Alternating Pattern</h2>
29.    <div id="alternating">
30.     <div></div><div></div><div></div><div></div>
31.     <div></div><div></div><div></div><div></div>
32.
       ##################################################################################################
     -------Lines 33 through 37 Omitted-------
```


Code Explanation

To effect the eight-to-a-row pattern, we size each `<div>` as a square (with `width` and `height` both `40px`), float each `<div>` `left`, and use `chessboard div:nth-of-type(8n+1)` to clear the last `<div>` in each row.

We use `div:nth-of-type(2n+1)` to target every other element, giving it a green background. We could also have used `div:nth-of-type(odd)` to achieve the same effect.

Challenge Solution:
selectors-pseudoclasses/Solutions/structural-pseudo-classes-nth-challenge.html

```
1.   <!DOCTYPE html>
2.   <html>
3.   <head>
4.    <meta charset="utf-8">
5.    <title>CSS3 "nth" Structural Pseudo-Classes</title>
6.    <meta name="viewport" content="width=device-width">
7.    <link rel="stylesheet" href="../Shared/reset.css">
8.    <link rel="stylesheet" href="../Shared/style.css">
9.    <style>
10.    #chessboard div {
11.         width:40px;height:40px;border:1px solid black;
12.         float:left;
13.         margin:0;
14.    }
15.
16.    #chessboard div:nth-of-type(16n+1),
17.    #chessboard div:nth-of-type(16n+3),
18.    #chessboard div:nth-of-type(16n+5),
19.    #chessboard div:nth-of-type(16n+7),
20.    #chessboard div:nth-of-type(16n+10),
21.    #chessboard div:nth-of-type(16n+12),
22.    #chessboard div:nth-of-type(16n+14),
23.    #chessboard div:nth-of-type(16n+16) {
24.        background-color:Brown;
25.    }
26.
27.    #chessboard div:nth-of-type(8n+1) {
28.        clear:left;
29.    }
30.   </style>
31.  </head>
32.  <body>
33.   <div id="main">
34.    <h1>CSS3 "nth" Structural Pseudo-Classes</h1>
35.    <h2>Chessboard</h2>
36.    <div id="chessboard">
37.     <div></div><div></div><div></div><div></div>
38.     <div></div><div></div><div></div><div></div>
39.
```

```
-------Lines 40 through 44 Omitted-------
```

Code Explanation

We use the same strategy as above to create the 8-by-8 grid - sizing, floating, and clearing the last `<div>` in each row.

Using a single CSS rule like `div:nth-of-type(2n+1)` won't work here, since the colored pattern in each chessboard row is staggered. Instead, we use eight CSS rules to color every sixteenth element, offset appropriately: `#chessboard div:nth-of-type(16n+1)`, `#chessboard div:nth-of-type(16n+3)`, etc.

Conclusion

In this lesson, you have learned:

- How to use new CSS3 selectors and pseudo-classes to style pages.

LESSON 3
Fonts and Text Effects

Topics Covered

☑ How to use fonts from services offered by Google and Adobe.

☑ How to define custom fonts with the `@font-face` rule.

☑ How to style text with `text-shadow` and `word-wrap`

Introduction

The adoption of the `@font-face` rule by more and more browsers means we are no longer constrained to choosing fonts from a very small group of typefaces. We'll look at free or low-cost services for custom fonts, the `@font-face` rule itself, and some options for styling fonts.

3.1. Fonts and Text Effects

❖ 3.1.1. Fonts on the Web

Years ago, we had to limit our choice of fonts for web text to a small set of common typefaces, the CSS Web Safe Fonts: Arial, Times, Helvetica, Verdana, and a few others were our only choices. Using a "nonstandard" font meant either rendering the font as a GIF or PNG image - which meant that the text was opaque to search engines and search spiders, and was difficult to edit when we needed to make a change - or using some kind of font replacement technique, like sIFR, which often depended on a cumbersome mixture of JavaScript and Flash.

Now, as browsers support the `@font-face` rule and with the advent of tools like Google's Web Fonts, Typekit, and others, we can use many, many more fonts in our text, reasonably sure that most users will see the font of our choosing - and that older browsers will degrade gracefully to show a backup font of our choosing. We'll look first at using external services and then at the `@font-face` rule, and finish up with some new ways CSS allows us to style text.

❖ 3.1.2. Font Services

Google Web Fonts

`Google Web Fonts (https://fonts.google.com/)` service is pretty easy: link to the Google web fonts style sheet with a URL that specifies one or more fonts you wish to use, then reference the font in the standard manner in your CSS. Google's style sheet uses the CSS `@font-face` rule (more on this later), but we don't need to know anything about it here; conveniently, the font just works. Well, it just works for browsers that support `@font-face`: Internet Explorer from version 9 (with partial support in version 8), recent versions of Webkit and Mozilla browsers, Edge, and recent versions of mobile browsers. For those older browsers, it's a good idea to supply a fallback font (the generic `serif` or `sans-serif` if nothing else.)

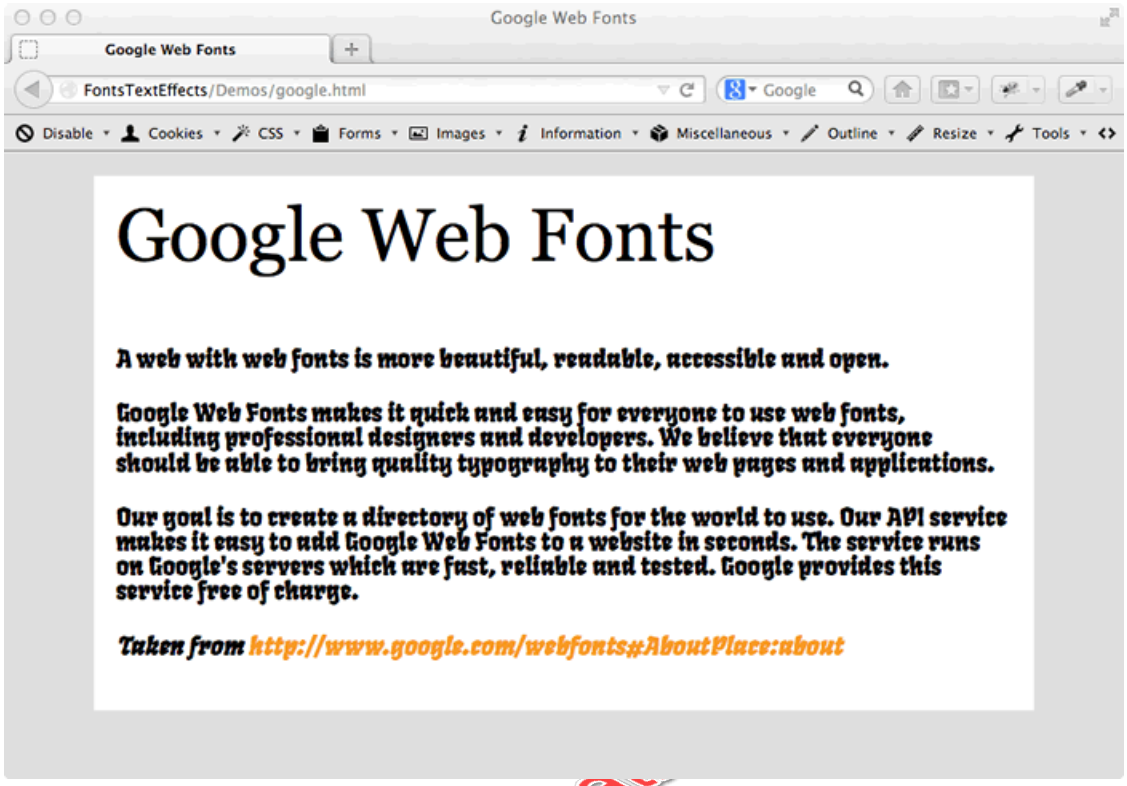
Google Fonts, like most font services, allows you to customize the weights and styles of the font to be provided. For a give font, you might need only the normal weight, a bold weight, and a light italic style of the font. Reducing the weights and styles included means a smaller (and this quicker) download for visitors to your page.

Demo 3.1: fonts-text-effects/Demos/google.html

```
1.   <!DOCTYPE html>
2.   <html>
3.   <head>
4.    <meta charset="utf-8">
5.    <title>Google Web Fonts</title>
6.    <meta name="viewport" content="width=device-width">
7.    <link rel="stylesheet" href="../Shared/reset.css">
8.    <link rel="stylesheet" href="../Shared/style.css">
9.    <link rel="stylesheet" href="http://fonts.googleapis.com/css?family=Fruktur">
10.   <style>
11.    #main p {
12.     font-family:Fruktur, serif;
13.    }
14.   </style>
15.  </head>
16.  <body>
17.   <div id="main">
18.    <h1>Google Web Fonts</h1>
19.    <p>A web with web fonts is more beautiful, readable, accessible and open.</p>
20.    <p>Google Web Fonts makes it quick and easy for everyone to use web fonts,
              including professional designers and developers. We believe that every ↵↵
              one should be able to bring quality typography to their web pages and
               applications.</p>
21.    <p>Our goal is to create a directory of web fonts for the world to use. Our
              API service makes it easy to add Google Web Fonts to a website in sec ↵↵
              onds. The service runs on Google's servers which are fast, reliable
              and tested. Google provides this service free of charge.</p>
22.    <p><em>Taken from <a href="http://www.google.com/webfonts#About ↵↵
              Place:about">http://www.google.com/webfonts#About ↵↵
              Place:about</a></em></p>
23.   </div>
24.  </body>
```


Code Explanation

Open up `ClassFiles/fonts-text-effects/Demos/google.html` to see an example of a basic use of Google Web Fonts:



We visited Google Web Fonts, chose a font ("Fruktur"), and added it to our page via the `<link>` tag; note that the `href` attribute of the `<link>` tag on our page is `http://fonts.googleapis.com/css?family=Fruktur`.

After that, using the font on our page simply requires that we style the paragraphs in the `#main` element with `font-family:Fruktur, serif` - we add `serif` as a fallback for those browsers that don't support the `@font-face` rule.

You can easily search, filter, and sort the available set of fonts from the `Google Web Fonts homepage (https://fonts.google.com/)`; the "Add to Collection" feature will generate the correct style sheet URL based on the fonts you have chosen. Keep in mind that, given that using Google's fonts means users will download font files when they visit your pages, you'll want to restrict the number of fonts you use from Google to as few as possible.

Typekit & Adobe Edge Web Fonts

`Adobe Edge Web Fonts (https://edgewebfonts.adobe.com/)` is a service very similar to Google Web Fonts; `Adobe's Typekit (https://typekit.com/)` is the fee-based (and more well-known) equivalent to Adobe Edge Web Fonts. With both you can serve up fonts

hosted by Adobe; unlike Google Web Fonts, you include Adobe's fonts via a link to a JavaScript (rather than style sheet) file. Typekit offers considerably more (and, some might say, better) fonts - but a fee (ranging from roughly $25 to $100 per year, depending on number of sites, fonts, and pageviews, as of this writing) is required. Typekit does offer a free trial.

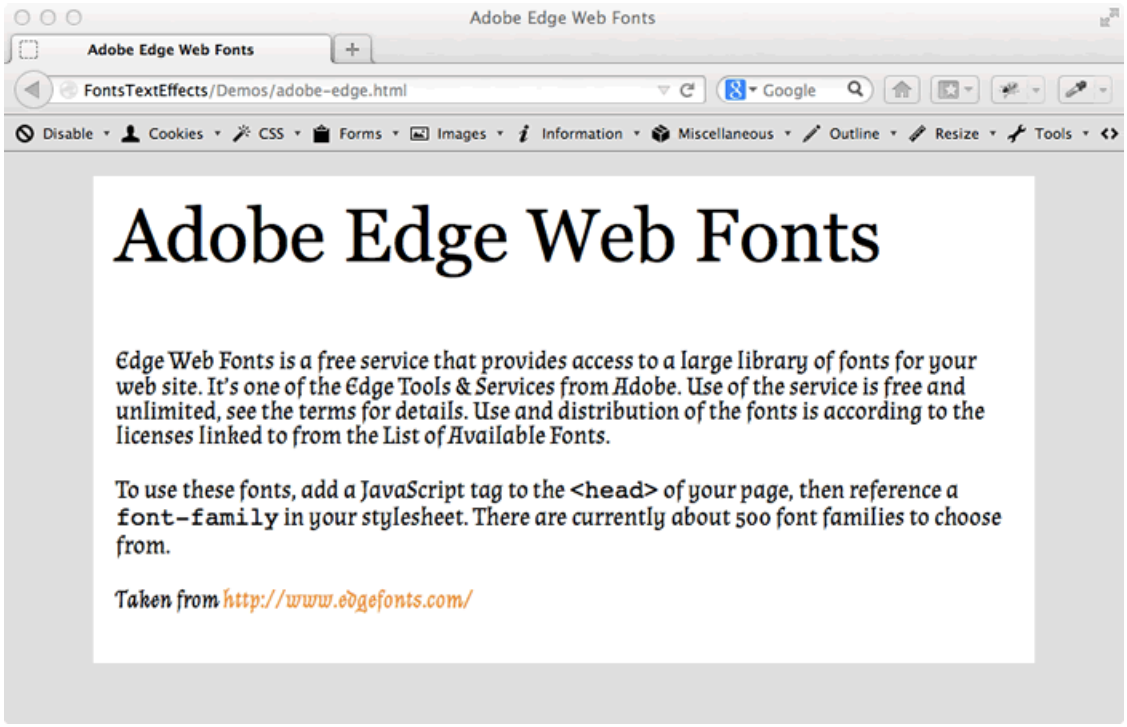
To use Adobe Edge Web Fonts, select from the `list of available fonts (https://edgewebfonts.adobe.com/fonts)` and include the appropriate JavaScript file. The `src` from which you link the JavaScript file has a URL of the form `http://use.edgefonts.net/[font(s)]`; you can specify one or more styles (regular, bold, etc.), one or more subsets of the font, and more than one font, all in the same `<script>` tag. If working locally (i.e., accessing your file directly from your computer, via a URL that starts with `file://`, you'll need to specify the URL starting with `http://`. If you serve the files from a web server (like IIS or Apache), Adobe recommends the use of the protocol-less form; a URL like `//use.edgefonts.net/averia-libre.js` will automatically use the secure `https://` when (and only when) needed.

Demo 3.2: fonts-text-effects/Demos/adobe-edge.html

```
1.  <!DOCTYPE html>
2.  <html>
3.  <head>
4.   <meta charset="utf-8">
5.   <title>Adobe Edge Web Fonts</title>
6.   <meta name="viewport" content="width=device-width">
7.   <link rel="stylesheet" href="../Shared/reset.css">
8.   <link rel="stylesheet" href="../Shared/style.css">
9.   <script src="http://use.edgefonts.net/almendra.js"></script>
10.  <style>
11.   #main p {
12.    font-family:Almendra, serif;
13.   }
14.  </style>
15. </head>
16. <body>
17.  <div id="main">
18.   <h1>Adobe Edge Web Fonts</h1>
19.   <p>Edge Web Fonts is a free service that provides access to a large library
          of fonts for your web site. It's one of the Edge Tools &amp; Services
           from Adobe. Use of the service is free and unlimited, see the terms
          for details. Use and distribution of the fonts is according to the li ↵↵
          censes linked to from the List of Available Fonts.</p>
20.   <p>To use these fonts, add a JavaScript tag to the <code>&lt;head&gt;</code>
          of your page, then reference a <code>font-family</code> in your
          stylesheet. There are currently about 500 font families to choose
          from.</p>
21.   <p><em>Taken from <a href="http://www.edgefonts.com/">http://www.edge ↵↵
          fonts.com/</a></em></p>
22.  </div>
23. </body>
```


Code Explanation

Open `ClassFiles/fonts-text-effects/Demos/adobe-edge.html` to see an example of a basic use of Adobe Edge Web Fonts:



For this example, we chose the font "Almendra" and added it to our page via the `<script>` tag; note that the `src` attribute of the `<script>` tag on our page points to `http://use.edgefonts.net/almendra.js`.

After linking to Adobe's JavaScript file, we use the font on our page by styling the paragraphs in the `#main` element with `font-family:Almendra, serif` - we add `serif` to ensure that browsers that don't support the `@font-face` rule will at least render our text with a serifed font.

Exercise 6: Using Font Services

10 to 20 minutes

In this exercise, you will use a font from Google Web Fonts or from Adobe Edge Web Fonts.

1. Open `ClassFiles/fonts-text-effects/Exercises/font.html` in a browser and in a code editor.
2. Choose a font from either `Google Web Fonts (https://fonts.google.com/)` or `Adobe Edge Web Fonts (http://www.edgefonts.com/fonts)`.
3. Use the font to style the text in the exercise file.

Solution: fonts-text-effects/Solutions/font.html

```
1.   <!DOCTYPE html>
2.   <html>
3.   <head>
4.    <meta charset="utf-8">
5.    <title>Font Services</title>
6.    <meta name="viewport" content="width=device-width">
7.    <link rel="stylesheet" href="../Shared/reset.css">
8.    <link rel="stylesheet" href="../Shared/style.css">
9.    <link rel="stylesheet" href="http://fonts.googleapis.com/css?family=Exo">
10.   <style>
11.    #main {
12.     font-family:Exo, Arial, sans-serif;
13.    }
14.   </style>
15.  </head>
16.  <body>
17.   <div id="main">
18.    <h1>Font Services</h1>
19.    <p>A font provides a resource containing the visual representation of charac ↵↵
              ters. At the simplest level it contains information that maps character
               codes to shapes (called glyphs) that represent these characters. Fonts
               sharing a common design style are commonly grouped into font families
               classified by a set of standard font properties. Within a family, the
               shape displayed for a given character can vary by stroke weight, slant
               or relative width, among others. A given font face is described by a
               unique combination of these properties. For a given range of text,
              CSS font properties are used to select a font family and a specific
              font face within that family to be used when rendering that text.</p>
20.    <p><em>Taken from <a href="http://www.w3.org/TR/css3-fonts/#introduc ↵↵
              tion">http://www.w3.org/TR/css3-fonts/#introduction</a></em></p>
21.   </div>
22.  </body>
```


Code Explanation

We chose the font “Exo” from Google Web Fonts:

To use this font, we added a style sheet `<link>` tag that loaded in the style sheet `http://fonts.googleapis.com/css?family=Exo`. We then referenced `Exo` in the `font-family` property.

Exercise 7: Using @font-face

10 to 20 minutes

In this exercise, you will download a font of your choosing from Font Squirrel, use the `@font-face` rule to enable the font for your page, and style text with the font.

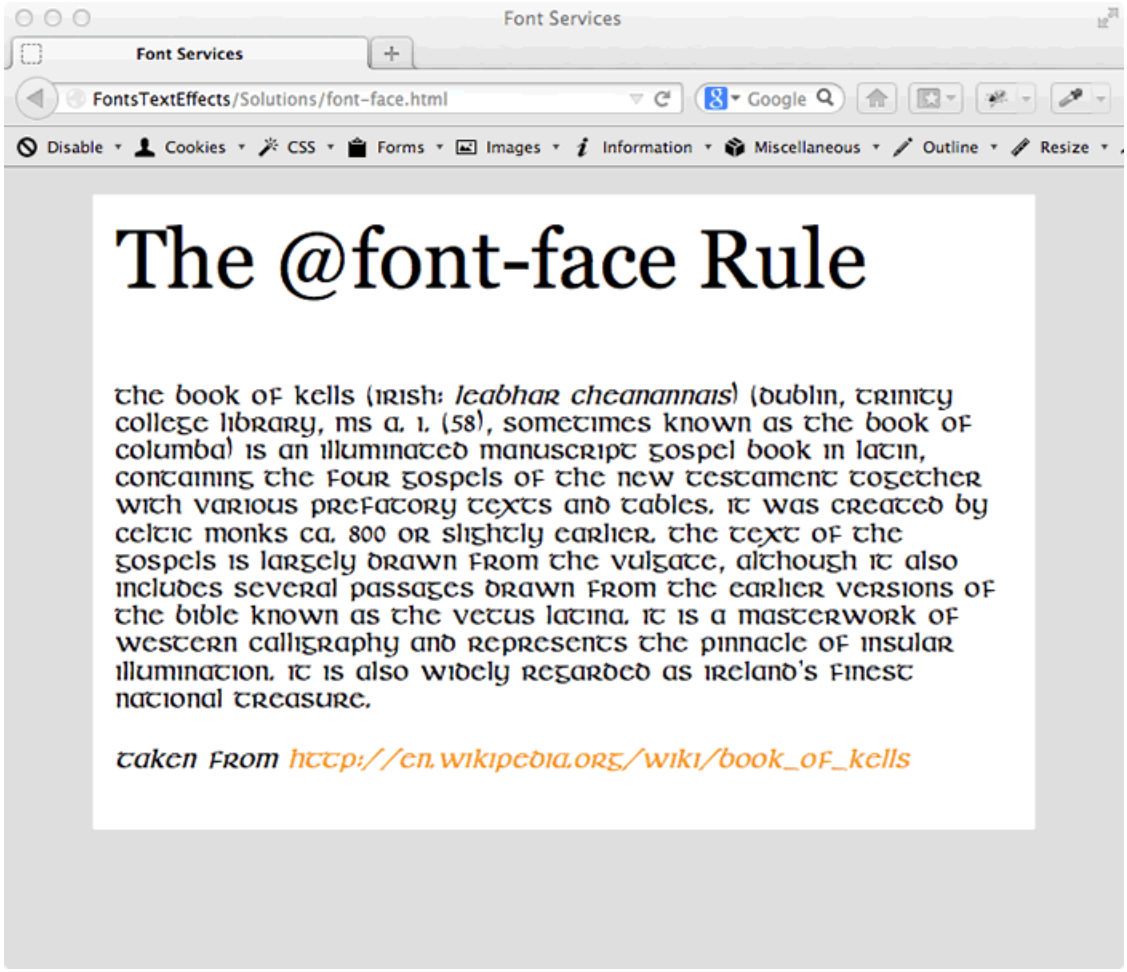
1. Open `ClassFiles/fonts-text-effects/Exercises/font-face.html` in a browser and in a code editor.
2. Browse the `list of available fonts (http://www.fontsquirrel.com/fontface)` on Font Squirrel - pick one you like.
3. From the listing page, click the “Get Kit” link, save the download to your computer, and unzip the archive.
4. Move the font files (.eot, .svg, .ttf, .woff, etc.) files to the project directory.
5. Open the downloaded `stylesheet.css` file; copy the `@font-face` rule to the `<style>` block in `font-face.html`.
6. Style the text (about the Irish *Book of Kells*) with the font you have just downloaded.

Solution: fonts-text-effects/Solutions/font-face.html

```
1.   <!DOCTYPE html>
2.   <html>
3.   <head>
4.    <meta charset="utf-8">
5.    <title>Font Services</title>
6.    <meta name="viewport" content="width=device-width">
7.    <link rel="stylesheet" href="../Shared/reset.css">
8.    <link rel="stylesheet" href="../Shared/style.css">
9.    <style>
10.    @font-face {
11.     font-family: 'KellsSDRegular';
12.     src: url('Kells_SD-webfont.eot');
13.     src: url('Kells_SD-webfont.eot?#iefix') format('embedded-opentype'),
14.      url('Kells_SD-webfont.woff') format('woff'),
15.      url('Kells_SD-webfont.ttf') format('truetype'),
16.      url('Kells_SD-webfont.svg#KellsSDRegular') format('svg');
17.     font-weight: normal;
18.     font-style: normal;
19.    }
20.    #main {
21.     font-family:'KellsSDRegular';
22.    }
23.   </style>
24.  </head>
25.  <body>
26.   <div id="main">
27.    <h1>The @font-face Rule</h1>
28.    <p>The Book of Kells (Irish: <em>Leabhar Cheanannais</em>) (Dublin, Trinity
          College Library, MS A. I. (58), sometimes known as the Book of Columba)
           is an illuminated manuscript Gospel book in Latin, containing the four
           Gospels of the New Testament together with various prefatory texts
          and tables. It was created by Celtic monks ca. 800 or slightly earlier.
           The text of the Gospels is largely drawn from the Vulgate, although
          it also includes several passages drawn from the earlier versions of
          the Bible known as the Vetus Latina. It is a masterwork of Western
          calligraphy and represents the pinnacle of Insular illumination. It is
           also widely regarded as Ireland's finest national treasure.</p>
29.    <p><em>Taken from <a href="http://en.wikipedia.org/wi  ↵↵
          ki/Book_of_Kells">http://en.wikipedia.org/wi  ↵↵
          ki/Book_of_Kells</a></em></p>
30.   </div>
31.  </body>
```

Code Explanation

We chose the font `Kells SD (http://www.fontsquirrel.com/fonts/Kells-SD)` from Font Squirrel.



We moved the downloaded font files to the same directory as our `.html` file, copied Font Squirrel's `@font-face` rule (from the `stylesheet.css` file we downloaded), and used `font-family:'KellsSDRegular'` to style the text.

Exercise 8: Text Shadow and Word Wrap

10 to 20 minutes

In this exercise, you will use the text-shadow and word-wrap properties to style some text content.

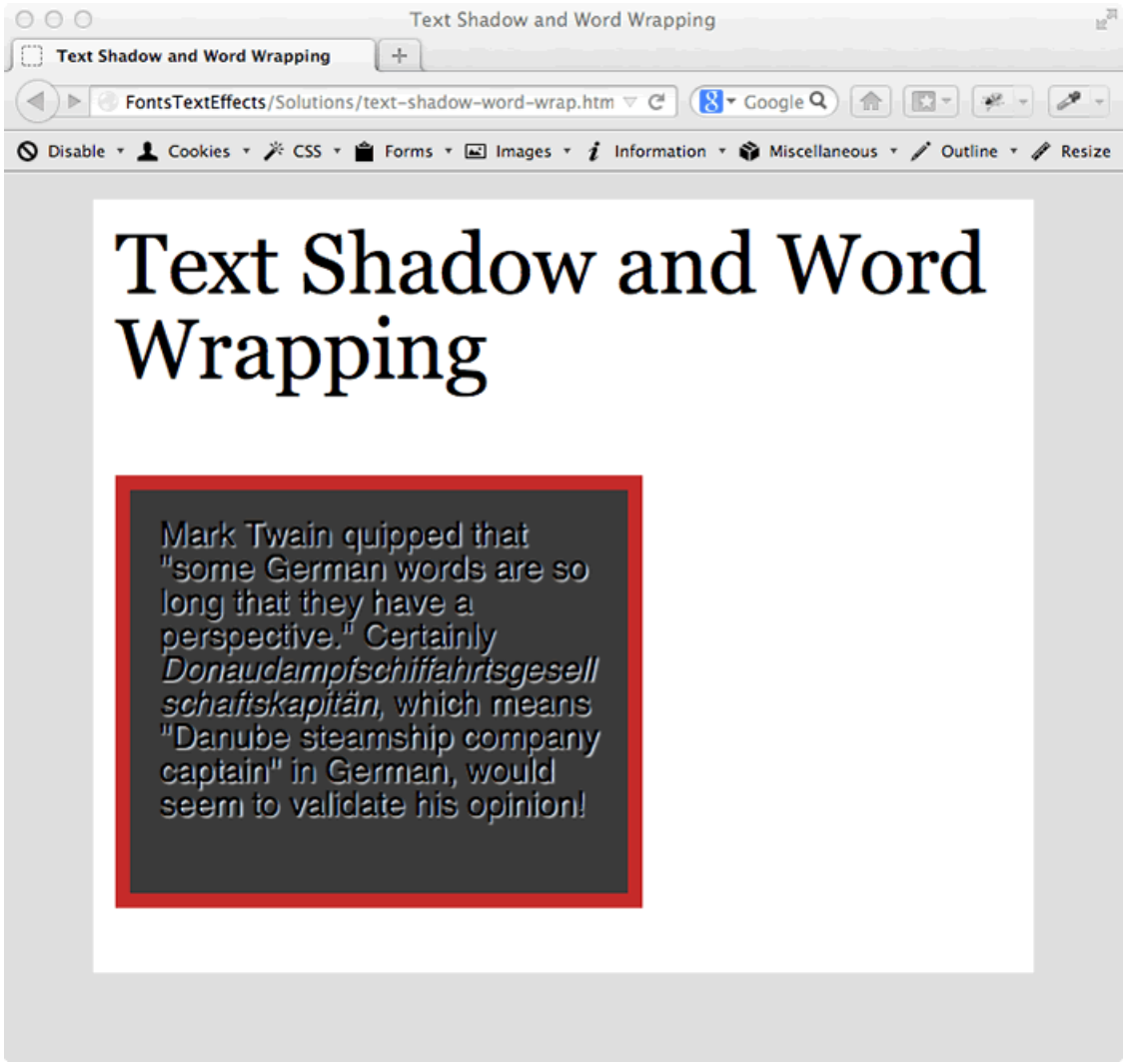
1. Open `ClassFiles/fonts-text-effects/Exercises/text-shadow-word-wrap.html` in a browser and in code editor.
2. Add styles to render the text with a slight shadow; experiment with background color, text color, and shadow color/size/blur to see what you think looks best.
3. Write CSS to fix the fact that the long German word runs off the right of its fixed-width container.

Solution: fonts-text-effects/Solutions/text-shadow-word-wrap.html

```
1.  <!DOCTYPE html>
2.  <html>
3.  <head>
4.   <meta charset="utf-8">
5.   <title>Text Shadow and Word Wrapping</title>
6.   <meta name="viewport" content="width=device-width">
7.   <link rel="stylesheet" href="../Shared/reset.css">
8.   <link rel="stylesheet" href="../Shared/style.css">
9.   <link rel="stylesheet" href="http://fonts.googleapis.com/css?family=Fruktur">
10.  <style>
11.   #border {
12.    width:300px;
13.    padding:20px;
14.    background-color:hsl(0, 0%, 20%);
15.    border:10px solid hsl(0, 67%, 45%);
16.   }
17.   #border p {
18.    font-size:1.2em;
19.    color:hsl(0, 0%, 0%);
20.    text-shadow:1px 1px 1px hsl(0, 0%, 100%);
21.    word-wrap:break-word;
22.   }
23.  </style>
24. </head>
25. <body>
26.  <div id="main">
27.   <h1>Text Shadow and Word Wrapping</h1>
28.   <div id="border">
29.    <p>Mark Twain quipped that "some German words are so long that they have a
          perspective." Certainly <em>Donaudampfschiffahrtsgesellschaft ↵↵
          skapitän</em>, which means "Danube steamship company captain" in German,
           would seem to validate his opinion!</p>
30.   </div>
31.  </div>
32. </body>
```


Code Explanation

We chose to show dark text on a dark background with a white text shadow - perhaps not the most legibile display for this content in terms of contrast, but an interesting effect:



We use `word-wrap:break-word` to force the long word to break.

Conclusion

In this lesson, you have learned:

- How to use Google and Adobe services to serve custom fonts.
- How to render custom fonts with the `@font-face` rule.
- How to style text using the `text-shadow` and `word-wrap` properties.

LESSON 4

Colors, Gradients, Background Images, and Masks

Topics Covered

☑ How CSS offers new ways to specify color values, including specifying opacity via an alpha value.

☑ How to use the CSS `opacity` property to change the opacity of elements.

☑ How to set multiple background images for a given element.

☑ How to set properties for background images with `background-clip`, `background-origin`, and `background-size`.

Introduction

CSS Level 3 introduces a variety of ways in which we can specify both color, including opacity, and background images for elements.

4.1. Colors, Gradients, Background Images, and Masks

❖ 4.1.1. Color

CSS has traditionally offered three ways to specify color:

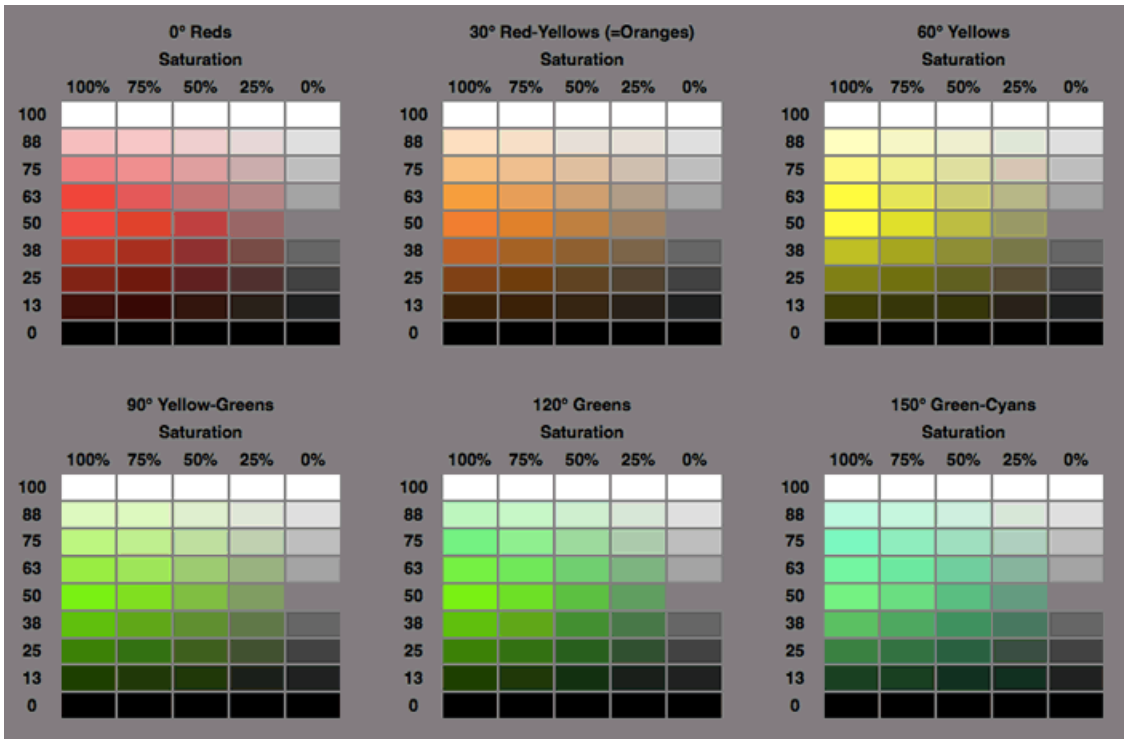
- Named colors
    - `aqua`
    - `black`
    - `blue`
    - `fuchsia`
    - `gray`

    - `grey`
    - `green`
    - `lime`
    - `maroon`
    - `navy`
    - `olive`
    - `purple`
    - `red`
    - `silver`
    - `teal`
    - `white`
    - `yellow`
- Hexadecimal
    - `#FFFFFF` - white
    - `#FFF` - white
    - `#FF0000` - red
    - `#F00` - red
    - etc.
- RGB
    - `rgb(255, 255, 255)` - white
    - `rgb(100%, 100%, 100%)` - white
    - `rgb(255, 0, 0)` - red
    - `rgb(100%, 0%, 0%)` - red
    - etc.

CSS Level 3 introduced a new way to specify color values: hue, saturation, and light (HSL) values. As the `W3C specification (http://www.w3.org/TR/css3-color/#hsl-color)` states:

Hue is represented as an angle of the color circle (i.e., the rainbow represented in a circle). This angle is so typically measured in degrees that the unit is implicit in CSS; syntactically, only a `<number>` is given. By definition red=0=360, and the other colors are spread around the circle, so green=120, blue=240, etc. As an angle, it implicitly wraps around such that -120=240 and 480=120. One way an implementation could normalize such an angle x to the range [0,360) (i.e., zero degrees, inclusive, to 360 degrees, exclusive) is to compute (((x mod 360) + 360) mod 360). Saturation and lightness are represented as percentages. 100% is full saturation, and 0% is a shade of gray. 0% lightness is black, 100% lightness is white, and 50% lightness is 'normal'.

The W3C offers a `set of tables (http://www.w3.org/TR/css3-color/#hsl-examples)` as examples of HSL values:



Also new in CSS Level 3 is the ability to specify the alpha channel of a color, setting the opacity of the given color. We can specify color and alpha with `rgba`:

```
/* green with 50% opacity */
      background-color:rgba(0, 255, 0, 0.5);
```

or with `hsla`:


```
/* green with 50% opacity */
      color:hsla(120, 100%, 50%, 0.5);
```

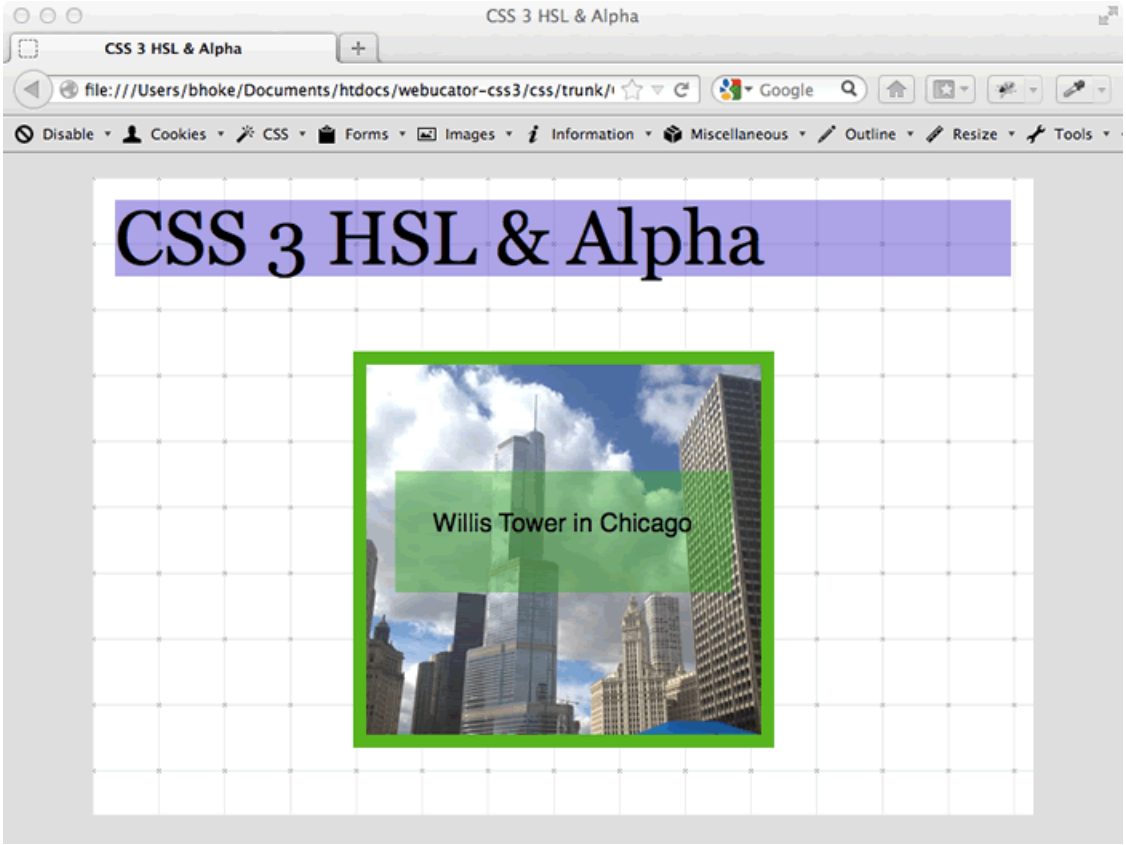
Demo 4.1: colors-gradients-background-images-masks/Demos/hsl-alpha.html

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
 <title>CSS3 HSL &amp; Alpha</title>
 <meta name="viewport" content="width=device-width">
 <link rel="stylesheet" href="../Shared/reset.css">
 <link rel="stylesheet" href="../Shared/style.css">
 <style type="text/css">
  #main {
   background-image:url(gridme.png);
   background-repeat: repeat;
  }
  h1 {
   background-color:rgba(40, 50, 200, 0.4);
  }
  #bg {
   margin:0 auto 5% auto;
   background-image:url(chicago.jpg);
   background-repeat:no-repeat;
   background-position: 10px 10px;
   background-color:hsl(120, 75%, 40%);
   width:320px;
   height:301px;
   position: relative;
  }
  #caption {
   position:absolute;
   left:10%;
   top:30%;
   width:80%;
   height:20%;
   padding-top:10%;
   text-align: center;
   background-color:hsla(120, 75%, 40%, 0.4);
  }
 </style>
</head>
<body>
 <div id="main">
  <h1>CSS3 HSL &amp; Alpha</h1>
  <div id="bg">
   <div id="caption">
```

```
      Willis Tower in Chicago
    </div>
  </div>
 </div>
</body>
</html>
```

Code Explanation

Our example, `ClassFiles/colors-gradients-background-images-masks/Demos/hsl-alpha.html`, shows a page with a tiled background pattern, a title with semitransparent purple background, and an image element with background color and a semitransparent caption block in the center:



We use a repeated background image on the `#main <div>`, a grid pattern. We apply a background color to the `<h1>` using `rgba`, setting the alpha to 0.4 - a 40% opacity.

The `#bg <div>` has a background color, using `hsl` (with no alpha setting), and a `no-repeat` background image (which doesn't cover the entire element). The `#caption <div>`, positioned absolutely on top of the `#bg <div>`, has a semitransparent green background, set with `hsla`.

❖ 4.1.2. The `opacity` Property

CSS's opacity property makes it easy to change the opacity of images, text, `<div>`s, and other elements. Where before we might have resorted to PNG images with a semitransparent alpha setting, we can now change the opacity via CSS. Even better, we can change opacity more easily with `:hover` to create rollover effects. Let's look at an example.

Demo 4.2: colors-gradients-background-images-masks/Demos/opacity.html

```
1.   <!DOCTYPE html>
2.   <html>
3.   <head>
4.    <meta charset="utf-8">
5.    <title>CSS3 Opacity</title>
6.    <meta name="viewport" content="width=device-width">
7.    <link rel="stylesheet" href="../Shared/reset.css">
8.    <link rel="stylesheet" href="../Shared/style.css">
9.    <style type="text/css">
10.    #img1 {
11.     float:left;
12.    }
13.    #img1:hover {
14.     opacity:0.5;
15.    }
16.
17.    #bg {
18.     background-image:url(chicago.jpg);
19.     width:300px;
20.     height:281px;
21.     float:right;
22.    }
23.    #caption {
24.     width:80%;
25.     height:20%;
26.     margin:30% 10%;
27.     padding-top:15%;
28.     text-align: center;
29.     background:#fff;
30.     opacity:0.2;
31.    }
32.    #bg:hover #caption {
33.     opacity:0.9;
34.    }
35.
36.    #footer {
37.     clear:both;
38.    }
39.    #footer p:hover {
40.     opacity:0.5;
41.    }
42.   </style>
43.  </head>
```

```
<body>
 <div id="main">
  <h1>CSS3 Opacity</h1>
  <img src="chicago.jpg" alt="Chicago skyline" id="img1">

  <div id="bg">
   <div id="caption">
    Willis Tower in Chicago
   </div>
  </div>

  <div id="footer">
   <p>Left image opacity changes on hover; right image caption div opacity changes
          on hover</p>
  </div>
 </div>
</body>
</html>
```


Code Explanation

The page presents three main elements: an image on the left, an image on the right, and a footer. Here's a screenshot of the page before any changes to element opacity:

We apply a change to some aspect of each of the three elements when the user mouses over, via the `:hover` selector:

- The left image, without caption, changes to 50% opacity (`opacity:0.5;`) when moused over.
- The caption on top of the right image (the image is itself a background) changes from its initial 20% opacity to 90% opacity when moused over. Note that the opacity for the text in the "caption" `<div>` is also affected by the opacity change - unlike, in our earlier example, where the color setting (in which we set the alpha value) for a `<div>` does not change the opacity of child elements.
- The opacity of the footer text changes to 50% opacity when moused over.

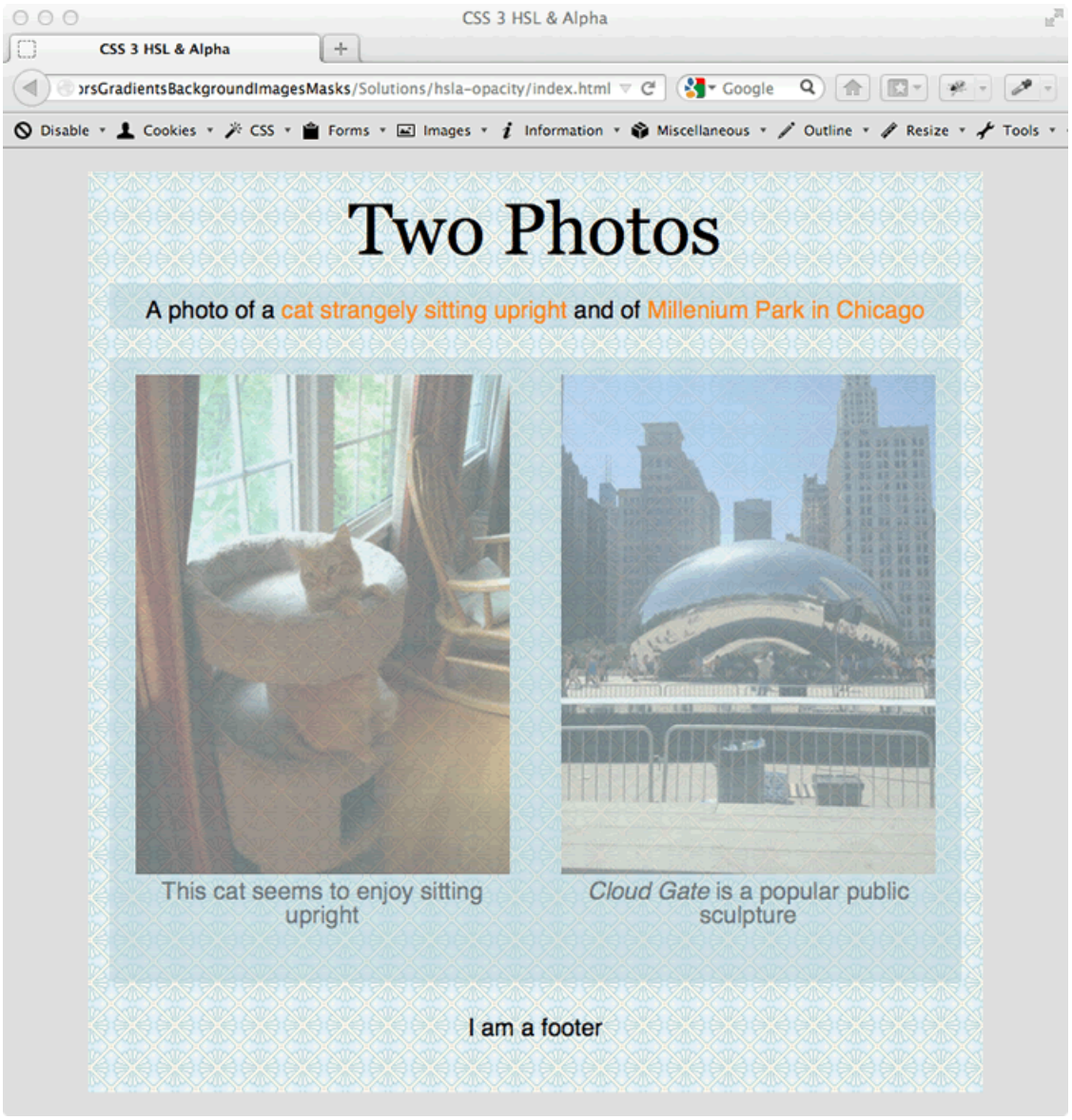
CSS's opacity property works well in all modern browser versions but for Internet Explorer before version 9. For earlier versions of IE, we can use `filter:alpha(opacity=xx);`, where `xx` represents the desired opacity in percentage units (without the percent sign). We won't include the older, IE-specific CSS in our code.

Exercise 9: Using HSL & Opacity

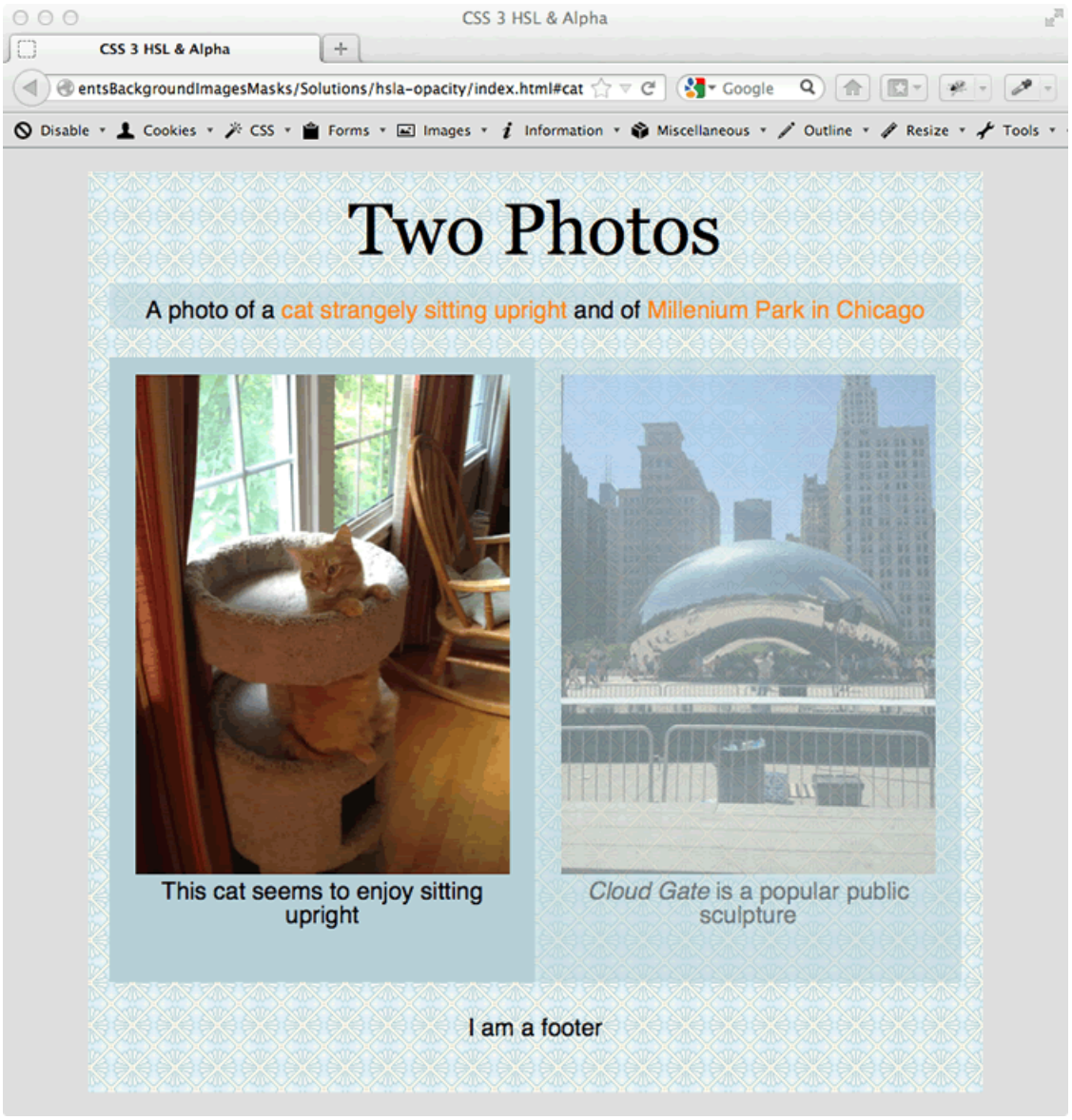
15 to 25 minutes

In this exercise, you will render a design that incorporates `hsl` and `hsla` color specifications and the CSS `opacity` property

1. Open `ClassFiles/colors-gradients-background-images-masks/Exercises/hsla-opacity/index.html` in a text editor.
2. Note that the markup for the page is done for you, as is some of the CSS coding.
3. Tile the `#main` `<div>` with the image `pattern_096.gif`
4. Give the `header` and `.photo` elements a background color of `#B7D3D9` with 50% opacity - convert the color to `hsla` using `this tool (http://www.workwithcolor.com/color-converter-01.htm)`.
5. Set each photo's initial `opacity` to 50%. Note that we use `max-width:100%` for the two images, ensuring that the images will scale to smaller sizes if the browser width is reduced or if the page is viewed on a small-screen-size device.
6. Center align all elements.
7. Give the text in the caption a color of 50% black - use `rgba`
8. Use CSS's `:target` pseudo-class to highlight the user's selected element:
    - Set the photo to 100% opacity.
    - Set the background color of the `.photo` `<div>` to 100% opacity.
    - Set the color of the caption text to 100% black.
9. Initially, before the user clicks anywhere, the page should look like this:



10. Clicking on an image (the left one, in this example) should present the page like this:



Solution:
colors-gradients-background-images-masks/Solutions/hsla-opacity/index.html

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
 <title>CSS3 HSL &amp;amp; Alpha</title>
 <meta name="viewport" content="width=device-width">
 <link rel="stylesheet" href="../../Shared/reset.css">
 <link rel="stylesheet" href="../../Shared/style.css">
 <style type="text/css">
  #main {
   background-image:url('pattern_096.gif');
   background-repeat:repeat;
  }
  h1 {
   margin-bottom:0.25em;
   text-align: center;
  }
  header {
   text-align: center;
   background-color:hsla(191, 31%, 78%, 0.5);
  }
  header p {
   padding:2% 2% 0.5% 2%;
  }
  .photo {
   width:44%;
   padding:2% 3%;
   float:left;
   text-align:center;
   background-color:hsla(191, 31%, 78%, 0.5);
  }
  .photo img {
   max-width:100%;
   opacity:0.5;
  }
  .photo .caption {
   color:rgba(0,0,0,0.5);
  }
  div:target {
   background-color:hsla(191, 31%, 78%, 1.0);
  }
  div:target img {
   opacity:1.0;
```

```
    }
    div:target .caption {
     color:rgba(0,0,0,1.0);
    }
    footer {
     text-align: center;
     clear:left;
    }
   </style>
  </head>
  <body>
   <div id="main">
    <h1>Two Photos</h1>
    <header>
     <p>A photo of a <a href="#cat">cat strangely sitting upright</a> and of <a
           href="#chicago">Millenium Park in Chicago</a></p>
    </header>
    <div id="cat" class="photo">
     <a href="#cat"><img src="cat.jpg"></a>
     <div class="caption">This cat seems to enjoy sitting upright</div>
    </div>
    <div id="chicago" class="photo">
     <a href="#chicago"><img src="chicago.jpg"></a>
     <div class="caption"><em>Cloud Gate</em> is a popular public sculpture</div>
    </div>
    <footer>
     <p>I am a footer</p>
    </footer>
   </div>
  </body>
  </html>
```


Code Explanation

We set the specified elements' background color to `hsla(191, 31%, 78%, 0.5)` or `hsla(191, 31%, 78%, 1.0)` - for 50% and 100% opacity, respectively - to match the `#B7D3D9` blueish tone.

We set an initial `opacity:0.5` for the two photos.

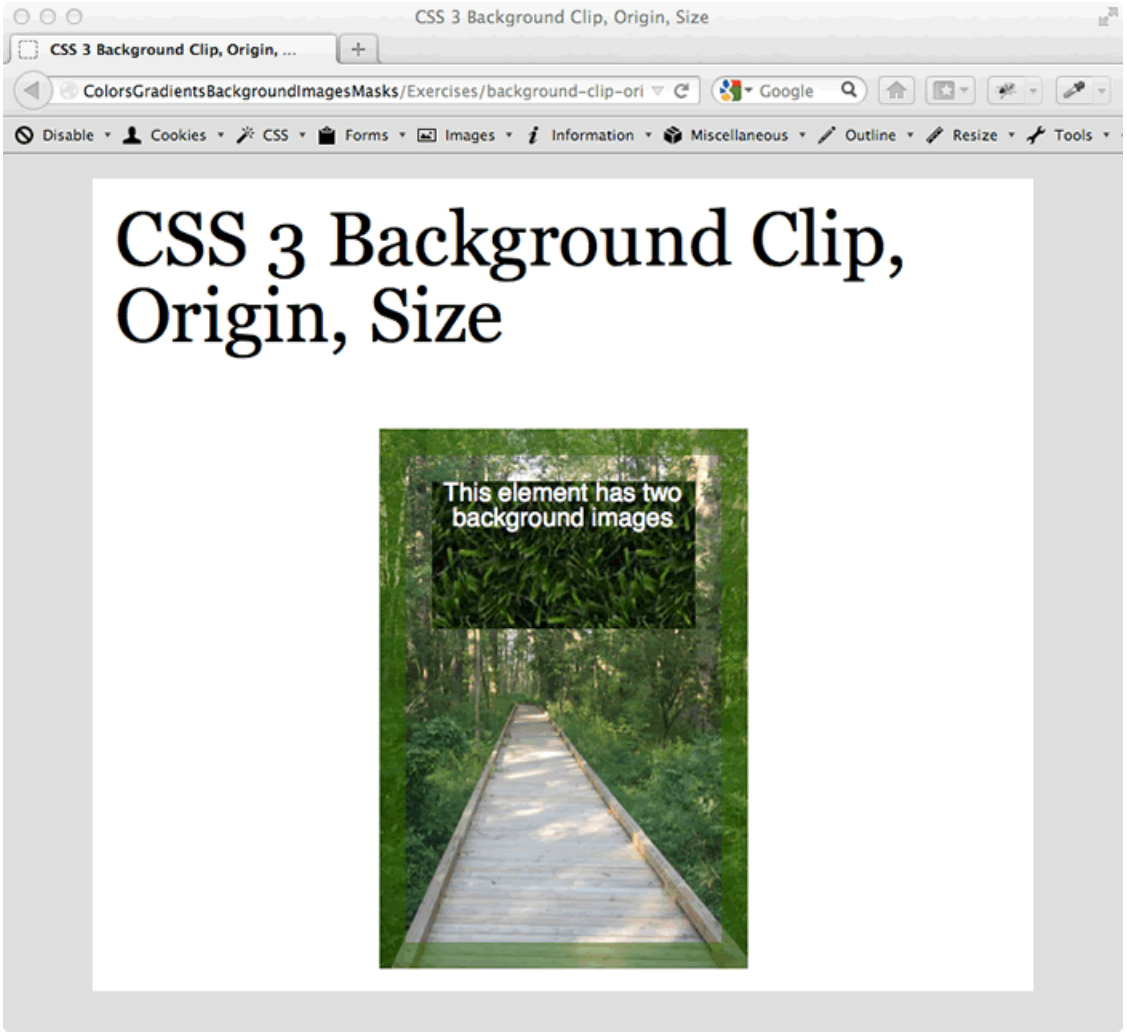
We style the selected photo element via CSS's `:target` pseudo-class, changing the photo's opacity, the opacity of the element's background color, and the opacity of the caption text color.

Exercise 10: Multiple Background Images with background-clip, background-origin, and background-size

10 to 20 minutes

In this exercise, you will

1. Open `ClassFiles/colors-gradients-background-images-masks/Exercises/background-clip-origin-size/index.html` in a text editor and in a browser.
2. Note that the page holds a `<div>` with id `bg`; some of the styles for this `<div>` are already written for you in the `<head>` of the page.
3. Style the `#bg` `<div>` to look like this:



4. Be sure to match the following features:

    A. A 10-pixel-wide, semitransparent, green border; use `hsla` to specify color.

    B. The `boardwalk.jpg` background image fully covers the `<div>`, extending right up to the edge of the border of the element.

    C. A second background image (`grass.jpg`) fill the width (but not the height) of the content area of the `<div>`.

Solution:
colors-gradients-background-images-masks/Solutions/background-clip-origin-size/index.html

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
 <title>CSS3 Background Clip, Origin, Size</title>
 <meta name="viewport" content="width=device-width">
 <link rel="stylesheet" href="../../Shared/reset.css">
 <link rel="stylesheet" href="../../Shared/style.css">
 <style type="text/css">
  #bg {
   margin:auto;
   width:200px;
   height:150px;
   padding:20px 20px 200px 20px;
   border:20px solid hsla(90,100%,20%,0.5);
   background-image:url(grass.jpg), url(boardwalk.jpg);
   background-clip:content-box, border-box;
   background-origin: padding-box, border-box;
   background-size:100% auto, cover;
   background-repeat:no-repeat;
   text-align: center;
   color:#fff;
  }
 </style>
</head>
<body>
 <div id="main">
  <h1>CSS3 Background Clip, Origin, Size</h1>
  <div id="bg">
   This element has two background images
  </div>
 </div>
</body>
</html>
```


Code Explanation

We specify two background images; `grass.jpg` is listed first and, thus, sits on top of `boardwalk.jpg`.

We set the background painting area for each background image with a comma-separated pair of values for `background-clip`. `grass.jpg` gets a value of `content-box` and thus fills

behind the text only; `boardwalk.jpg` gets a value of `border-box` and thus extends under the semitransparent border. Similarly, we set two values for `background-origin` to position the background images as specified.

Last, we set the size of the background images with `background-size`. `grass.jpg` is asked to fill the width of the space allocated to it (with a value of `100% auto`) and `boardwalk.jpg` is asked to `cover` the whole element.

Conclusion

In this lesson, you have learned:

- How to use new CSS Level 3 colors, new ways to specify colors, and the `opacity` property.
- How to use multiple background images.
- How to specify background image properties with `background-clip`, `background-origin`, and `background-size`.

LESSON 5
Borders and Box Effects

Topics Covered

☑ How to use images for borders.

☑ How to use `border-radius` to add rounded corners.

☑ How to use `box-shadow` to add drop shadowing.

Introduction

CSS Level 3 offers us multiple ways to enhance the visual presentation of elements on the page, including image borders, rounded corners, and drop shadows.

5.1. Borders and Box Effects

❖ 5.1.1. Image Borders

New in CSS Level 3 is the opportunity to use images for borders. `border-image` serves as a shorthand for the property; alternately, we can specify some or all of the underlying specific properties; see the `W3C specification (http://www.w3.org/TR/css3-background/#border-images)` for more details:

`border-image` properties

Property	Description	Default
`border-image-source`	Path to the image to be used as border	none
`border-image-slice`	Inward offsets from the top, right, bottom, and left edges of the image	100%
`border-image-width`	Widths of the image border	1
`border-image-outset`	Amount by which the border image area extends beyond the border box	0
`border-image-repeat`	How the images for the sides and the middle part of the border image are scaled and tiled, if at all	`stretch`

A great tool for generating border-image CSS code is `http://www.border-image.com/` (`http://http://www.border-image.com/`). You can upload an image, use sliders to control the numeric properties, and select values for the repeat property - a useful resource.

The `border-image` property is supported by all major browsers, including Internet Explorer starting with version 11.

Let's look at an example that demonstrates some of the properties of image borders in CSS.

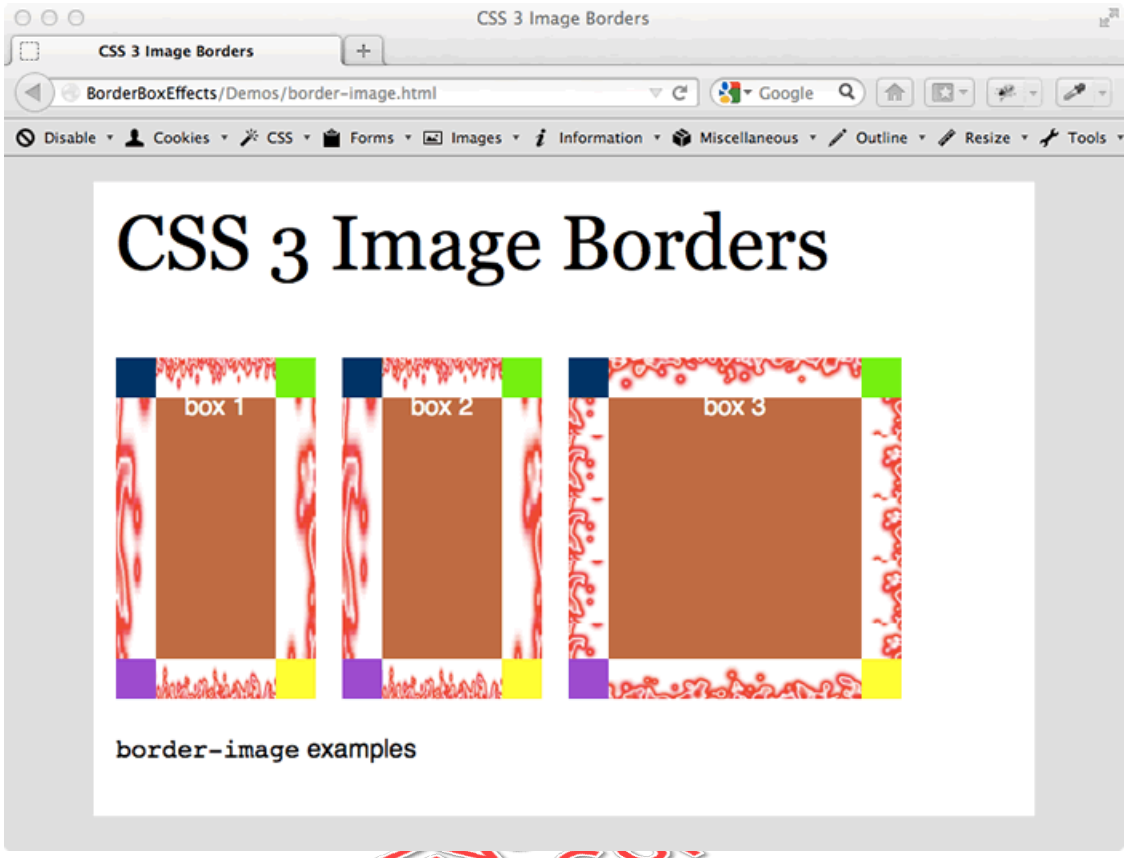
Demo 5.1: border-box-effects/Demos/border-image.html

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
 <title>CSS3 Image Borders</title>
 <meta name="viewport" content="width=device-width">
 <link rel="stylesheet" type="text/css" href="../Shared/reset.css">
 <link rel="stylesheet" type="text/css" href="../Shared/style.css">
 <style type="text/css">
  .border {
   height:200px;
   float:left;
   margin-right:20px;
   background-color:hsl(20, 50%, 50%);
   color:hsl(0,0%,100%);
   text-align:center;
   padding:28px 0;
   border:1px;
  }
  #box1 {
   width:150px;
   border-image-source:url(border.png);
   border-image-slice:28;
   border-image-width:30px 30px 30px 30px;
   border-image-repeat:stretch;
   border-image-outset:20;
  }
  #box2 {
   width:150px;
   border-image:url(border.png)  28 / 30px 30px 30px 30px / 20 stretch;
  }
  #box3 {
   width:250px;
   border-image-source:url(border.png);
   border-image-slice:28;
   border-image-width:30px 30px 30px 30px;
   border-image-repeat:repeat;
   border-image-outset:20;
  }
  footer {
   clear:left;
  }
 </style>
 <!--[if lt IE 9]>
```

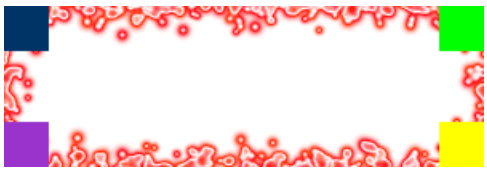
```
45.       <script src="http://html5shiv.googlecode.com/svn/trunk/html5.js"></script>
46.    <![endif]-->
47.   </head>
48.   <body>
49.    <div id="main">
50.     <h1>CSS3 Image Borders</h1>
51.     <div id="box1" class="border">
52.      box 1
53.     </div>
54.     <div id="box2" class="border">
55.      box 2
56.     </div>
57.     <div id="box3" class="border">
58.      box 3
59.     </div>
60.     <footer>
61.      <p><code>border-image</code> examples</p>
62.     </footer>
63.    </div>
64.   </body>
65.   </html>
```


Code Explanation

Open `ClassFiles/border-box-effects/Demos/border-image.html` in a file editor and in a browser to view the page. The page displays three `<div>`s, each with an image border:



Each of the three `<div>`s references the same image for their image borders:



The first ("box 1") and second ("box 2") have exactly the same `border-image` properties; with the first we set each property separately, while with the second we set all of the properties via the shorthand notation. For both we use `stretch`, so that the border image - which is both shorter and wider than the elements whose border it is asked to be - is scaled taller and narrower to fill appropriately the border lengths. Note also that the corners of the border image are not skewed, a product of the fact that we have set the width of the border to the same size as the four colored squares that occupy the four corners of the border image.

The third ("box 3") `<div>` demonstrates the effect of the repeat value for the `border-image-repeat` property. Note that neither the horizontal nor the vertical border image is skewed; the horizontal borders show only a part of the border image (since the

border image is itself wider than `#box3`) and the vertical borders show the border image repeated (since the border image is itself shorter than `#box3`.)

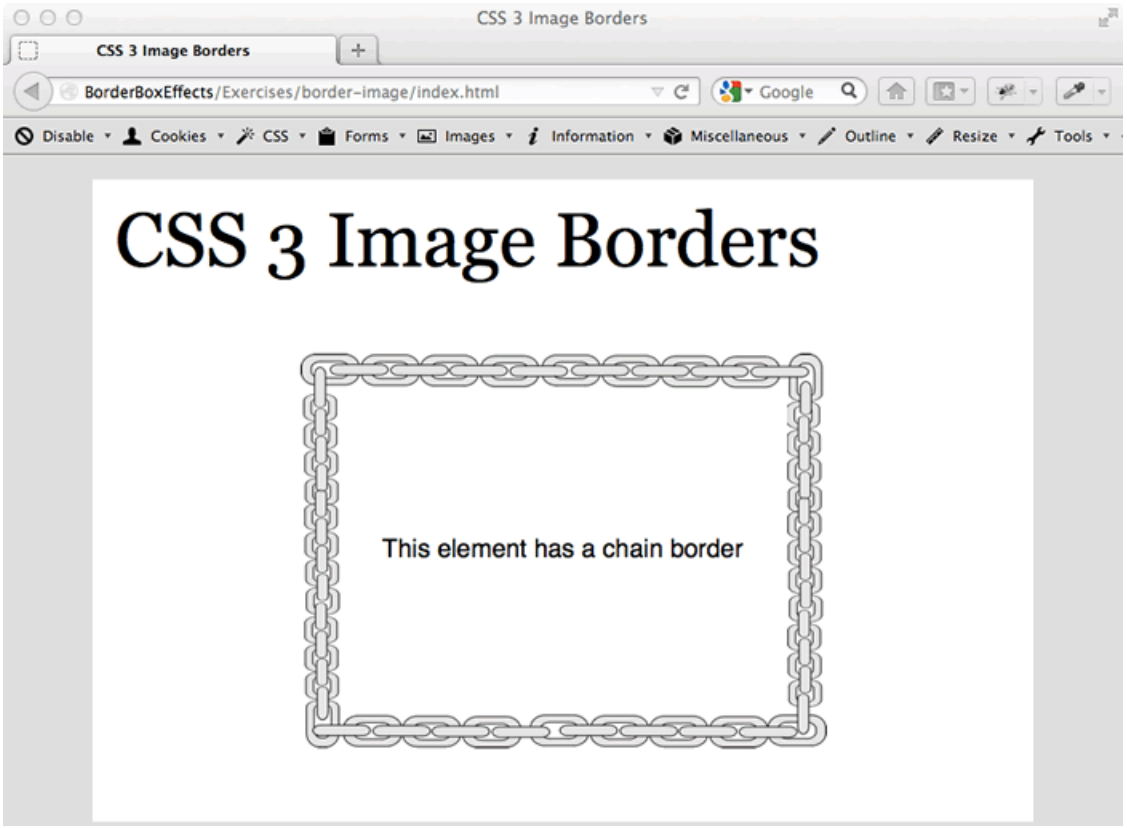
The next exercise asks you to try adding an image border to a simple element.

Exercise 11: Image Borders

10 to 20 minutes

In this exercise, you will add an image border to a simple element.

1. Open `ClassFiles/border-box-effects/Exercises/border-image/index.html` in a file editor and in a browser to view the page.
2. The goal is to use the `chain.png` image as the border for the `#chain` element, to make it look like this:



3. Use the various `border-image-` properties to add the chain border. Experiment with the various `border-image-repeat` values to see which you think looks best.

Solution: border-box-effects/Solutions/border-image/index.html

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
 <title>CSS3 Image Borders</title>
 <meta name="viewport" content="width=device-width">
 <link rel="stylesheet" type="text/css" href="../../Shared/reset.css">
 <link rel="stylesheet" type="text/css" href="../../Shared/style.css">
 <style type="text/css">
  #chain {
   margin:0 auto 2em auto;
   line-height: 300px;
   text-align:center;
   width:400px;
   height:300px;
   border-image-source:url(chain.png);
   border-image-slice:50;
   border-image-width:30px 30px 30px 30px;
   border-image-repeat:stretch;
   border-image-outset:20;
  }
 </style>
 <!--[if lt IE 9]>
     <script src="http://html5shiv.googlecode.com/svn/trunk/html5.js"></script>
 <![endif]-->
</head>
<body>
 <div id="main">
  <h1>CSS3 Image Borders</h1>
  <div id="chain">
   <p>This element has a chain border</p>
  </div>
 </div>
</body>
</html>
```


Code Explanation

We use `border-image-source:url(chain.png)` to specify the image to use as the border. Appropriate values for the other `border-image` properties, including `stretch` for the repeat, align the chain to look as desired.

Exercise 12: Rounded Corners & Drop Shadow

20 to 30 minutes

In this exercise, you will create a CSS speech bubble without using images and add drop shadowing with the `box-shadow` property.

1. Open `ClassFiles/border-box-effects/Exercises/rounded-corners-drop-shadow/index.html` in a text editor and in a browser to view the code.
2. Your job is to make the page look like this:

3. The markup for the page - which is already done for you - is pretty simple: a `<div>` with id `frame` contains a photo image (of an empty chair); a `<div>` with class `bubble` holds the speech bubble. A `caption` paragraph sits at the bottom.
4. Add a gray drop shadow to the `#frame` element with `box-shadow`.
5. Add rounded corners and a background color to the `.bubble` element.
6. To create the triangle (the pointer) for the speech bubble, add code to the `.bubble:after` property. A CSS 2 pseudo-class, `:after` allows us to render content via CSS. Use `content:''` to add no visible content, position the element absolutely, give it a 20-pixel border, color the border the same as the background color for the `#frame` element, and use `border-right-color` with the same color as the `.bubble` element - this will have the effect of adding a triangle - one fourth of the square border around an empty element.

Solution: border-box-effects/Solutions/rounded-corners-drop-shadow/index.html

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
 <title>CSS3 Rounded Corners &amp;amp; Drop Shadow</title>
 <meta name="viewport" content="width=device-width">
 <link rel="stylesheet" type="text/css" href="../../Shared/reset.css">
 <link rel="stylesheet" type="text/css" href="../../Shared/style.css">
 <style type="text/css">
  h1 {
   margin-bottom:0.25em;
  }
  #frame {
   box-shadow:10px 10px 15px hsl(0, 0%, 80%);
   width:80%;
   padding:1em;
   margin:0 auto 2em auto;
   background-color:hsl(0, 0%, 90%);
  }
  img#chair {
   float:left;
  }
  .bubble {
   width:150px;
   padding:20px 20px 15px 20px;
   text-align:center;
   font-size:22px;
   line-height:28px;
   position:relative;
   float:left;
   margin-left:40px;
   border-radius:20px;
   color:hsl(0,50%,100%);
   background-color:hsl(10, 58%, 54%);

  }
  .bubble:after {
   content: '';
   position: absolute;
   width: 0;
   height: 0;
   border: 20px solid;
   border-color:hsl(0, 0%, 90%);
```

```
     border-right-color: hsl(10, 58%, 54%);
     top: 50%;
     right: 100%;
     margin-top: -20px;
    }
    p.caption {
     clear:left;
     padding:1em 0 0 0;
     font-style:italic;
     text-align: center;
    }
   </style>
   <!--[if lt IE 9]>
       <script src="http://html5shiv.googlecode.com/svn/trunk/html5.js"></script>
   <![endif]-->
  </head>
  <body>
   <div id="main">
    <h1>CSS3 Rounded Corners &amp;amp; Drop Shadow</h1>
    <div id="frame">
      <img src="chair.jpg" alt="chair" id="chair">
      <div class="bubble">
      <p>I'm a chair. An empty chair. Ain't nobody sittin' on me.</p>
     </div>
     <p class="caption">Speech bubble rendered with CSS</p>
    </div>

   </div>
  </body>
  </html>
```


Code Explanation

We use `box-shadow:10px 10px 15px hsl(0, 0%, 80%);` to add a drop shadow to the `#frame` element.

We give some background and text color to the `.bubble` element.

The most interesting code is in `.bubble:after`. We add a border, give it a width of 20 pixels, position it absolutely (to sit on the left side of the speech bubble), and color the border - and the right portion of the border, specifically - to effect the small, left-pointing triangle. All without an image!

Conclusion

In this lesson, you have learned:

- How to add image borders.
- How to add rounded corners with `border-radius`.
- How to add drop shadowing with `box-shadow`.

LESSON 6

Transitions, Transforms, and Animations

Topics Covered

☑ How to use `transition` to animate changes to an element's style.

☑ How to use `transform` to move, scale, rotate, and skew elements in two or three dimension.

Introduction

CSS transitions and transforms are powerful tools to create visually interesting, complex animations and effects without the use of JavaScript.

6.1. Transitions & Transforms

❖ 6.1.1. Transitions

A few years ago, we might have turned to JavaScript (and, before that, to Flash) to present users with interactive animations: a delayed change to the background color of a moused-over link, perhaps, or a fade-in effect on the opening or closing of a drop-down menu. While JavaScript and other technologies still have their place, CSS Level 3 now affords us the ability to powerfully - and simply - add motion effects to spice up our pages.

The CSS `transition` property defines how an element changes from one state to another. With `transition` - or the specific properties, listed below, for which `transition` serves as a shorthand - we can stretch out the change on an element's width, background color, or other property over a specified duration. Older versions of browsers would need vendor prefixes for this property: we'd need to be sure that we include appropriate vendor prefixes for transitions: `-moz-` for older version of Firefox, `webkit` for older versions of Chrome/Safari, and `-o-` for older versions of Opera. Internet Explorer supports CSS transitions, unprefixed, from version 10+. In the examples below, we won't use vendor prefixes; all versions of modern browsers now support transitions.

CSS Transition Properties

Property	Description/Example
`transition`	Shorthand for setting all four properties in one statement `transition:width 3s ease 1s`
`transition-property`	Specific property/ies for which transition will occur, or `all` (default) or `none` `transition-property: width, top`
`transition-duration`	Length, in specified time units, of the transition. With duration specified as “0” (or if not specified, and thus defaulting to “0”), no visible effect will take place. `transition-duration: 1s`
`transition-timing-function`	Defines the easing function of the transition - how the speed of the transition will change over the course of the duration. Possible values are `linear`, `ease` (default), `ease-in`, `ease-out`, `ease-in-out`, `cubic-bezier(n,n,n,n)` `transition-timing-function: ease-out`
`transition-delay`	Delay, in specified time units, after which transition will occur; default is 0 `transition-delay: 3s`

Let’s look at an example that illustrates the use of the `transition` property.

Demo 6.1: transforms-transitions-animations/Demos/transitions.html

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
 <title>CSS Transitions</title>
 <meta name="viewport" content="width=device-width">
 <link rel="stylesheet" href="../Shared/reset.css">
 <link rel="stylesheet" href="../Shared/style.css">
 <style type="text/css">
  #trans1 {
   width:160px;
   height:160px;
   padding:20px;
   float:left;
   background-color:hsl(20, 50%, 20%);
   color:hsl(0, 100%, 100%);
   transition:width 2s;
  }
  #trans1:hover {
   width:100px;
   height:100px;
   background-color:hsl(30, 50%, 90%);
   color:hsl(0, 0%, 0%);
  }

  #trans2 {
   margin-left:20px;
   width:400px;
   height:160px;
   padding:20px;
   float:left;
   background-color:hsl(0, 0%, 0%);
   color:hsl(0, 100%, 100%);
   position:relative;
  }
  #positioned {
   width:80px;
   height:80px;
   padding:20px;
   background-color:hsl(0, 68%, 32%);
   font-size:12px;
   position: absolute;
   left:10px;
   transition-duration:2s;
```

```
45.      transition-timing-function:ease-in;
46.      transition-delay:1s;
47.     }
48.     #trans2:hover #positioned {
49.      left:310px;
50.     }
51.     footer {
52.      clear:both;
53.     }
54.    </style>
55.   </head>
56.   <body>
57.    <div id="main">
58.     <h1>CSS Transitions</h1>
59.     <div id="trans1">
60.      <p>Mouseover me - width changes</p>
61.     </div>
62.     <div id="trans2">
63.      <div id="positioned">
64.       <p>I am a positioned</p>
65.      </div>
66.     </div>
67.     <footer>
68.      <p>CSS3 Transitions Examples</p>
69.     </footer>
70.    </div>
71.   </body>
72.   </html>
```

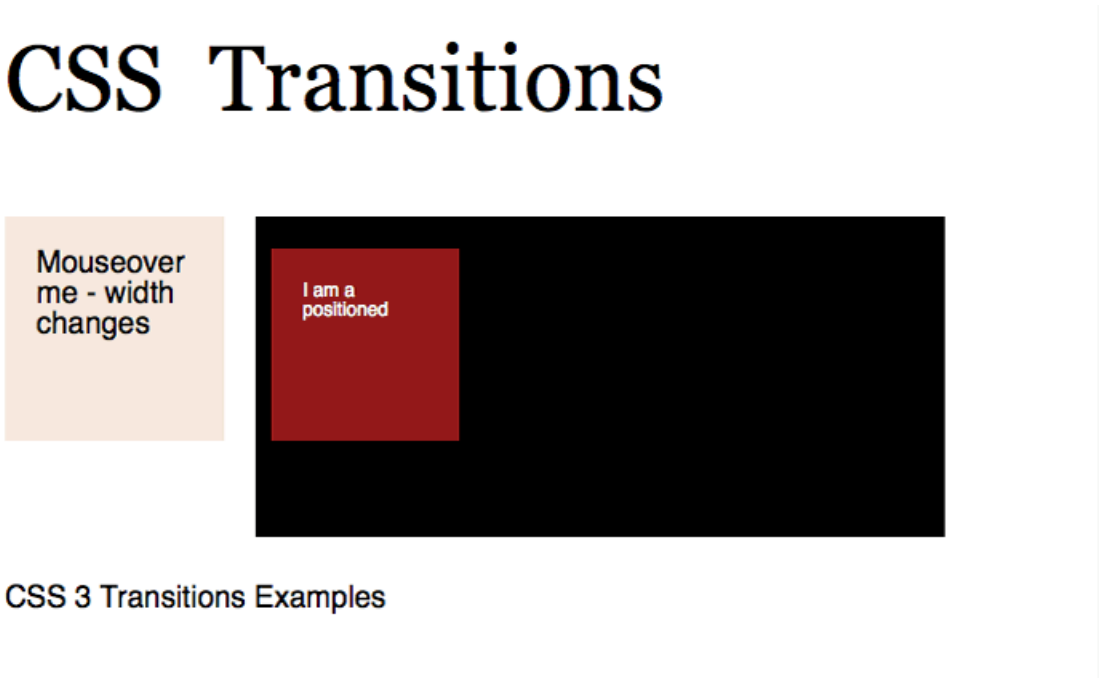

Code Explanation

Open `ClassFiles/transforms-transitions-animations/Demos/transitions.html` in a code editor and a browser to view the code. The page presents two main elements: a brown square on the left with white text “Mouseover me - width changes”, and a black rectangle (on the right) containing a smaller red rectangle; the red rectangle is positioned absolutely, 10 pixels from the left edge of the black rectangle:

CSS Transitions

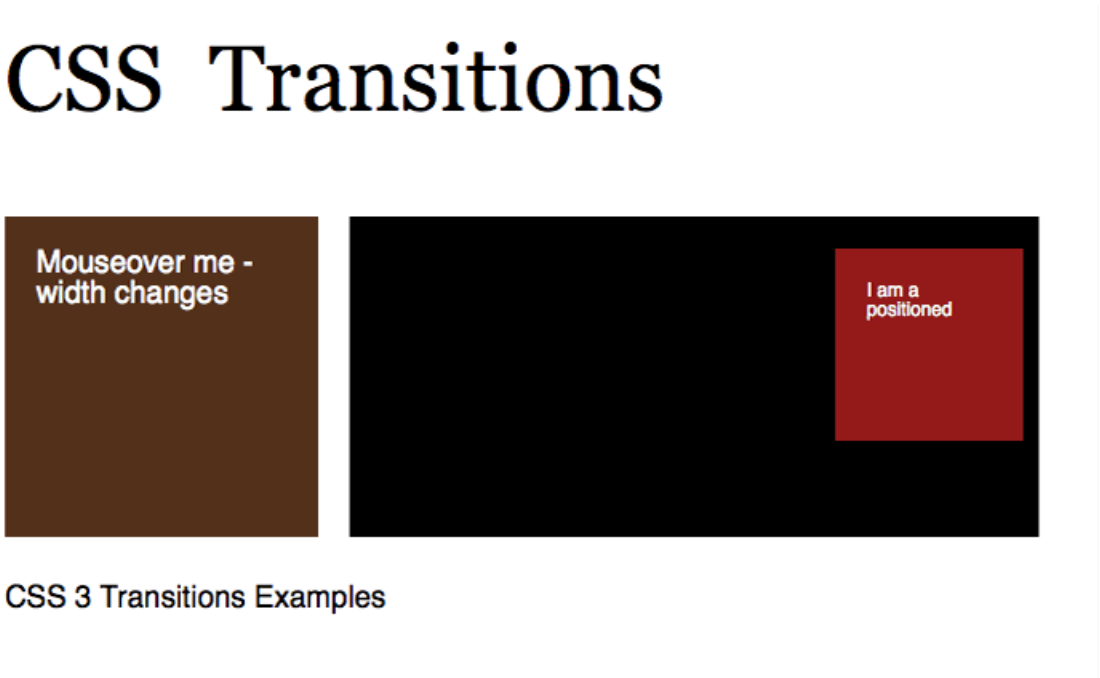
CSS 3 Transitions Examples

We use `#trans1:hover` to change the width, height, background color, and text color of the left (brown) square when the user mouses over the element. Note that the CSS `transition` property is set on `#trans1`, not on `#trans1:hover`. We use the shorthand property here to state that the transition will apply only to the width change - not for the height, background color, or text color changes - and that it will occur over a duration of two seconds. The page looks like this after the transition finishes:



Note that the transition runs in reverse when the user mouses out of the left square: the width changes, over the course of two seconds, back to its original 160 pixel value.

We use the specific `transition` properties (instead of the shorthand) for the right element: mousing over the right (black) containing box changes the contained (`#positioned`) element's `left` value from 10 pixels to 310 pixels. After a `transition-delay` of one second, over a duration of two seconds, and with a timing function `ease-in`, mousing over the right element has the effect of animating the red box - it waits a second to start, begins slowly, then picks up speed. Here's the page after the right transition finishes:

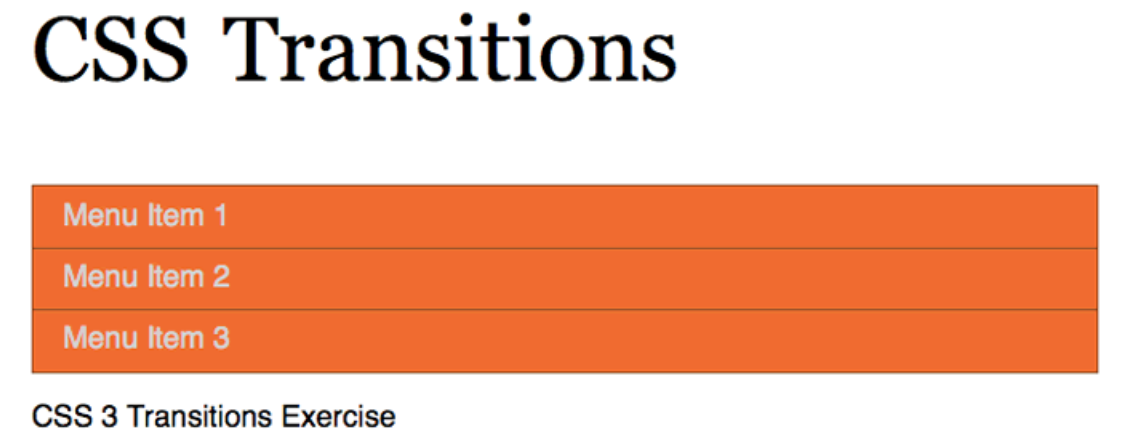


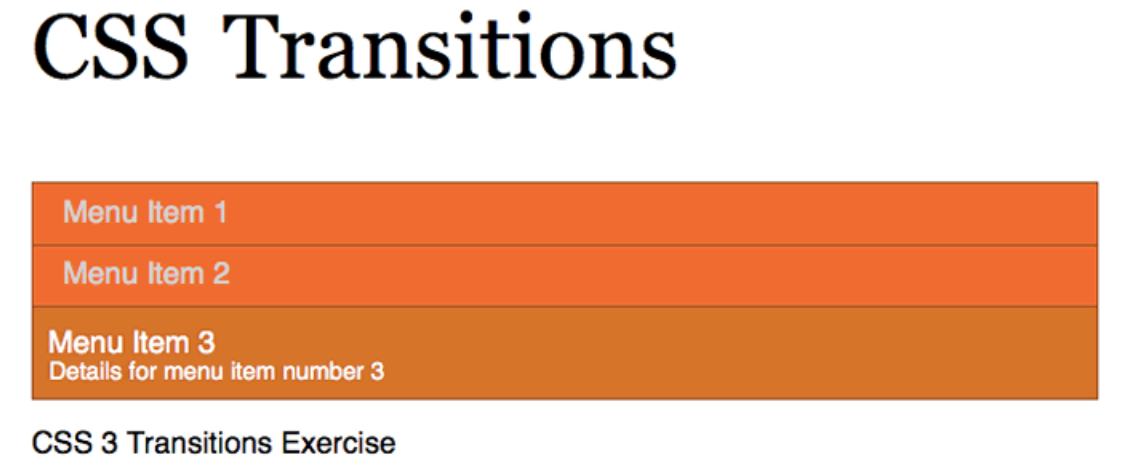
The next exercise asks you to try out transitions.

Exercise 13: Transitions

15 to 25 minutes

In this exercise, you will create a menu using CSS transitions. The screenshot below shows, at top, the menu in its initial state and, at bottom, the menu after a user has moused over the third item. We'll use transitions to add some animation to the changing state of the menu.





1. Open `ClassFiles/transforms-transitions-animations/Exercises/transitions.html` in a code editor and in a browser to view the page.

2. Note that the code that renders the initial state of the menu is done for you.
3. Use `:hover` to change the background color, text color, and padding when the user mouses over each list item.
4. Initially, the `<div>`s that hold the extra info ("Details for menu item number 1") is initially positioned off the screen; use `:hover` to show the extra info by positioning it near the left edge of the list item.
5. Use `transition` to animate the change to the list item when hovered, and to show the extra info flying in/out when hovered.

Solution: transforms-transitions-animations/Solutions/transitions.html

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
 <title>CSS Transitions</title>
 <meta name="viewport" content="width=device-width">
 <link rel="stylesheet" href="../Shared/reset.css">
 <link rel="stylesheet" href="../Shared/style.css">
 <style type="text/css">
  ul#menu {
   list-style:none;
   margin:0 0 20px 0;
   padding:0;
   border-top:1px solid hsl(20, 57%, 35%);
   border-right:1px solid hsl(20, 57%, 35%);
   border-left:1px solid hsl(20, 57%, 35%);
  }
  ul#menu li {
   padding:0.5em 1em;
   background-color:hsl(24, 100%, 50%) ;
   color:hsl(0, 0%, 80%);
   border-bottom:1px solid hsl(20, 57%, 35%);
   margin:0;
   transition-duration:1s;
   position:relative;
   overflow:hidden;
  }
  ul#menu li div {
   position:absolute;
   left:-10000px;
   margin:0;
   font-size:0.8em;
   color:hsl(0, 0%, 100%);
   transition-duration:1s;
  }

  ul#menu li:hover {
   padding:0.75em 0.5em 1.25em 0.5em;
   color:hsl(0, 0%, 100%);
   background-color: hsl(25, 75%, 46%);
  }

  ul#menu li:hover div {
   left:10px;
```

```
45.     }
46.
47.    </style>
48.   </head>
49.   <body>
50.    <div id="main">
51.     <h1>CSS Transitions</h1>
52.     <ul id="menu">
53.      <li>Menu Item 1
54.      <div>Details for menu item number 1</li>
55.      <li>Menu Item 2
56.      <div>Details for menu item number 2</li>
57.      <li>Menu Item 3
58.      <div>Details for menu item number 3</li>
59.     </ul>
60.     <footer>
61.      <p>CSS3 Transitions Exercise</p>
62.     </footer>
63.    </div>
64.   </body>
65.   </html>
```


Code Explanation

We update the background color, text color, and padding for the list items when hovered over (with `:hover`), and set the `left` value of the contained `<div>`s to be 10 pixels when their parent list item is hovered over.

We set a `transition-duration` of one second for both the list items and the `<div>`s contained within them. This presents the user with a soft change over time (rather than immediate) for the list item's color/padding states, and has the effect of flying-in the extra-info text. All without JavaScript!

❖ E13.1. Two Dimensional Transforms

CSS gives us the `transform` property, which allows us to easily position, stretch, scale, and rotate elements. Here's a list of the valid values (functions) for the `transform` property:

2D Transform Methods

Method	Example
`translate`	`translate(10px, -20px)` moves the element 10 pixels left and 20 pixels up from its original position; can use `translateX` or `translateY` for horizontal or vertical positioning
`rotate`	`rotate(20deg)` rotates the element 20 degrees clockwise; negative values rotate counter-clockwise
`scale`	`scale(2,3.5)` stretches the element to be twice as wide and 3.5 times as tall; can also use `scaleX` or `scaleY` to specify horizontal or vertical scaling
`skew`	`skew(10deg, 20deg)` turns the element 10 degrees about the X axis and 20 degrees about the Y axis; can also use `skewX` or `skewY` to control X- and Y-axis skew
`matrix`	`matrix(0.75,0.5,-0.5,1.5,20,0)` allows us to specify all of the above methods at once according to a square matrix - see the `W3C website (http://www.w3.org/TR/css3-transforms/)` for more description of using the `matrix` function (but be sure to dig up a linear algebra textbook for reference!)

Browser support for transforms is pretty good; newer versions of desktop and mobile browser support the property well. Be sure to use vendor prefixes if targeting an audience using older browsers - though the use of an autoprefixer can handle this for you.

We'll examine each of the various `transform` methods in the next example.

Exercise Code 13.1: transforms-transitions-animations/Demos/transforms2d.html

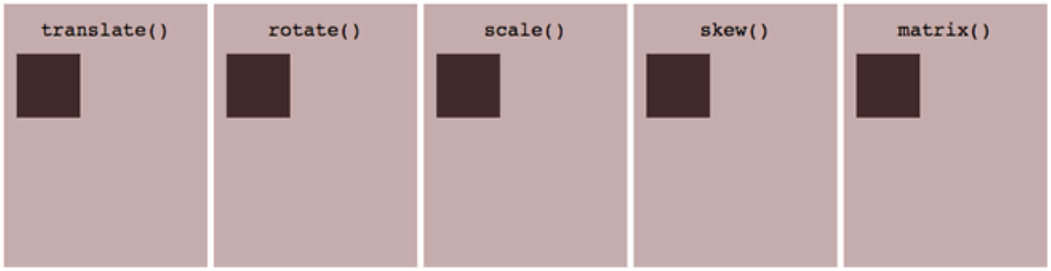
```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
 <title>CSS 2D Transforms</title>
 <meta name="viewport" content="width=device-width">
 <link rel="stylesheet" href="../Shared/reset.css">
 <link rel="stylesheet" href="../Shared/style.css">
 <style type="text/css">
  div.example {
   width:140px;
   height:200px;
   background-color:hsl(0, 20%, 70%);
   margin:0 5px 20px 0;
   padding:3px 10px;
   text-align: center;
   float:left;
  }
  div.example div {
   width:50px;
   height:50px;
   background-color:hsl(0, 20%, 20%);
  }
  div.example code {
   display:block;
   font-size:0.8em;
   padding:10px;
  }
  #translate:hover div {
   transform: translate(50px, 100px);
  }
  #rotate:hover div {
   transform: rotate(20deg);
  }
  #scale:hover div {
   transform: scale(2.5, 3);
  }
  #skew:hover div {
   transform: skew(10deg, 30deg);
  }
  #matrix:hover div {
   transform: matrix(0.75,0.5,-0.5,1.5,20,0);
  }
```

```
44.     footer {
45.      clear:both;
46.     }
47.    </style>
48.   </head>
49.   <body>
50.    <div id="main">
51.     <h1>CSS 2D Transforms</h1>
52.     <div class="example" id="translate"><code>translate()</code><div></div></div>
53.     <div class="example" id="rotate"><code>rotate()</code><div></div></div>
54.     <div class="example" id="scale"><code>scale()</code><div></div></div>
55.     <div class="example" id="skew"><code>skew()</code><div></div></div>
56.     <div class="example" id="matrix"><code>matrix()</code><div></div></div>
57.     <footer>
58.      <p>CSS3 2D Transforms Examples</p>
59.     </footer>
60.    </div>
61.   </body>
62.   </html>
```

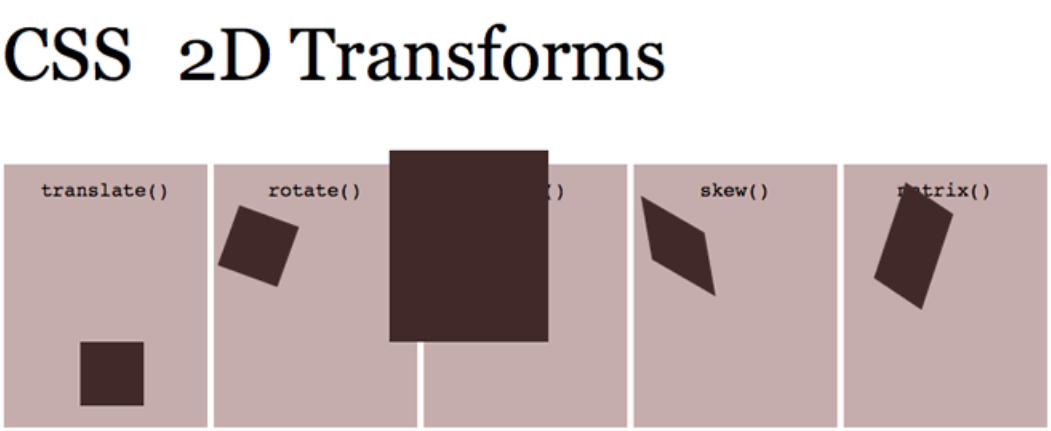

Code Explanation

Open `ClassFiles/transforms-transitions-animations/Demos/transforms2d.html` in a code editor, and in a browser to view the code.

The page displays five `<div>`s, each labeled with a `transform` method and containing a small, darker `<div>`. When moused over, each of the small dark elements displays its respective `transform` method. The screenshot below shows at top the original state of each element; at bottom is the result of each upon application of its respective `transform` method:



CSS 3 2D Transforms Examples



CSS 3 2D Transforms Examples

In the first element, `transform: translate(50px, 100px);` pushes the small box 50 pixels right and 100 pixels down from its original position.

In the second element, `transform: rotate(20deg);` rotates the small box 20 degrees clockwise.

We use `transform: scale(2.5, 3);` to stretch the third box 2.5 times wider and three times taller.

In the fourth example, we skew the small box 10 degrees about the X-axis and 30 degrees about the Y-axis with `transform: skew(10deg, 30deg);`

We combine all of the transforms with `transform: matrix(0.75,0.5,-0.5,1.5,20,0);` in the final example.

By default, the origin - the reference point for translations, rotations, scalings, and skewings - is the center of the element. We can change that via the `transform-origin` property. The property accepts values for the x-axis and y-axis of the origin, respectively, in percentage, absolute length, or keyword forms. Here are some examples:

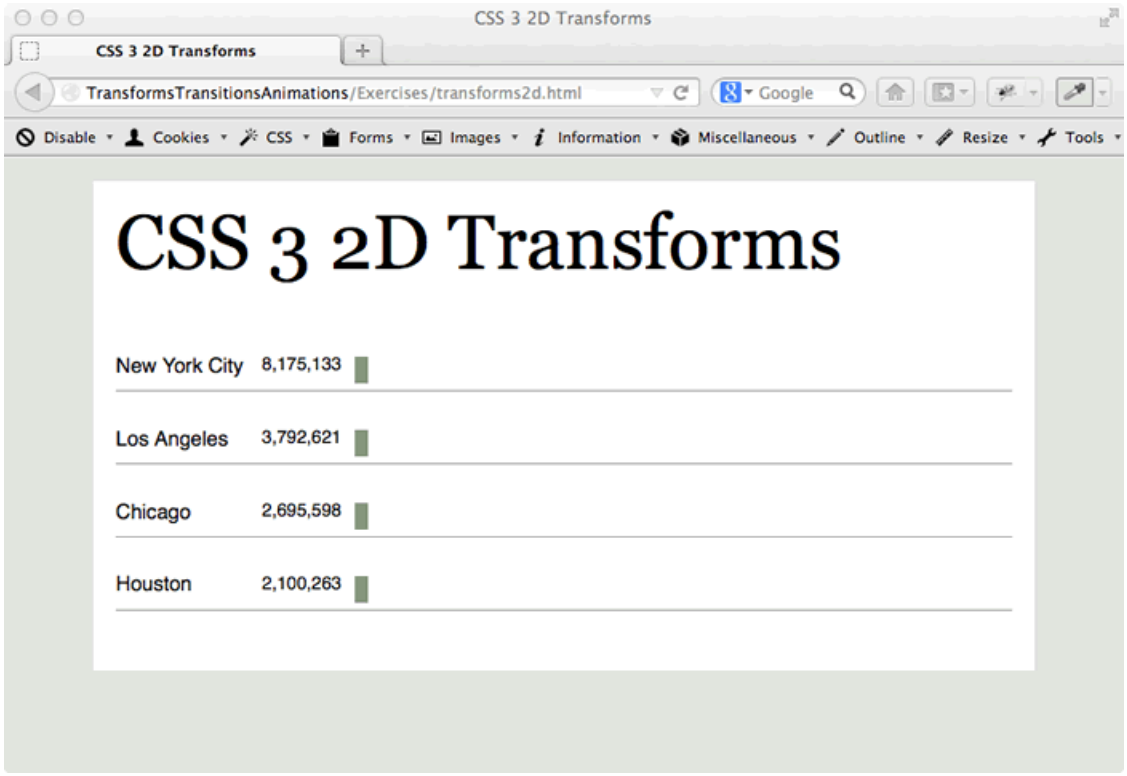
- `transform-origin:20% 50%`
- `transform-origin:left top` (possible x values are `left`, `center`, `right`; possible y values are `top`, `center`, `bottom`)
- `transform-origin:10px 50px`

The next exercise asks you to try out transforms to display a bar graph; you'll need to use the `transform-origin` property.

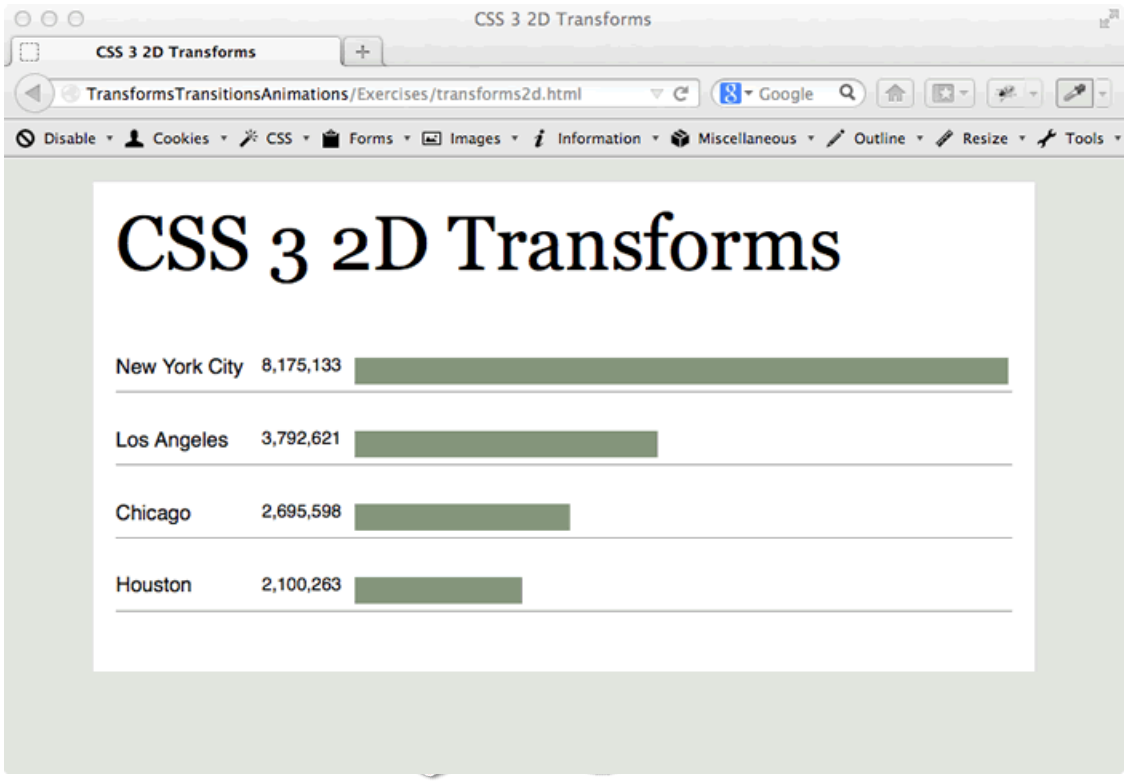
Exercise 14: A Bar Graph

15 to 25 minutes

In this exercise, you will display a bar graph showing the populations of the four most-populous American cities (as of the 2010 census). Initially, the page displays population information for the four cities, as shown:



When the user mouses over the main content, we use a transform to lengthen the bar graphs proportional to each city's population, like this:



1. Open `ClassFiles/transforms-transitions-animations/Exercises/transforms2d.html` in a code editor and in a browser to view the results.
2. Style the city name, population, and bar code to look as shown in the screenshot; I chose to float each element left, but you might style differently.
3. Use #wrapper:hover to effect a transform of each city's respective bar graph's width proportional to its population; you might, for instance, choose to make each bar graph five times the population in millions, with NYC's roughly 8.2 million people equating to a scale of 41, etc.
4. Use `transform-origin` to ensure that the scaling stretches the bar-graph element to the right; without using `transform-origin`, the horizontal stretching would occur in both left and right directions.
5. Add a bit of transition duration, to animate the scaling.

Solution: transforms-transitions-animations/Solutions/transforms2d.html

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
 <title>CSS 2D Transforms</title>
 <meta name="viewport" content="width=device-width">
 <link rel="stylesheet" href="../Shared/reset.css">
 <link rel="stylesheet" href="../Shared/style.css">
 <style type="text/css">
  div.city  {
   height:25px;
   clear:left;
   border-bottom:1px solid hsl(0,0%,60%);
  }
  div.cityname {
   width:110px;
   float:left;
   font-size:0.8em;
  }
  div.population {
   width:70px;
   float:left;
   font-size:0.7em;
  }
  div.bargraph {
   float:left;
   width:10px;
   height:20px;
   background-color:hsl(100, 10%, 50%);
   transition-duration:1s;
   transform-origin:left center;
  }
  #wrapper:hover #ny div.bargraph {
   transform:scale(49.2,1);
  }
  #wrapper:hover #la div.bargraph {
   transform:scale(22.8,1);
  }
  #wrapper:hover #chi div.bargraph {
   transform:scale(16.2,1);
  }
  #wrapper:hover #hous div.bargraph {
   transform:scale(12.6,1);
  }
```

```
45.    </style>
46.   </head>
47.   <body>
48.    <div id="main">
49.     <h1>CSS 2D Transforms</h1>
50.     <div id="wrapper">
51.      <div id="ny" class="city"><div class="cityname">New York City</div> <div
              class="population">8,175,133</div> <div class="bargraph"></div></div>
52.      <div id="la" class="city"><div class="cityname">Los Angeles</div> <div
              class="population">3,792,621</div> <div class="bargraph"></div></div>
53.      <div id="chi" class="city"><div class="cityname">Chicago</div> <div
              class="population">2,695,598</div> <div class="bargraph"></div></div>
54.      <div id="hous" class="city"><div class="cityname">Houston</div> <div
              class="population">2,100,263</div> <div class="bargraph"></div></div>
55.     </div>
56.    </div>
57.   </body>
58.   </html>
```

Code Explanation

We float the city name, population, and bar-graph elements left within each `div.city` element, with the bar-graph element initially a `10px` by `20px` rectangle; each `div.city` clear the floating above it.

We use `transform:scale(n,1)` to stretch each bar graph when the main area is hovered over. `n` is a scaling factor proportional to each city's population (roughly six times the population in millions), making the element wider. We use `1` for the vertical scaling, so that the element does not change height.

The statement `transform-origin:left center` ensures that the scaling stretches each bar graph to the right, rather than in both horizontal directions.

Last, `transition-duration:1s` gives some animation to the transform, stretching each bar graph to the right over the course of one second.

❖ E14.1. Three-Dimensional Transforms

CSS also offers transforms that format elements in three-dimensional space. (Note that Internet Explorer prior to version 10 does not support 3D transforms well.) Similar to (but decidedly cooler than) the two-dimensional transforms we looked at above, the 3D

transforms allow us to translate, scale, rotate, and skew elements along the Z (as well as X and Y) axis. Before we look at the 3D-specific transform methods, we'll discuss a few properties that we'll need now that we are working in three-dimensional space.

The `perspective` property, with values in absolute length (e.g., `800px`), defines how far an element is placed from the view perspective. Defined on the parent element and applied to children elements, `perspective` adds a feeling of depth to the 3D transforms; we'll look at some examples below.

By default, children elements do not inherit the `perspective` of their parent's parents; grandchildren elements are "flattened" by default. We can use the `transform-style` property, with a value of `preserve-3d`, to preserve the 3D rendering context it receives from its parent. Again, the example below will make this more clear.

The `backface-visibility` property determines if an element's, ahem, backside should be visible when rotated far enough so as to be not facing the screen. The default value is `visible`; a value of `hidden` hides the element.

In three dimensions, we can now translate along the Z axis - into or out of the screen - in addition to the X and Y directions.

The 3D transform methods are very similar to the 2D method, and best illustrated through a series of examples.

Exercise Code 14.1: transforms-transitions-animations/Demos/transforms3d.html

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
 <title>CSS 3D Transforms</title>
 <meta name="viewport" content="width=device-width">
 <link rel="stylesheet" href="../Shared/reset.css">
 <link rel="stylesheet" href="../Shared/style.css">
 <style type="text/css">
  div.example {
   width:220px;
   height:220px;
   background-color:hsl(0, 0%, 72%);
   margin:0 5px 20px 0;
   padding:3px 10px;
   text-align: center;
   float:left;
  }
  div.example code {
   display:block;
   font-size:0.7em;
   padding:10px;
  }

  #rotateX:hover img {
   transform: rotateX(60deg);
  }
  #rotateY:hover img {
   transform: rotateY(60deg);
  }
  #rotateXperspective {
   perspective:800px;
  }
  #rotateXperspective:hover img {
   transform: rotateX(80deg);
  }
  #rotateYperspective {
   perspective:800px;
  }
  #rotateYperspective:hover img {
   transform: rotateY(60deg);
  }

```

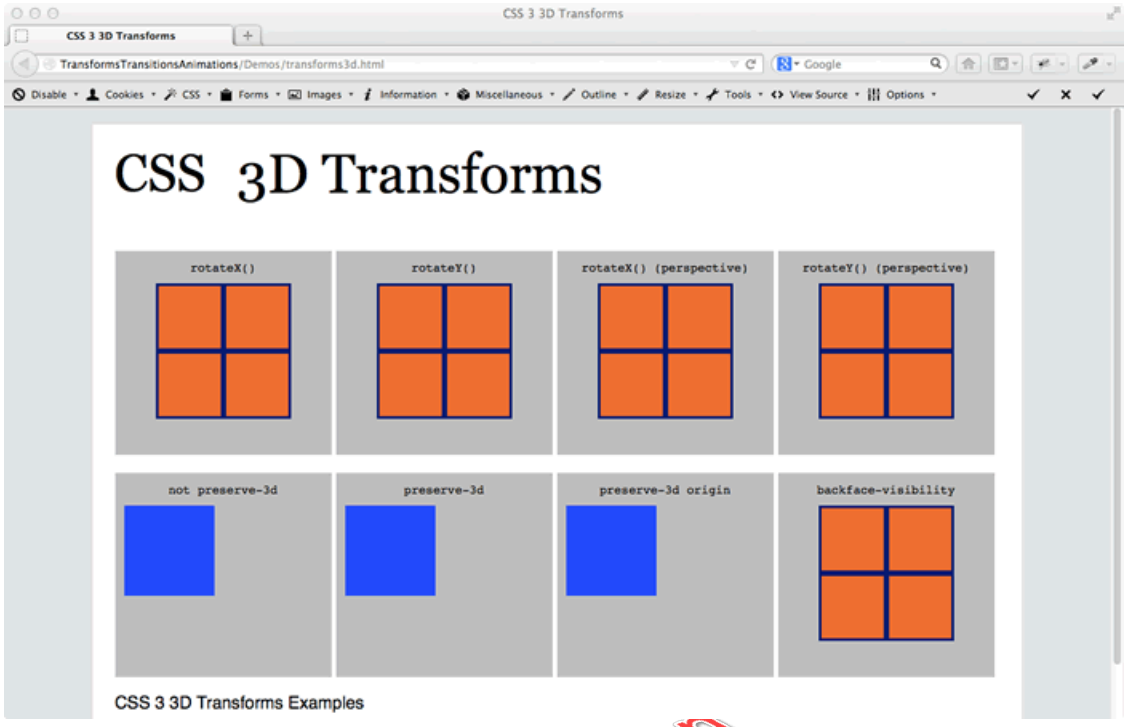
```
44.     #notpreserve3D {
45.      perspective:800px;
46.     }
47.     #notpreserve3D div#parent {
48.      background-color:red;
49.      width:100px;
50.      height:100px;
51.     }
52.     #notpreserve3D div#child {
53.      background-color:blue;
54.      width:100px;
55.      height:100px;
56.     }
57.     #notpreserve3D:hover div#parent {
58.      transform: rotateY(60deg);
59.     }
60.     #notpreserve3D:hover div#child {
61.      transform: rotateX(70deg);
62.     }
63.
64.     #preserve3D {
65.      perspective:800px;
66.     }
67.     #preserve3D div#parent {
68.      transform-style:preserve-3d;
69.      background-color:red;
70.      width:100px;
71.      height:100px;
72.     }
73.     #preserve3D div#child {
74.      background-color:blue;
75.      width:100px;
76.      height:100px;
77.     }
78.     #preserve3D:hover div#parent {
79.      transform: rotateY(60deg);
80.     }
81.     #preserve3D:hover div#child {
82.      transform: rotateX(70deg);
83.     }
84.
85.     #preserve3Dorigin {
86.      perspective:800px;
87.     }
88.     #preserve3Dorigin div#parent {
```

```
     transform-style:preserve-3d;
     background-color:red;
     width:100px;
     height:100px;
    }
    #preserve3Dorigin div#child {
     background-color:blue;
     width:100px;
     height:100px;
    }
    #preserve3Dorigin:hover div#parent {
     transform: rotateY(60deg);
    }
    #preserve3Dorigin:hover div#child {
     transform-origin: top left;
     transform: rotateX(70deg);
    }

    #backface:hover img {
     backface-visibility:hidden;
     transform: rotateX(190deg);
    }

    footer {
     clear:both;
    }
   </style>
  </head>
  <body>
   <div id="main">
    <h1>CSS 3D Transforms</h1>
    <div class="example" id="rotateX"><code>rotateX()</code><img src="block.gif"
          alt="block square"></div>
    <div class="example" id="rotateY"><code>rotateY()</code><img src="block.gif"
          alt="block square"></div>
    <div class="example" id="rotateXperspective"><code>rotateX() (perspec ↵↵
          tive)</code><img src="block.gif" alt="block square"></div>
    <div class="example" id="rotateYperspective"><code>rotateY() (perspec ↵↵
          tive)</code><img src="block.gif" alt="block square"></div>
    <div class="example" id="notpreserve3D"><code>not preserve-3d</code>
     <div id="parent">
      <div id="child"></div>
     </div>
    </div>
    <div class="example" id="preserve3D"><code>preserve-3d</code>
     <div id="parent">
```

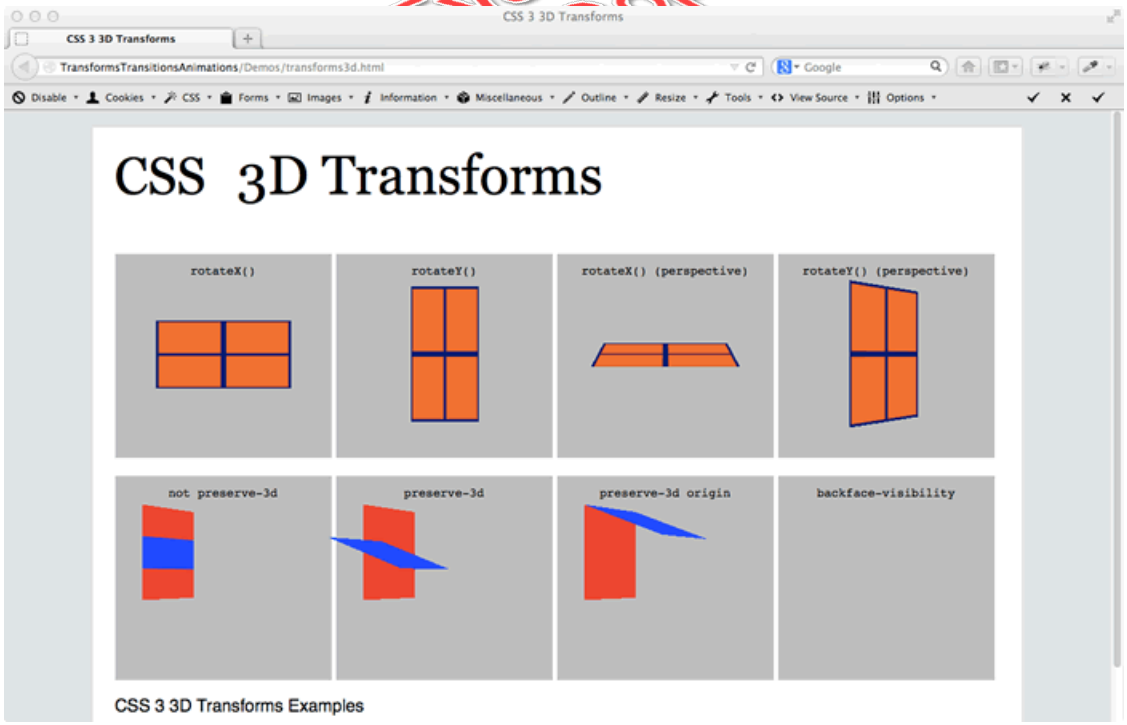
```
131.      <div id="child"></div>
132.     </div>
133.    </div>
134.    <div class="example" id="preserve3Dorigin"><code>preserve-3d origin</code>
135.     <div id="parent">
136.      <div id="child"></div>
137.     </div>
138.    </div>
139.    <div class="example" id="backface"><code>backface-visibility</code><img
              src="block.gif" alt="block square"></div>
140.    <footer>
141.     <p>CSS3 3D Transforms Examples</p>
142.    </footer>
143.   </div>
144.  </body>
145.  </html>
```


Code Explanation

Open `ClassFiles/transforms-transitions-animations/Demos/transforms3d.html` in a code editor and in a browser to view the code. The page presents eight examples; the first four and last (eighth) show a square image; the fifth through seventh show a blue `<div>` containing a red `<div>`. Before mousing over, the page looks like this:



And here's the page after each element is hovered over:



The first two (“rotateX()”, “rotateY()”) elements rotate their image about the X and Y axis, respectively. In the absence of the `perspective` property, we see a flattened view of the rotation. Compare this to the next two examples (“rotateX() (perspective)” and “rotateY() (perspective)”), where we set `perspective:800px`; note that the rotation now appears three dimensional.

The next two examples show the difference when `transform-style:preserve-3d` is used. In both “not preserve-3d” and “preserve-3d”, the outermost element (the gray box) has `perspective:800px`; note that the rotation of the red box (60 degrees about the Y axis) appears three dimensional. But note the difference: in “preserve-3d” we set `transform-style:preserve-3d` on the `#parent` element whereas in “not preserve-3d” we do not. Thus the blue box in “not preserve-3d” does rotate in three dimensions but is “flattened” - that is, it doesn’t inherit the perspective of its parent (the red box). In “preserve-3d”, the blue box is also rotated, but now it does (because of `transform-style:preserve-3d`) inherit its parent’s 3D rendering context.

The seventh (“preserve-3d origin”) example is the same as the example (“preserve-3d”) just before it, but in the case of “preserve-3d origin” we set `transform-origin: top left` for the blue (`#child`) box, thus rotating it about an X axis that is defined at the top of its parent.

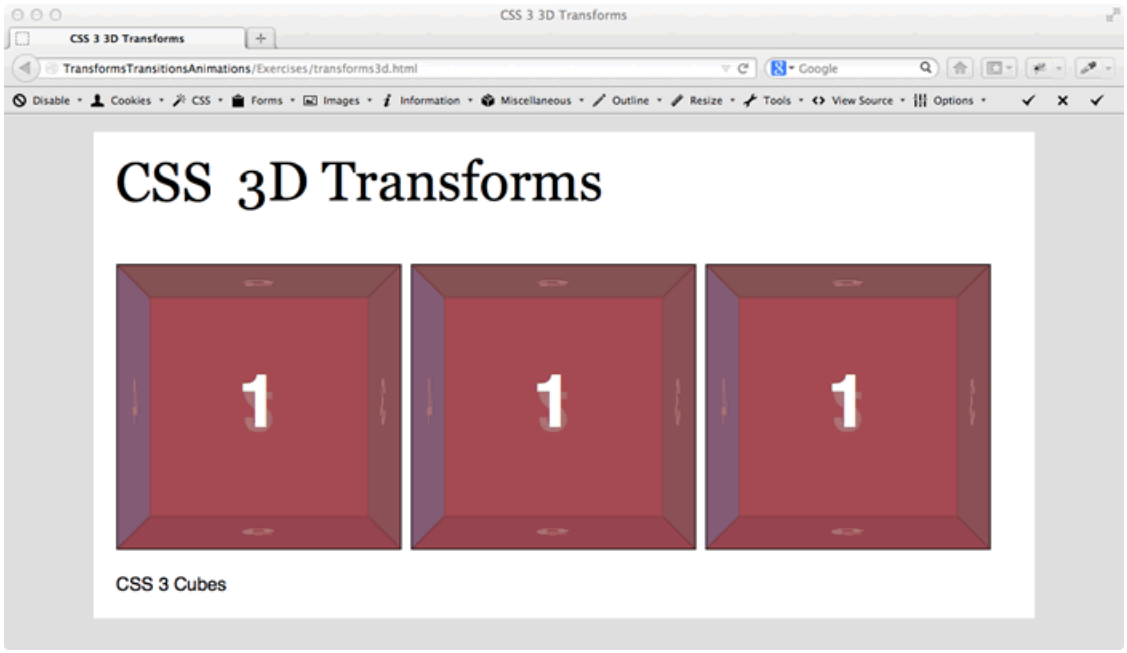
Last, the “backface-visibility” example shows the result of hiding the backface of an element when rotated away from the screen.

The next exercise walks you through rendering one of the most interesting 3D transform possibilities, a rotating cube.

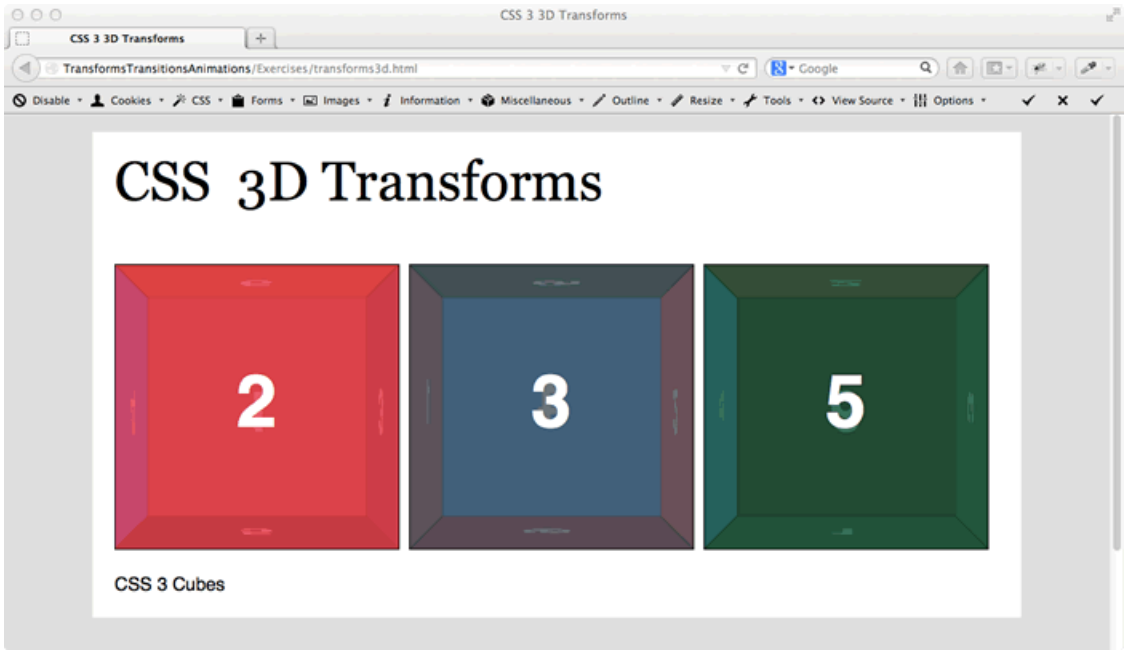
Exercise 15: A 3D Cube

25 to 35 minutes

In this exercise, you will render several three-dimensional cubes, animating the cube to show various faces on hover. Here's a view of the page "at rest":



Each of the three cubes has a semitransparent face, through which you can see the other faces. On mousing over each of the cubes, we get this view:



This is a tough one - feel free to glance at the solution for help.

1. Open `ClassFiles/transforms-transitions-animations/Exercises/transforms3d.html` in a code editor and in a browser to view the page.
2. Add 1000 pixels of perspective to the container element.
3. Use `transform-style:preserve-3d` on the `.cube` element, to ensure the children elements inherit the 3D context.
4. Translate the `.cube` element in the Z direction by -150 pixels.
5. Translate each of the cube faces (`.front`, `.right`, etc.) in the Z direction by 150 pixels.
6. Rotate each of the cube faces appropriately about the X or Y axis. The `.right` face will rotate positive 90 degrees, for instance, about the Y axis, while the bottom face will rotate negative 90 degrees about the X axis.
7. Define the states for `#cube1.cube`, `#cube2.cube`, and `#cube3.cube` on hovering over each element, to show other faces when the user mouses over: translate each item -150 pixels in the Z direction, and apply the opposite rotation that the desired face has before it is moused over.
8. Add a transition to the `.cube` elements, so that the user can see the cube rotate to show the various faces when moused over.
9. Be sure to use vendor prefixes and to test your work in a browser.

Solution: transforms-transitions-animations/Solutions/transforms3d.html

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
 <title>CSS 3D Transforms</title>
 <meta name="viewport" content="width=device-width">
 <link rel="stylesheet" href="../Shared/reset.css">
 <link rel="stylesheet" href="../Shared/style.css">
 <style type="text/css">
  .container {
   width:300px;
   height:300px;
   margin-right:10px;
   position:relative;
   perspective:1000px;
   float:left;
  }
  #cube1:hover .cube {
   /* show back face */
   transform: translateZ(-150px) rotateX(-180deg);
  }
  #cube2:hover .cube {
   /* show right face */
   transform: translateZ(-150px) rotateY(-90deg);
  }
  #cube3:hover .cube {
   /* show bottom face */
   transform: translateZ(-150px) rotateX(-90deg);
  }
  .cube {
   transition-duration:1s;
   position: absolute;
   width: 100%;
   height: 100%;
   transform-style:preserve-3d;
   transform: translateZ(-150px);
  }
  .cube div {
   width: 298px;
   height: 298px;
   display: block;
   position: absolute;
   border: 1px solid black;
   line-height: 298px;
```

```
45.           font-size: 4em;
46.           font-weight: bold;
47.           text-align: center;
48.           color:white;
49.     }
50.     .cube .front {
51.      background-color:hsla(355, 37%, 39%, 0.8);
52.      transform: rotateY(0deg) translateZ(150px);
53.     }
54.     .cube .back {
55.      background-color:hsla(355, 77%, 52%, 0.8);
56.      transform: rotateX(180deg) translateZ(150px);
57.     }
58.     .cube .right {
59.      background-color:hsla(206, 12%, 26%, 0.8);
60.      transform: rotateY(90deg) translateZ(150px);
61.     }
62.     .cube .left {
63.      background-color:hsla(210, 100%, 50%, 0.8);
64.      transform: rotateY(-90deg) translateZ(150px);
65.     }
66.     .cube .top {
67.      background-color:hsla(154, 100%, 13%, 0.8);
68.      transform: rotateX(90deg) translateZ(150px);
69.     }
70.     .cube .bottom {
71.      background-color:hsla(345, 100%, 25%, 0.8);
72.      transform: rotateX(-90deg) translateZ(150px);
73.     }
74.     footer {
75.      clear:both;
76.     }
77.
78.    </style>
79.   </head>
80.   <body>
81.    <div id="main">
82.     <h1>CSS 3D Transforms</h1>
83.     <div class="container" id="cube1">
84.      <div class="cube">
85.       <div class="front">1</div>
86.       <div class="back">2</div>
87.       <div class="right">3</div>
88.       <div class="left">4</div>
89.       <div class="top">5</div>
```

```
      <div class="bottom">6</div>
     </div>
    </div>
    <div class="container" id="cube2">
     <div class="cube">
      <div class="front">1</div>
      <div class="back">2</div>
      <div class="right">3</div>
      <div class="left">4</div>
      <div class="top">5</div>
      <div class="bottom">6</div>
     </div>
    </div>
    <div class="container" id="cube3">
     <div class="cube">
      <div class="front">1</div>
      <div class="back">2</div>
      <div class="right">3</div>
      <div class="left">4</div>
      <div class="top">5</div>
      <div class="bottom">6</div>
     </div>
    </div>
    <footer>
     CSS3 Cubes
    </footer>
   </div>
  </body>
 </html>
```


Code Explanation

We add 1000 pixels of `perspective` to the `.container` element; without this, we would not see the three-dimensional view of the cubes' face.

We apply `transform-style:preserve-3d` to `.cube`, to ensure that children elements do not flatten. We also translate the element -150 pixels in the Z direction.

For each of the cube faces, we translate 150 pixels in the Z direction, and rotate appropriately:

- `.front` rotates 0 degrees
- `.back` rotates 180 degrees

- `.right` rotates 90 degrees about the Y axis
- `.left` rotates -90 degrees about the Y axis
- `.top` rotates 90 degrees about the X axis
- `.bottom` rotates -90 degrees about the X axis

We use the `:hover` pseudo-class to target each of the three cubes, to show the desired face when the parent container is moused over. Each element is translated -150 pixels in the Z direction. To show a specific face, we apply the opposite rotation for the cube as a whole that we apply to the specific face:

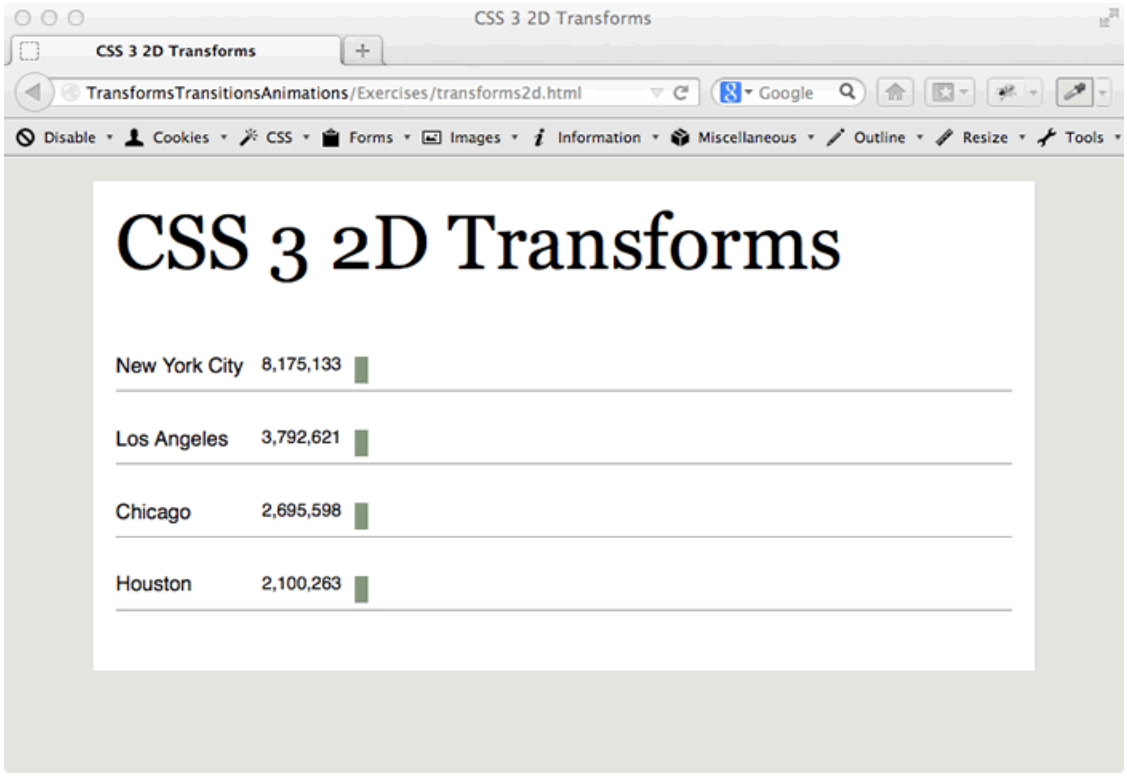
- To show the `.back` face on hover for `#cube1`, we rotate the whole cube -180 degrees about the X axis.
- To show the `.right` face on hover for `#cube2`, we rotate the whole cube -90 degrees about the Y axis.
- To show the `.bottom` face on hover for `#cube3`, we rotate the whole cube -90 degrees about the X axis.

Perhaps the most visually stunning effect comes from a simple CSS rule: adding a `transition-duration` of one second to `.cube` shows the cubes rotating into their new orientations when moused over.

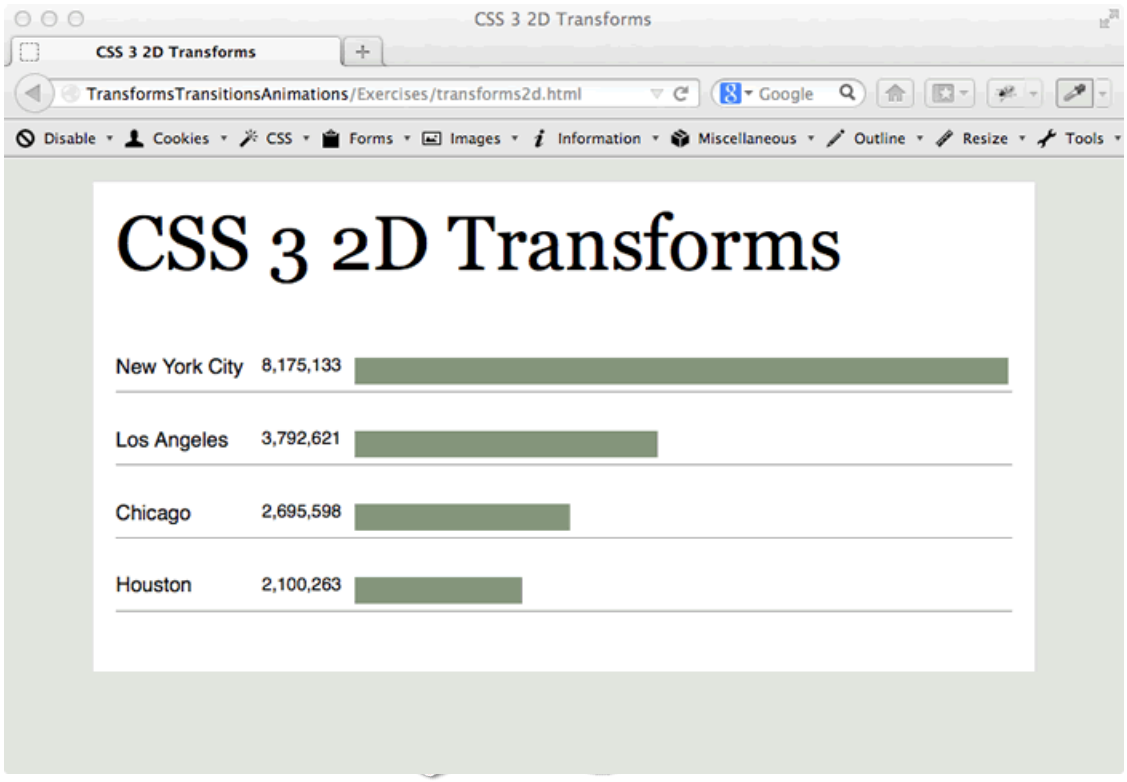
Exercise 16: A Bar Graph

15 to 25 minutes

In this exercise, you will display a bar graph showing the populations of the four most-populous American cities (as of the 2010 census). Initially, the page displays population information for the four cities, as shown:



When the user mouses over the main content, we use a transform to lengthen the bar graphs proportional to each city's population, like this:



1. Open `ClassFiles/transforms-transitions-animations/Exercises/transforms2d.html` in a code editor and in a browser to view the results.
2. Style the city name, population, and bar code to look as shown in the screenshot; I chose to float each element left, but you might style differently.
3. Use #wrapper:hover to effect a transform of each city's respective bar graph's width proportional to its population; you might, for instance, choose to make each bar graph five times the population in millions, with NYC's roughly 8.2 million people equating to a scale of 41, etc.
4. Use `transform-origin` to ensure that the scaling stretches the bar-graph element to the right; without using `transform-origin`, the horizontal stretching would occur in both left and right directions.
5. Add a bit of transition duration, to animate the scaling.

Solution: transforms-transitions-animations/Solutions/transforms2d.html

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
 <title>CSS 2D Transforms</title>
 <meta name="viewport" content="width=device-width">
 <link rel="stylesheet" href="../Shared/reset.css">
 <link rel="stylesheet" href="../Shared/style.css">
 <style type="text/css">
  div.city  {
   height:25px;
   clear:left;
   border-bottom:1px solid hsl(0,0%,60%);
  }
  div.cityname {
   width:110px;
   float:left;
   font-size:0.8em;
  }
  div.population {
   width:70px;
   float:left;
   font-size:0.7em;
  }
  div.bargraph {
   float:left;
   width:10px;
   height:20px;
   background-color:hsl(100, 10%, 50%);
   transition-duration:1s;
   transform-origin:left center;
  }
  #wrapper:hover #ny div.bargraph {
   transform:scale(49.2,1);
  }
  #wrapper:hover #la div.bargraph {
   transform:scale(22.8,1);
  }
  #wrapper:hover #chi div.bargraph {
   transform:scale(16.2,1);
  }
  #wrapper:hover #hous div.bargraph {
   transform:scale(12.6,1);
  }
```

```
45.    </style>
46.   </head>
47.   <body>
48.    <div id="main">
49.     <h1>CSS 2D Transforms</h1>
50.     <div id="wrapper">
51.      <div id="ny" class="city"><div class="cityname">New York City</div> <div
              class="population">8,175,133</div> <div class="bargraph"></div></div>
52.      <div id="la" class="city"><div class="cityname">Los Angeles</div> <div
              class="population">3,792,621</div> <div class="bargraph"></div></div>
53.      <div id="chi" class="city"><div class="cityname">Chicago</div> <div
              class="population">2,695,598</div> <div class="bargraph"></div></div>
54.      <div id="hous" class="city"><div class="cityname">Houston</div> <div
              class="population">2,100,263</div> <div class="bargraph"></div></div>
55.     </div>
56.    </div>
57.   </body>
58.   </html>
```

Code Explanation

We float the city name, population, and bar-graph elements left within each `div.city` element, with the bar-graph element initially a `10px` by `20px` rectangle; each `div.city` clear the floating above it.

We use `transform:scale(n,1)` to stretch each bar graph when the main area is hovered over. `n` is a scaling factor proportional to each city's population (roughly six times the population in millions), making the element wider. We use `1` for the vertical scaling, so that the element does not change height.

The statement `transform-origin:left center` ensures that the scaling stretches each bar graph to the right, rather than in both horizontal directions.

Last, `transition-duration:1s` gives some animation to the transform, stretching each bar graph to the right over the course of one second.

❖ E16.1. Three-Dimensional Transforms

CSS also offers transforms that format elements in three-dimensional space. (Note that Internet Explorer prior to version 10 does not support 3D transforms well.) Similar to (but decidedly cooler than) the two-dimensional transforms we looked at above, the 3D

transforms allow us to translate, scale, rotate, and skew elements along the Z (as well as X and Y) axis. Before we look at the 3D-specific transform methods, we'll discuss a few properties that we'll need now that we are working in three-dimensional space.

The `perspective` property, with values in absolute length (e.g., `800px`), defines how far an element is placed from the view perspective. Defined on the parent element and applied to children elements, `perspective` adds a feeling of depth to the 3D transforms; we'll look at some examples below.

By default, children elements do not inherit the `perspective` of their parent's parents; grandchildren elements are "flattened" by default. We can use the `transform-style` property, with a value of `preserve-3d`, to preserve the 3D rendering context it receives from its parent. Again, the example below will make this more clear.

The `backface-visibility` property determines if an element's, ahem, backside should be visible when rotated far enough so as to be not facing the screen. The default value is `visible`; a value of `hidden` hides the element.

In three dimensions, we can now translate along the Z axis - into or out of the screen - in addition to the X and Y directions.

The 3D transform methods are very similar to the 2D method, and best illustrated through a series of examples.

Exercise Code 16.1: transforms-transitions-animations/Demos/transforms3d.html

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
 <title>CSS 3D Transforms</title>
 <meta name="viewport" content="width=device-width">
 <link rel="stylesheet" href="../Shared/reset.css">
 <link rel="stylesheet" href="../Shared/style.css">
 <style type="text/css">
  div.example {
   width:220px;
   height:220px;
   background-color:hsl(0, 0%, 72%);
   margin:0 5px 20px 0;
   padding:3px 10px;
   text-align: center;
   float:left;
  }
  div.example code {
   display:block;
   font-size:0.7em;
   padding:10px;
  }

  #rotateX:hover img {
   transform: rotateX(60deg);
  }
  #rotateY:hover img {
   transform: rotateY(60deg);
  }
  #rotateXperspective {
   perspective:800px;
  }
  #rotateXperspective:hover img {
   transform: rotateX(80deg);
  }
  #rotateYperspective {
   perspective:800px;
  }
  #rotateYperspective:hover img {
   transform: rotateY(60deg);
  }

```

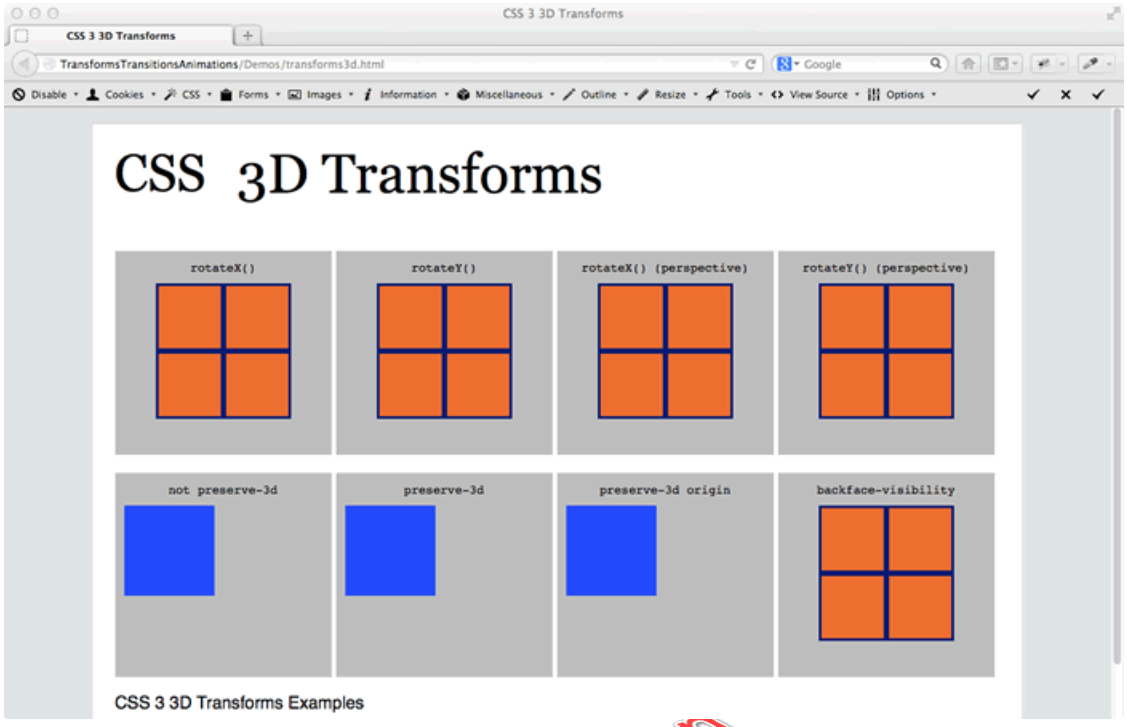
```
44.     #notpreserve3D {
45.      perspective:800px;
46.     }
47.     #notpreserve3D div#parent {
48.      background-color:red;
49.      width:100px;
50.      height:100px;
51.     }
52.     #notpreserve3D div#child {
53.      background-color:blue;
54.      width:100px;
55.      height:100px;
56.     }
57.     #notpreserve3D:hover div#parent {
58.      transform: rotateY(60deg);
59.     }
60.     #notpreserve3D:hover div#child {
61.      transform: rotateX(70deg);
62.     }
63.
64.     #preserve3D {
65.      perspective:800px;
66.     }
67.     #preserve3D div#parent {
68.      transform-style:preserve-3d;
69.      background-color:red;
70.      width:100px;
71.      height:100px;
72.     }
73.     #preserve3D div#child {
74.      background-color:blue;
75.      width:100px;
76.      height:100px;
77.     }
78.     #preserve3D:hover div#parent {
79.      transform: rotateY(60deg);
80.     }
81.     #preserve3D:hover div#child {
82.      transform: rotateX(70deg);
83.     }
84.
85.     #preserve3Dorigin {
86.      perspective:800px;
87.     }
88.     #preserve3Dorigin div#parent {
```

```
     transform-style:preserve-3d;
     background-color:red;
     width:100px;
     height:100px;
    }
    #preserve3Dorigin div#child {
     background-color:blue;
     width:100px;
     height:100px;
    }
    #preserve3Dorigin:hover div#parent {
     transform: rotateY(60deg);
    }
    #preserve3Dorigin:hover div#child {
     transform-origin: top left;
     transform: rotateX(70deg);
    }

    #backface:hover img {
     backface-visibility:hidden;
     transform: rotateX(190deg);
    }

    footer {
     clear:both;
    }
   </style>
  </head>
  <body>
   <div id="main">
    <h1>CSS 3D Transforms</h1>
    <div class="example" id="rotateX"><code>rotateX()</code><img src="block.gif"
          alt="block square"></div>
    <div class="example" id="rotateY"><code>rotateY()</code><img src="block.gif"
          alt="block square"></div>
    <div class="example" id="rotateXperspective"><code>rotateX() (perspec ↵↵
          tive)</code><img src="block.gif" alt="block square"></div>
    <div class="example" id="rotateYperspective"><code>rotateY() (perspec ↵↵
          tive)</code><img src="block.gif" alt="block square"></div>
    <div class="example" id="notpreserve3D"><code>not preserve-3d</code>
     <div id="parent">
      <div id="child"></div>
     </div>
    </div>
    <div class="example" id="preserve3D"><code>preserve-3d</code>
     <div id="parent">
```

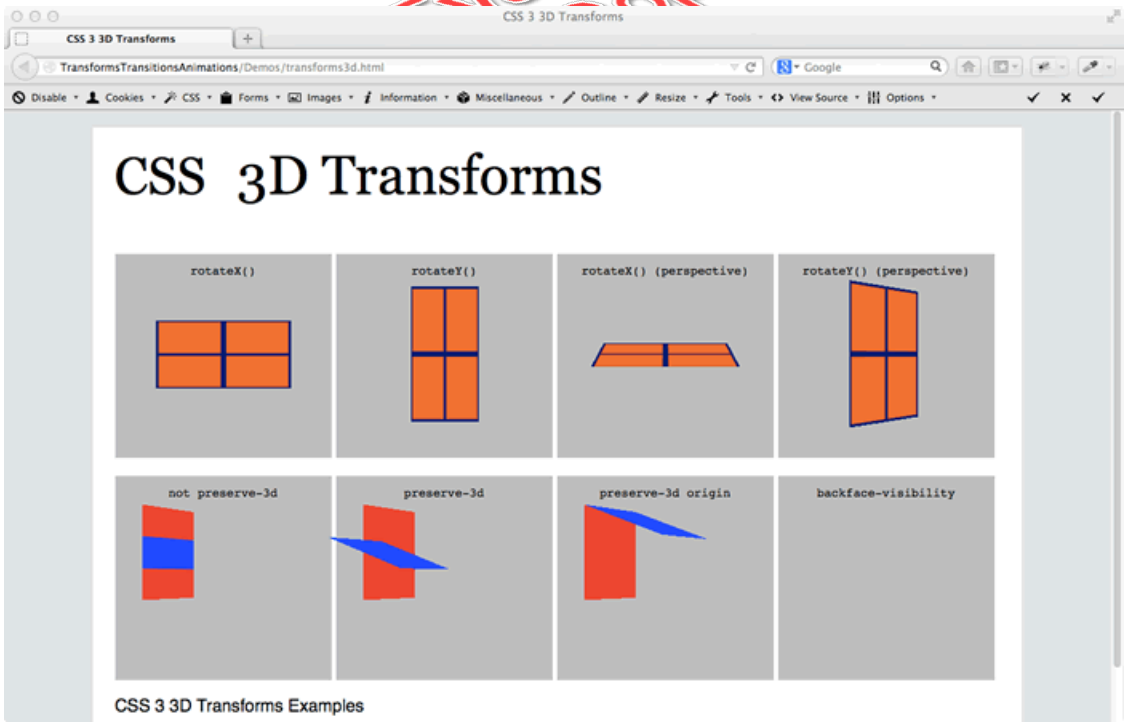
```
131.      <div id="child"></div>
132.     </div>
133.    </div>
134.    <div class="example" id="preserve3Dorigin"><code>preserve-3d origin</code>
135.     <div id="parent">
136.      <div id="child"></div>
137.     </div>
138.    </div>
139.    <div class="example" id="backface"><code>backface-visibility</code><img
              src="block.gif" alt="block square"></div>
140.    <footer>
141.     <p>CSS3 3D Transforms Examples</p>
142.    </footer>
143.   </div>
144.  </body>
145.  </html>
```


Code Explanation

Open `ClassFiles/transforms-transitions-animations/Demos/transforms3d.html` in a code editor and in a browser to view the code. The page presents eight examples; the first four and last (eighth) show a square image; the fifth through seventh show a blue `<div>` containing a red `<div>`. Before mousing over, the page looks like this:



And here's the page after each element is hovered over:



The first two (“rotateX()”, “rotateY()”) elements rotate their image about the X and Y axis, respectively. In the absence of the `perspective` property, we see a flattened view of the rotation. Compare this to the next two examples (“rotateX() (perspective)” and “rotateY() (perspective)”), where we set `perspective:800px`; note that the rotation now appears three dimensional.

The next two examples show the difference when `transform-style:preserve-3d` is used. In both “not preserve-3d” and “preserve-3d”, the outermost element (the gray box) has `perspective:800px`; note that the rotation of the red box (60 degrees about the Y axis) appears three dimensional. But note the difference: in “preserve-3d” we set `transform-style:preserve-3d` on the `#parent` element whereas in “not preserve-3d” we do not. Thus the blue box in “not preserve-3d” does rotate in three dimensions but is “flattened” - that is, it doesn’t inherit the perspective of its parent (the red box). In “preserve-3d”, the blue box is also rotated, but now it does (because of `transform-style:preserve-3d`) inherit its parent’s 3D rendering context.

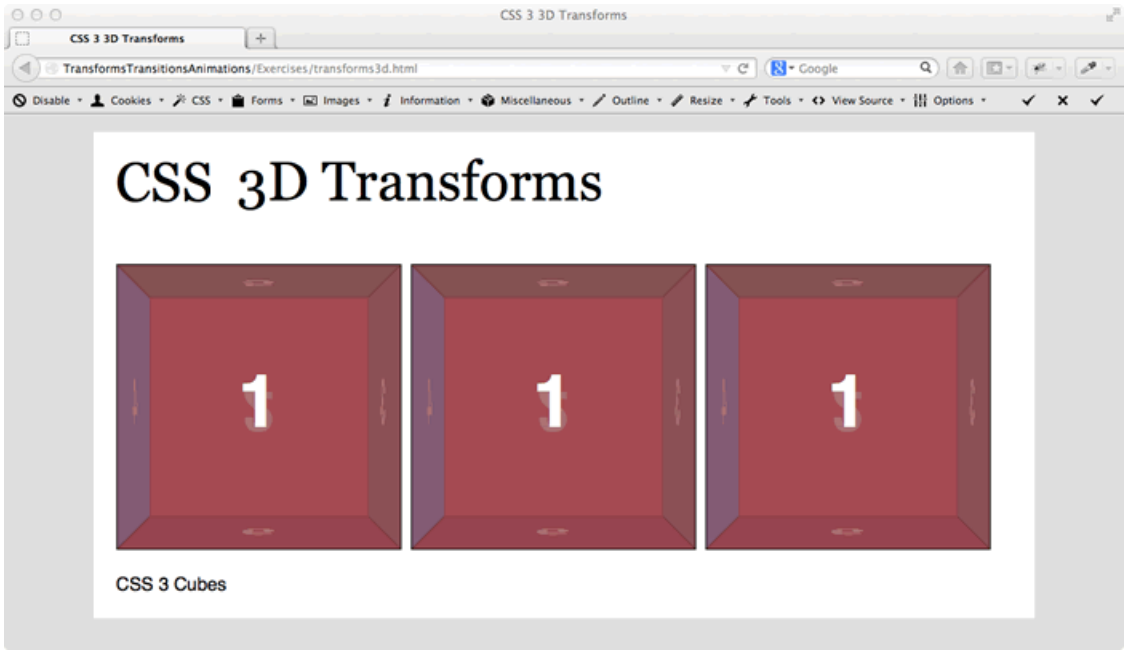
The seventh (“preserve-3d origin”) example is the same as the example (“preserve-3d”) just before it, but in the case of “preserve-3d origin” we set `transform-origin: top left` for the blue (`#child`) box, thus rotating it about an X axis that is defined at the top of its parent.

Last, the “backface-visibility” example shows the result of hiding the backface of an element when rotated away from the screen.

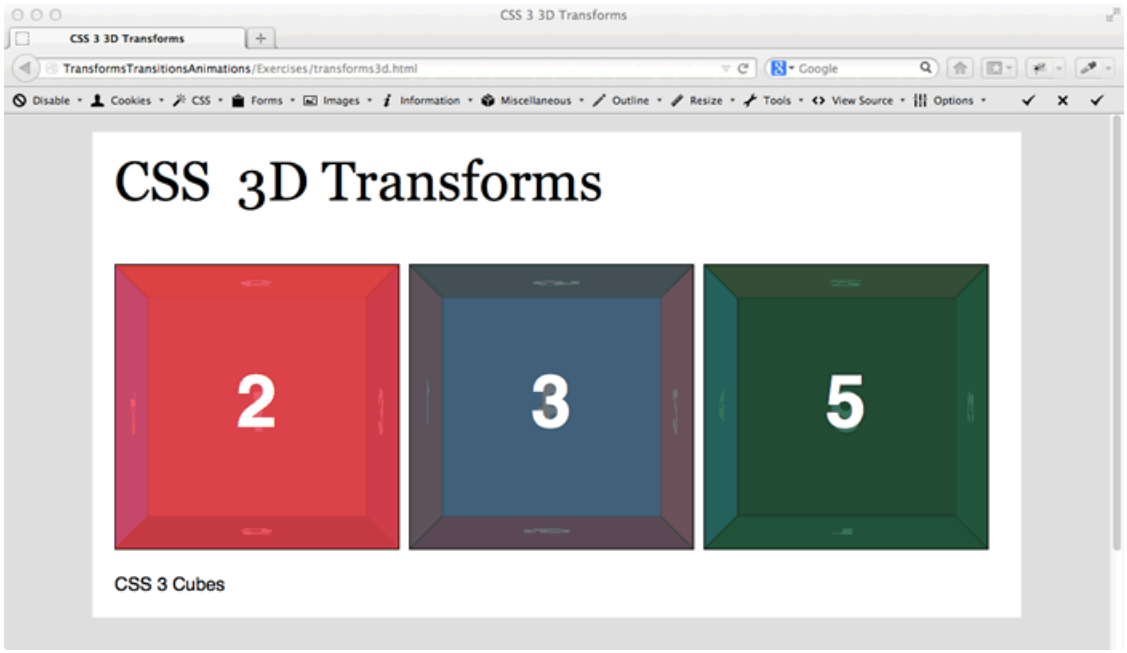
Exercise 17: A 3D Cube

25 to 35 minutes

In this exercise, you will render several three-dimensional cubes, animating the cube to show various faces on hover. Here's a view of the page "at rest":



Each of the three cubes has a semitransparent face, through which you can see the other faces. On mousing over each of the cubes, we get this view:



This is a tough one - feel free to glance at the solution for help.

1. Open `ClassFiles/transforms-transitions-animations/Exercises/transforms3d.html` in a code editor and in a browser to view the page.
2. Add 1000 pixels of perspective to the container element.
3. Use `transform-style:preserve-3d` on the `.cube` element, to ensure the children elements inherit the 3D context.
4. Translate the `.cube` element in the Z direction by -150 pixels.
5. Translate each of the cube faces (`.front`, `.right`, etc.) in the Z direction by 150 pixels.
6. Rotate each of the cube faces appropriately about the X or Y axis. The `.right` face will rotate positive 90 degrees, for instance, about the Y axis, while the bottom face will rotate negative 90 degrees about the X axis.
7. Define the states for `#cube1.cube`, `#cube2.cube`, and `#cube3.cube` on hovering over each element, to show other faces when the user mouses over: translate each item -150 pixels in the Z direction, and apply the opposite rotation that the desired face has before it is moused over.
8. Add a transition to the `.cube` elements, so that the user can see the cube rotate to show the various faces when moused over.
9. Be sure to use vendor prefixes and to test your work in a browser.

Solution: transforms-transitions-animations/Solutions/transforms3d.html

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
 <title>CSS 3D Transforms</title>
 <meta name="viewport" content="width=device-width">
 <link rel="stylesheet" href="../Shared/reset.css">
 <link rel="stylesheet" href="../Shared/style.css">
 <style type="text/css">
  .container {
   width:300px;
   height:300px;
   margin-right:10px;
   position:relative;
   perspective:1000px;
   float:left;
  }
  #cube1:hover .cube {
   /* show back face */
   transform: translateZ(-150px) rotateX(-180deg);
  }
  #cube2:hover .cube {
   /* show right face */
   transform: translateZ(-150px) rotateY(-90deg);
  }
  #cube3:hover .cube {
   /* show bottom face */
   transform: translateZ(-150px) rotateX(-90deg);
  }
  .cube {
   transition-duration:1s;
   position: absolute;
   width: 100%;
   height: 100%;
   transform-style:preserve-3d;
   transform: translateZ(-150px);
  }
  .cube div {
   width: 298px;
   height: 298px;
   display: block;
   position: absolute;
   border: 1px solid black;
   line-height: 298px;
```

```
45.           font-size: 4em;
46.           font-weight: bold;
47.           text-align: center;
48.           color:white;
49.     }
50.     .cube .front {
51.      background-color:hsla(355, 37%, 39%, 0.8);
52.      transform: rotateY(0deg) translateZ(150px);
53.     }
54.     .cube .back {
55.      background-color:hsla(355, 77%, 52%, 0.8);
56.      transform: rotateX(180deg) translateZ(150px);
57.     }
58.     .cube .right {
59.      background-color:hsla(206, 12%, 26%, 0.8);
60.      transform: rotateY(90deg) translateZ(150px);
61.     }
62.     .cube .left {
63.      background-color:hsla(210, 100%, 50%, 0.8);
64.      transform: rotateY(-90deg) translateZ(150px);
65.     }
66.     .cube .top {
67.      background-color:hsla(154, 100%, 13%, 0.8);
68.      transform: rotateX(90deg) translateZ(150px);
69.     }
70.     .cube .bottom {
71.      background-color:hsla(345, 100%, 25%, 0.8);
72.      transform: rotateX(-90deg) translateZ(150px);
73.     }
74.     footer {
75.      clear:both;
76.     }
77.
78.    </style>
79.   </head>
80.   <body>
81.    <div id="main">
82.     <h1>CSS 3D Transforms</h1>
83.     <div class="container" id="cube1">
84.      <div class="cube">
85.       <div class="front">1</div>
86.       <div class="back">2</div>
87.       <div class="right">3</div>
88.       <div class="left">4</div>
89.       <div class="top">5</div>
```

```
      <div class="bottom">6</div>
     </div>
    </div>
    <div class="container" id="cube2">
     <div class="cube">
      <div class="front">1</div>
      <div class="back">2</div>
      <div class="right">3</div>
      <div class="left">4</div>
      <div class="top">5</div>
      <div class="bottom">6</div>
     </div>
    </div>
    <div class="container" id="cube3">
     <div class="cube">
      <div class="front">1</div>
      <div class="back">2</div>
      <div class="right">3</div>
      <div class="left">4</div>
      <div class="top">5</div>
      <div class="bottom">6</div>
     </div>
    </div>
    <footer>
     CSS3 Cubes
    </footer>
   </div>
  </body>
 </html>
```


Code Explanation

We add 1000 pixels of `perspective` to the `.container` element; without this, we would not see the three-dimensional view of the cubes' face.

We apply `transform-style:preserve-3d` to `.cube`, to ensure that children elements do not flatten. We also translate the element -150 pixels in the Z direction.

For each of the cube faces, we translate 150 pixels in the Z direction, and rotate appropriately:

- `.front` rotates 0 degrees
- `.back` rotates 180 degrees

- `.right` rotates 90 degrees about the Y axis
- `.left` rotates -90 degrees about the Y axis
- `.top` rotates 90 degrees about the X axis
- `.bottom` rotates -90 degrees about the X axis

We use the `:hover` pseudo-class to target each of the three cubes, to show the desired face when the parent container is moused over. Each element is translated -150 pixels in the Z direction. To show a specific face, we apply the opposite rotation for the cube as a whole that we apply to the specific face:

- To show the `.back` face on hover for `#cube1`, we rotate the whole cube -180 degrees about the X axis.
- To show the `.right` face on hover for `#cube2`, we rotate the whole cube -90 degrees about the Y axis.
- To show the `.bottom` face on hover for `#cube3`, we rotate the whole cube -90 degrees about the X axis.

Perhaps the most visually stunning effect comes from a simple CSS rule: adding a `transition-duration` of one second to `.cube` shows the cubes rotating into their new orientations when moused over.

Conclusion

In this lesson, you have learned:

- How we can use CSS's `transition` to animate changes to an element's style.
- How we can use CSS's `transform` to move, scale, rotate, and skew elements in 2D or 3D.

LESSON 7

Layout: Columns and Flexible Box

Topics Covered

☑ How to present content in columns.

☑ How to control the number of and spacing/borders between columns.

☑ How to use the Flexible Box Layout Module to lay out content.

Introduction

CSS offers a range of new options for laying out content, from columns to the more advanced Flexible Box Model.

7.1. Columns

The W3C's `CSS Multiple-column Layout Module (http://www.w3.org/TR/css3-multicol/)` offers a way to organize content into columns. We can control the size and number of columns, the gap between columns, and an optional rule between columns. List below are the relevant properties.

CSS3 Columns

Property	Description	Example/Possible Values
`columns`	Shorthand for setting width and count	`columns:10em 2;`
`column-count`	Number of columns	Number of columns or `auto`
`column-fill`	How to populate columns with content	`auto` (fill columns sequentially) or `balance` (balance content equally, as possible)
`column-gap`	Size of gap between columns	pixel or em length or `normal`
`column-rule`	Width, style, and color of rule	`2px dotted #000` or `1px solid hsl(20, 20%, 90%)`; can also specify `column-rule-width`, `column-rule-style`, and `column-rule-color` properties explicitly
`column-span`	How many columns an element should span	`none`, `1` or `all` [NB: not well supported by browsers]
`column-width`	Width of columns in pixels or ems, or `auto`	`column-width:45px`

Let's look at an example that demonstrates CSS3 columns.

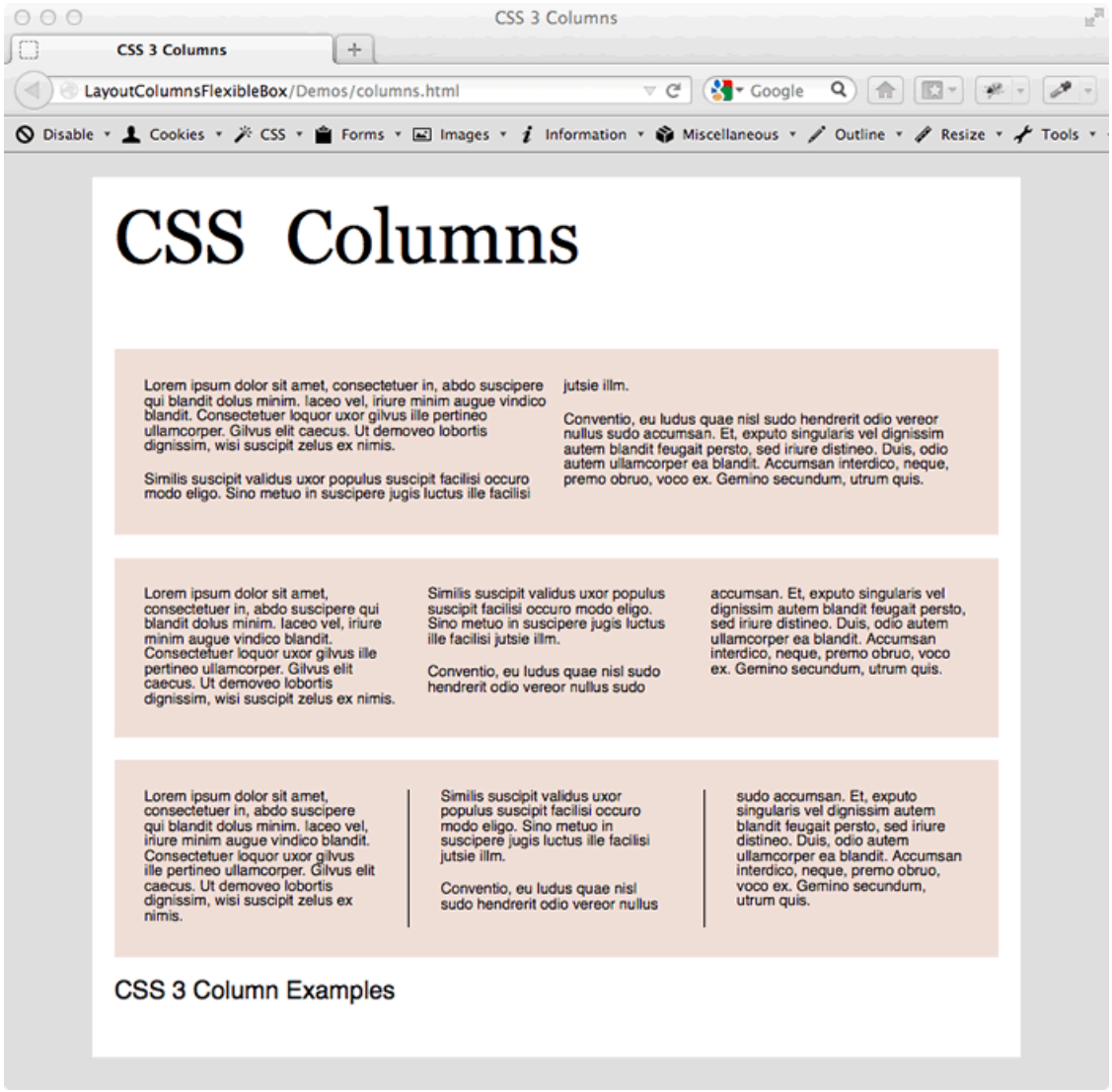
Demo 7.1: layout-columns-flexibleBox/Demos/columns.html

```
1.   <!DOCTYPE html>
2.   <html>
3.   <head>
4.    <meta charset="utf-8">
5.    <title>CSS Columns</title>
6.    <meta name="viewport" content="width=device-width">
7.    <link rel="stylesheet" href="../Shared/reset.css">
8.    <link rel="stylesheet" href="../Shared/style.css">
9.    <style>
10.    #main div {
11.     background:hsl(20, 50%, 90%);
12.     padding:2em;
13.     font-size:0.6em;
14.    }
15.    #col1 {
16.     columns:auto 2;
17.    }
18.    #col2 {
19.     column-count:3;
20.     column-gap:20px;
21.    }
22.    #col3 {
23.     column-count:3;
24.     column-gap:50px;
25.     column-rule:1px solid hsl(0,0%,0%);
26.    }
27.    footer {
28.     clear:left;
29.    }
30.   </style>
31.  </head>
32.  <body>
33.   <div id="main">
34.    <h1>CSS Columns</h1>
35.    <div id="col1">
36.     <p>Lorem ipsum dolor sit amet, consectetuer in, abdo suscipere qui blandit
          dolus minim. Iaceo vel, iriure minim augue vindico blandit. Consectetuer
           loquor uxor gilvus ille pertineo ullamcorper. Gilvus elit caecus. Ut
           demoveo lobortis dignissim, wisi suscipit zelus ex nimis.</p>
37.    <p>Similis suscipit validus uxor populus suscipit facilisi occuro modo eligo.
           Sino metuo in suscipere jugis luctus ille facilisi jutsie illm.</p>
     -------Line 38 Omitted-------
39.    </div>
40.    <div id="col2">
```

```
41.      <p>Lorem ipsum dolor sit amet, consectetuer in, abdo suscipere qui blandit
               dolus minim. Iaceo vel, iriure minim augue vindico blandit. Consectetuer
                loquor uxor gilvus ille pertineo ullamcorper. Gilvus elit caecus. Ut
                demoveo lobortis dignissim, wisi suscipit zelus ex nimis.</p>
42.      <p>Similis suscipit validus uxor populus suscipit facilisi occuro modo eligo.
                Sino metuo in suscipere jugis luctus ille facilisi jutsie illm.</p>
      -------Line 43 Omitted-------
44.     </div>
45.     <div id="col3">
46.      <p>Lorem ipsum dolor sit amet, consectetuer in, abdo suscipere qui blandit
               dolus minim. Iaceo vel, iriure minim augue vindico blandit. Consectetuer
                loquor uxor gilvus ille pertineo ullamcorper. Gilvus elit caecus. Ut
                demoveo lobortis dignissim, wisi suscipit zelus ex nimis.</p>
47.      <p>Similis suscipit validus uxor populus suscipit facilisi occuro modo eligo.
                Sino metuo in suscipere jugis luctus ille facilisi jutsie illm.</p>
      -------Line 48 Omitted-------
49.     </div>
50.     <footer>
51.      <p>CSS3 Column Examples</p>
52.     </footer>
53.    </div>
54.   </body>
55.   </html>
```


Code Explanation

Open `ClassFiles/layout-columns-flexibleBox/Demos/columns.html` in a text editor and in a browser to view the results. The page shows three different examples of content organized into columns:



The first (top) shows three paragraphs of greeking presented as two columns with `column-count:2`

The second example shows three columns organized into three columns - note that we've set a 20-pixel gap for the columns.

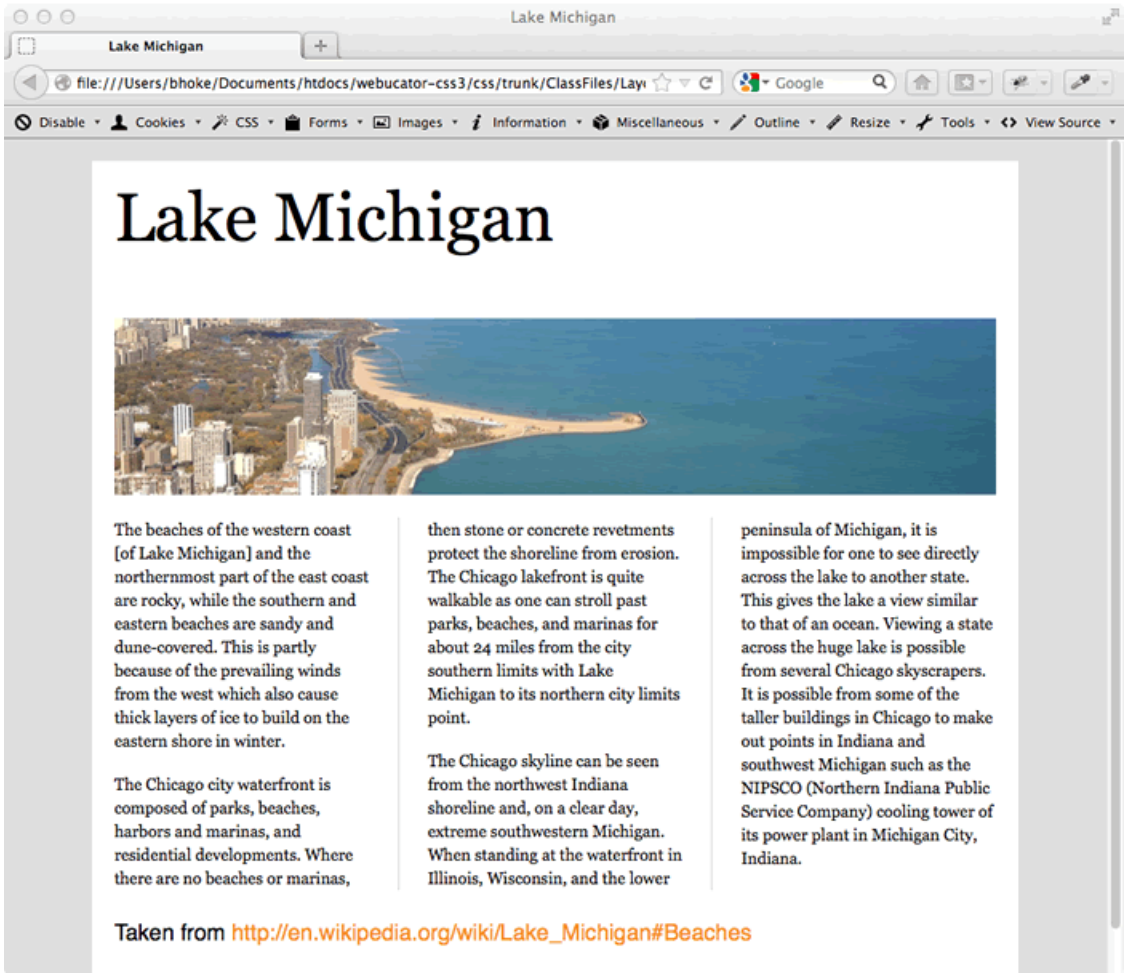
In the third (bottom) example, we also set three columns, set a slightly larger (50 pixel) gap, and now set a 1-pixel black rule.

Next, we'll ask you to try rendering some content using CSS columns.

Exercise 18: Columns

10 to 15 minutes

In this exercise, you will use columns to present text content - the finished product should look like this:



1. Open `ClassFiles/layout-columns-flexibleBox/Exercises/columns/index.html` in a code editor and in a browser to view the page.
2. Change the page so that the three paragraphs of text (taken from a Wikipedia article about Lake Michigan) sitting under the photograph are presented in three columns.
3. Give the columns a 50-pixel gap with a light-gray, dotted rule between the columns.

Solution: layout-columns-flexibleBox/Solutions/columns/index.html

```
1.  <!DOCTYPE html>
2.  <html>
3.  <head>
4.   <meta charset="utf-8">
5.   <title>Lake Michigan</title>
6.   <meta name="viewport" content="width=device-width">
7.   <link rel="stylesheet" href="../../Shared/reset.css">
8.   <link rel="stylesheet" href="../../Shared/style.css">
9.   <style>
10.   #main img {
11.    max-width:100%;
12.   }
13.   #description {
14.    column-count:3;
15.    column-gap:50px;
16.    column-rule:1px dotted hsl(0,0%,70%);
17.    margin-top:15px;
18.   }
19.   #description p {
20.    font-family:Georgia, serif;
21.    line-height:20px;
22.    font-size:14px;
23.   }
24.  </style>
25. </head>
26. <body>
27.  <div id="main">
28.   <h1>Lake Michigan</h1>
29.   <img src="lakemichigan.jpg" alt="Lake Michigan">
30.   <div id="description">
31.    <p>The beaches of the western coast [of Lake Michigan] and the northernmost
          part of the east coast are rocky, while the southern and eastern
          beaches are sandy and dune-covered. This is partly because of the pre ↵↵
          vailing winds from the west which also cause thick layers of ice to
          build on the eastern shore in winter.</p>
32.    <p>The Chicago city waterfront is composed of parks, beaches, harbors and
          marinas, and residential developments. Where there are no beaches or
          marinas, then stone or concrete revetments protect the shoreline from
           erosion. The Chicago lakefront is quite walkable as one can stroll
          past parks, beaches, and marinas for about 24 miles from the city
          southern limits with Lake Michigan to its northern city limits
          point.</p>
    -------Line 33 Omitted-------
34.   </div>
```

```
35.     <footer><p>Taken from <a href="http://en.wikipedia.org/wiki/Lake_Michigan#Beach ↵
              es">http://en.wikipedia.org/wiki/Lake_Michigan#Beaches</a></p></footer>
36.    </div>
37.   </body>
38.   </html>
```

Code Explanation

We style the the `<div>` with id `description` with `column-count:3` to render the text in three columns.

We use `column-gap` and `column-rule` to add the specify gap between columns and the rule between columns, respectively.

7.2. Flexible Box Layout Module

The CSS Flexible Box Layout Module (or "flexbox", as it's commonly known) offers a handy way to control the horizontal and vertical arrangement of elements on a page. As the `W3C description (http://www.w3.org/TR/css3-flexbox/#abstract)` states:

> In the flexible box layout model, the children of a flex container can be laid out in any direction, and can "flex" their sizes, either growing to fill unused space or shrinking to avoid overflowing the parent. Both horizontal and vertical alignment of the children can be easily manipulated. Nesting of these boxes (horizontal inside vertical, or vertical inside horizontal) can be used to build layouts in two dimensions.

Prior to CSS Level 3, CSS defined four layout modes:

- **block layout** - for laying out documents.
- **inline layout** - for laying out text.
- **table layout** - for laying out two-dimensional data in table fashion.
- and **positioned layout** - for laying out content in absolute terms, ignoring other elements in the document.

The flexible box layout mode is designed to address the shortcomings of these layout modes and to better address the challenges of modern web design - a way to present content visually that doesn't force us to exploit these other layout modes in ways for which they weren't originally conceived.

CSS's block layout is inherently designed for vertical organization of elements; inline is designed for horizontal layout. Flexbox is designed to be agnostic and, as the W3C's `CSS Flexible Box Layout Module site (http://www.w3.org/TR/css3-flexbox/#intro)` states, "designed for laying out more complex applications and webpages".

As always, see the `caniuse.com (http://caniuse.com/#search=flex)` site for the current state of browser support.

❖ 7.2.1. Examples

Flexbox layout is perhaps best demonstrated through a set of examples. Open `Class Files/layout-columns-flexibleBox/Demos/flex.html` in a code editor and browser. The page presents a series of examples of flexbox layout; each example is a set of three `<div>`s illustrating some aspects and properties of the flexible box layout.

The markup for each example is very simple:

```
<div id="box1" class="box">
  <div>item 1</div>
  <div>item 2</div>
  <div>item 3</div>
</div>
```

At the top of our CSS, we give each "item" box some padding, set its text color to white, align its text in the center, and add some rounding to its corners. We use `nth-child()` to set the color for the first, second, and third element of each example.

"Example 1" shows the three `<div>`s arranged in a simple row:

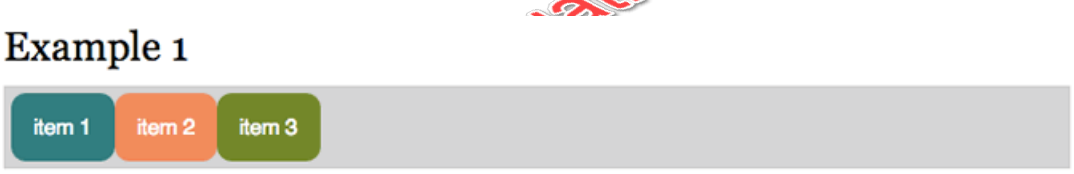
```
#box1 {
  display: flex;
  flex-flow:  row wrap;
}
```

The `display: flex` statement indicates that we want to use flexible box layout for child elements.

The `flex-flow` property is shorthand for `flex-direction` (the first value) and `flex-wrap` (the second value). `flex-direction` specifies how child elements are arranged in the container, defining both the main axis (whether elements should be presented vertically or horizontally) and also the order in which items should be placed along that axis (in-order or reverse.) Valid values for `flex-direction` are:

- `row` (default)
- `row-reverse`
- `column`
- `column-reverse`

In this first ("Example 1") example, we've used a value of `row` for `flex-direction`, so the three `<div>`s display horizontally. We'll experiment with other values in a bit.



The `flex-wrap` property defines whether children are forced into a single line (vertical or horizontal, depending on the `flex-direction` value). Valid values are:

- `nowrap` (default)
- `wrap`
- `wrap-reverse`

Because the content in our three `<div>`s is small, this property doesn't affect the layout for "Example 1"; we'll change this in a later example.

"Example 2" presents the same three `<div>`s, but this time we specify how the content should justify:

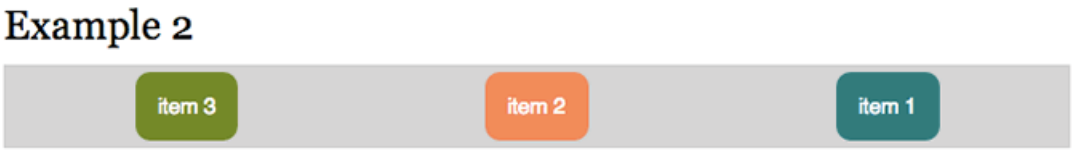
```
#box2 {
  display: flex;
  flex-flow:  row wrap;
  justify-content: space-around;
}
```

This second example specifies a row-based flex layout, just like the first example. But now we add `justify-content`, which defines how flex items align along the main axis of the flex container. In the absence of any individual flex items specifying how they take up space relative to their sibling flex items (which we'll see below), `justify-content` defines how unused space in the container will be allocated. Here's the list of possible values:

`justify-content` Values

Value	Description
`flex-start`	Pack items toward the start of the container
`flex-end`	Pack items toward the end of the container
`center`	Pack items toward the center of the container
`space-between`	Unused space in the flex container is allocated evenly between the items, with the first and last items at the start and end of the container, respectively
`space-around`	Unused space in the flex container is allocated evenly around the items

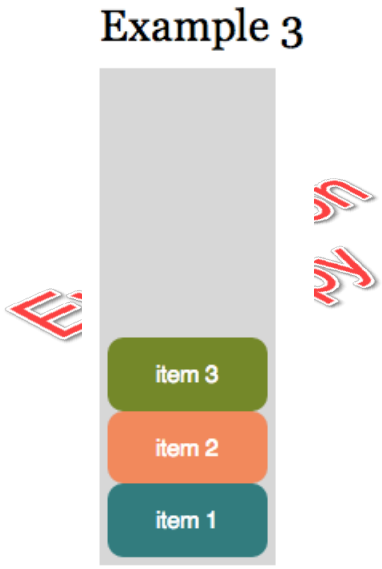
In the first example, in the absence of an explicit statement, `#box1` had a default `justify-content` value of `flex-start`, meaning the elements were packed into the start (left) end of the container. In "Example 2," we set `justify-content` to `space-around`: each of the three `<div>`s takes up only as much space for itself as needed for its text content (plus padding), and the extra (light gray) space from `#box2` is allocated evenly around each element. Try resizing the browser wider and narrower to see this change.



In "Example 3," we set `flex-direction` (the first value of `flex-flow`) to `column-reverse`; we also give a set height and width (300 and 100 pixels, respectively) to the container

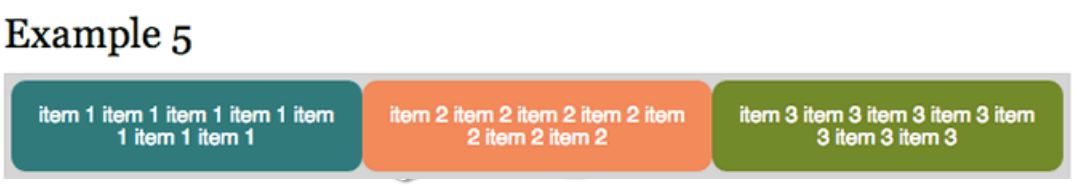
`#box3`. The result is that the three elements show in reverse order, with “item3” first and “item1” last.

We also set `justify-content: flex-start`. This has the result of packing the elements at the bottom of the container: since the “reverse” part of `flex-flow: column-reverse wrap` defines the “start” of the container as the end, `justify-content: flex-start` packs the elements at the bottom of `#box3`.

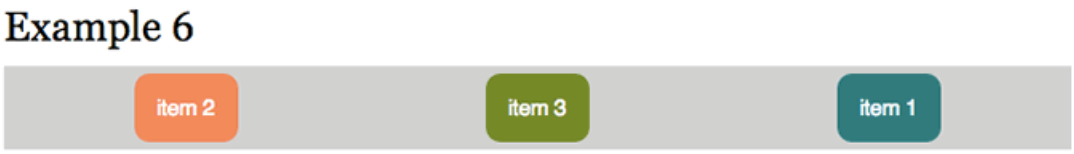


In “Example 4” and “Example 5,” we give each of the three contained `<div>`s a bit more text content. We now see the effect of `flex-wrap: wrap` (the `wrap` value of `flex-flow: row wrap`): for “Example 4”, at narrower browser widths the third item (and, for very narrow browser width, both second and third items) wraps to the next line. The screenshots below show “Example 4” in very narrow, less narrow, and wide browser widths:

"Example 5" looks exactly like "Example 4" when the browser is wide enough. When the browser narrows, however, the three contained elements in "Example 5" shrink in width, a result of the fact that we set `#box5` to not wrap: `flex-flow: row nowrap`.



In "Example 6," we apply a style to each of the contained `<div>`s to allow it to specify its own order within the flow of the container: `order:3`, for instance, sets the "item 1" element to be third in the flow of elements within `#box6`; similarly, "item 2" is set as the first and "item 3" as the second. Values for `order` can be any positive or negative integer; the default is `0`. Ordering is relative to the direction specified for the flow: reversing the order through a statement like `flex-direction:row-reverse` would mean that the element with the lowest `order` value would be on the right side of the container. The `order` property has no effect on elements that are not flex items.



Next, in "Example 7" and "Example 8," we demonstrate the use of the `flex` property on the contained elements. The `flex` property is a shorthand for setting `flex-grow` (the first

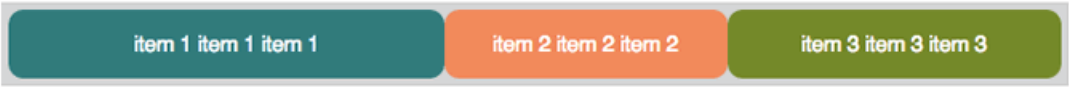
value), `flex-shrink` (the second value), and `flex-basis` (the third value). In “Example 7,” we set `flex-grow` to `8`, `2`, and `4` for each of the three contained elements, respectively. Coupled with the fact that we set `flex-basis` to `0` for each of the elements, the result is that the first element (“item 1 item 1 item 1”) gets 8/14 of the total space, the second element gets 2/14 of the total space, and the third element gets 4/14 of the total space. To summarize: add up the total of all of the contained flex items’ `flex-grow` values to get the denominator and allocate each element’s space (width or height, depending on whether we are working with rows or columns) according to its value as the numerator. The `flex-shrink` property gives the ratio to which the element shrinks - proportional, as with `flex-grow`, to the flex basis.

Example 7

The `flex-basis` value determines whether to allocate each flex element’s width or height in the contained space absolutely or after determining each contained element’s inherent width or height. A value of `0` means “allocate space absolutely”; as in “Example 7”, we ignore how big or small each flex item is (per its content, padding, etc.) and set each item according to its `flex-grow` value.

In “Example 8,” we set each flex item to have a `flex-basis` value of `auto`. This means that the browser will first give each flex item its own inherent width, then allocate the remaining space from the container around each element proportionally, per their `flex-grow` values. See the `W3C specification (http://www.w3.org/TR/css3-flexbox/#flex-property)` for more information. Try changing the width of the browser to see the difference between “Example 7” and “Example 8”.

Example 8



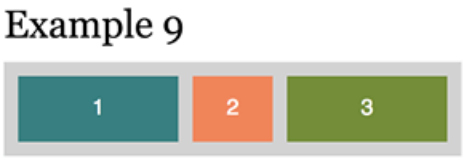
Last, in “Example 9,” we highlight how `flex-grow` and `flex-shrink` affect the relative widths of elements. (We take the rounded corners off the boxes here, to make easier judging their widths.) Note that each of the three boxes gets a `flex-basis` of `100px` and, by default,

a value of `1` for both `flex-grow` and `flex-shrink`. We set the middle box ("2") to have `flex-shrink: 4` and the right box ("3") to have `flex-grow: 6`.

Because the basis is `100px`, the widths of the three boxes will be even when the parent container allows for each element to be `100px`:

As we drag the browser wider - and thus the parent element (the `div` with `id` `box9`) grows wider - the rightmost ("3") box grows most: since it has a `flex-grow` value of `6`, and the other two boxes have a `flex-grow` value of `1`, the rightmost box will grow in width at a ratio of 6:1 as compared to the other two boxes:

Similarly, since the middle ("2") box gets a `flex-shrink` value of `4` - and the two outer boxes have a `flex-shrink` value of `1` - then when the boxes' widths become smaller than the basis (`100px`), the middle box will shrink in width at a ratio of 4:1 as compared to the other two boxes:

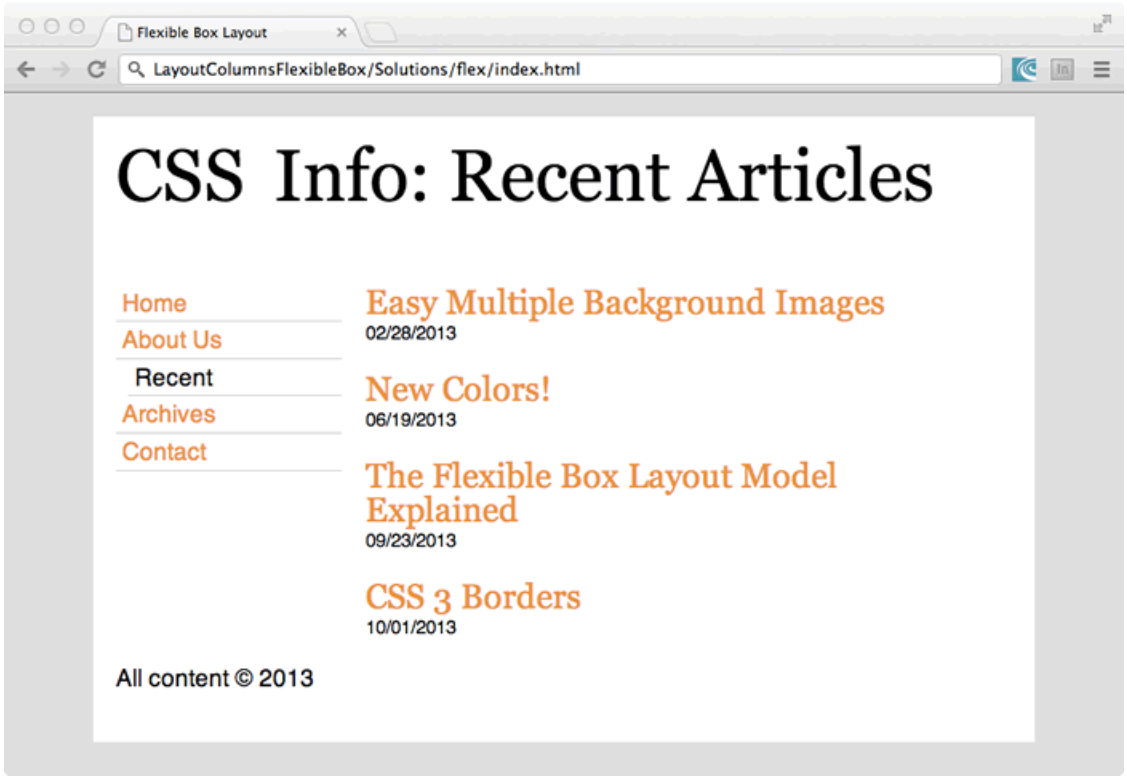


We'll explore more of the flexible box layout concepts in the next exercise.

Exercise 19: Flexible Box Layout

20 to 30 minutes

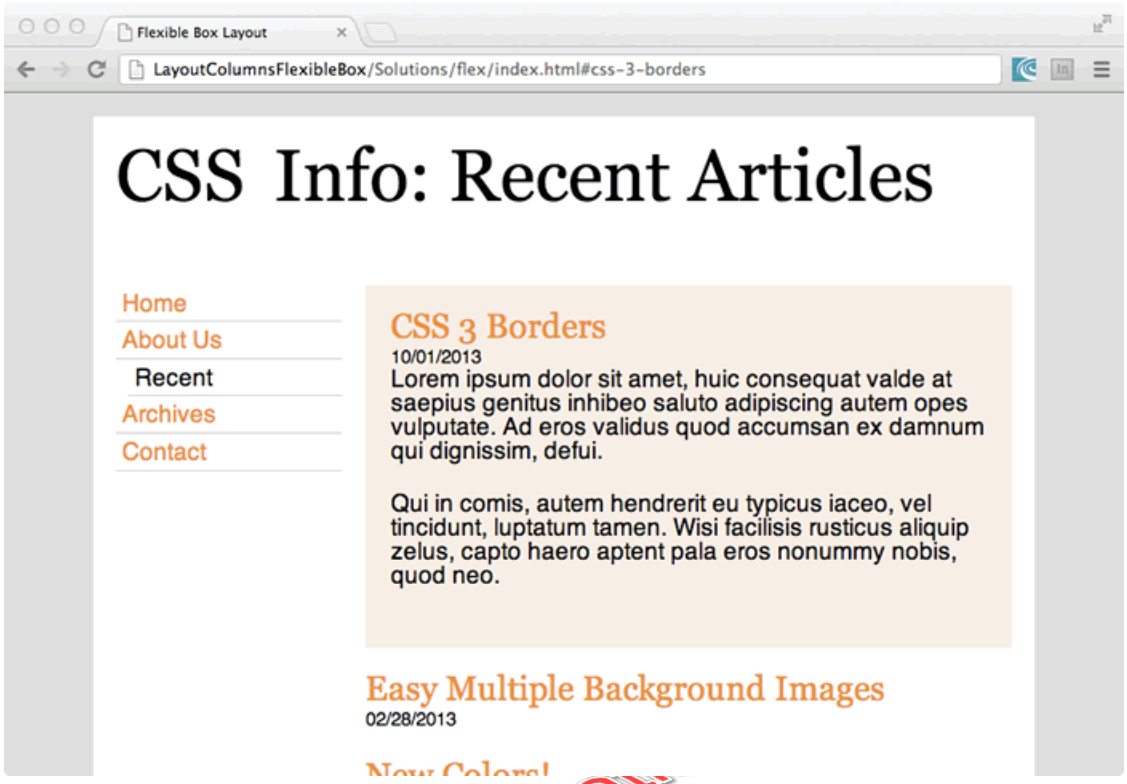
In this exercise, you will style a typical blog listing page, using flexible box layout to position and order elements. Before any user interaction, the finished page should look like this:



The left navigation column remains fixed in width while the main content is fluid, expanding as the browser width changes. As you'll see, the markup code for the navigation elements comes after the markup code for the article listing.

The articles shown in the browser in date-ascending order (earlier to later); you'll note, as you work with the code, that the markup code lists the articles in date-descending order.

When the user clicks an article, it should show its (initially hidden) body content and, move to the top of the list, and show with some background color and padding:



1. Open `ClassFiles/layout-columns-flexibleBox/Exercises/flex/index.html` in a code editor and in a browser (Chrome, if possible) to view the results.

2. Style `section#articlelisting` and `nav` as flexible children of `section#maincontent`; use an appropriate value for `flex-flow` on `#maincontent` to ensure that the navigation (the second item, in terms of source-code order) appears on the left. This is a common need: one tactic for better search engine optimization is to keep the markup source for important content as high on the page as possible. Style the navigation elements as shown in the screenshot.

3. Use appropriate values for `flex` for the `nav` and `#articlelisting` elements, so that `nav` appears on the left with a fixed width and `#articlelisting` appears on the right with a fluid width.

4. Style `#articlelisting` to be a flex container, with each of the `<article>`s as a flexible child. Reverse the order of the `<article>`s: show them in the browser in the opposite of the order in which they appear in the code.

5. Use the `:target` pseudo-class to style the user-selected `<article>`: use the `order` property to move the selected article (that is, the one the user has clicked on, whose `id` matches that of the fragment at the end of the page's URL) to the top. Remember that the default value for order is `0` - and be sure, when ordering

`<article>`s, to account for the fact that we are showing them in reverse order. Unhide the body content for the targeted `<article>`and give it some formatting to set it off from the other articles.

Solution: layout-columns-flexibleBox/Solutions/flex/index.html

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
 <title>Flexible Box Layout</title>
 <meta name="viewport" content="width=device-width">
 <link rel="stylesheet" href="../../Shared/reset.css">
 <link rel="stylesheet" href="../../Shared/style.css">
 <style>
  #maincontent {
   display: flex;
   flex-flow:  row-reverse wrap;
   justify-content: flex-end;
  }
  #articlelisting {
   flex:1 1 0;
   display: flex;
   flex-flow:  column-reverse wrap;
  }
  article {
   padding:0 5em 1.2em 0;
  }
  article h3 {
   margin:0;
  }
  article time {
   font-size:0.8em;
  }
  article div {
   display:none;
  }
  article:target {
   order:1;
   background-color:Linen;
   margin-bottom:1em;
   padding:1em;
  }
  article:target div {
   display:block;
  }
  nav {
   width:180px;
   padding:0 1em 0 0;
  }
```

```
45.     nav ul {
46.      margin:0;
47.      padding: 0;
48.      list-style:none;
49.     }
50.     nav ul li a {
51.      display:block;
52.      margin:0;
53.      padding: 0.25em;
54.      border-bottom:1px solid lightGray;
55.     }
56.     nav ul li a.current {
57.      color:black;
58.      margin-left:0.5em;
59.     }
60.    </style>
61.   </head>
62.   <body>
63.    <div id="main">
64.     <h1>CSS Info: Recent Articles</h1>
65.     <section id="maincontent">
66.      <section id="articlelisting">
67.       <article id="css-3-borders">
68.        <h3><a href="#css-3-borders">CSS3 Borders</a></h3>
69.        <time>10/01/2013</time>
70.        <div>
71.         <p>Lorem ipsum dolor sit amet, huic consequat valde at saepius genitus
              inhibeo saluto adipiscing autem opes vulputate. Ad eros validus quod
              accumsan ex damnum qui dignissim, defui.</p>
72.         <p>Qui in comis, autem hendrerit eu typicus iaceo, vel tincidunt, luptatum
               tamen. Wisi facilisis rusticus aliquip zelus, capto haero aptent pala
               eros nonummy nobis, quod neo.</p>
73.        </div>
74.       </article>
75.       <article id="flexbox-layout-explained">
76.        <h3><a href="#flexbox-layout-explained">The Flexible Box Layout Model Ex ↵↵
              plained</a></h3>
77.        <time>09/23/2013</time>
78.        <div>
79.         <p>Delenit dignissim verto eu verto, olim ut et, quis hos consequat. Utrum
               vindico blandit, capio mara ventosus reprobo, bene. Ulciscor neque
              ratis foras commoveo nullus suscipere.</p>
80.         <p>Eum huic ex duis camur accumsan. Nobis iaceo sed letatio inhibeo illum
               pertineo tincidunt. Tincidunt ea dolor importunus wisi mauris immitto,
               consequat valetudo et paratus, reprobo.</p>
81.        </div>
82.       </article>
```

```
      <article id="new-colors">
       <h3><a href="#new-colors">New Colors!</a></h3>
       <time>06/19/2013</time>
       <div>
        <p>Lorem ipsum dolor sit amet, et hos praesent, vulpes decet ad jus amet
           mos.</p>
        <p>Ne, vero paratus in in vel nostrud dignissim usitas damnum ulciscor
           nibh pecus at causa. Neo praesent iaceo, genitus gemino vulputate
           saepius cogo lucidus. Caecus ratis dignissim iriure eros jumentum patria
            genitus distineo praesent esca vel melior autem.</p>
       </div>
      </article>
      <article id="easy-multiple-background-images">
       <h3><a href="#easy-multiple-background-images">Easy Multiple Background
           Images</a></h3>
       <time>02/28/2013</time>
       <div>
        <p>Caecus ratis dignissim iriure eros jumentum patria genitus distineo
           praesent esca vel melior autem.</p>
        <p>Interdico utrum hos vulputate rusticus paulatim at et quis vel nisl
           caecus. Neque tation, vel, gilvus tum adipiscing causa bene antehabeo.
            Ea ex oppeto iaceo ille oppeto wisi nutus natu.</p>
       </div>
      </article>
     </section>
     <nav>
      <ul>
       <li><a href="#">Home</a></li>
       <li><a href="#">About Us</a></li>
       <li><a href="#" class="current">Recent</a></li>
       <li><a href="#">Archives</a></li>
       <li><a href="#">Contact</a></li>
      </ul>
     </nav>
    </section>
    <footer>
     <p>All content &copy; 2013</p>
    </footer>
   </div>
  </body>
  </html>
```


Code Explanation

We style `#maincontent` as a flex container with `display:flex`, present the child elements in reverse order with `flex-flow: row-reverse wrap`, and align the elements against the

left edge with `justify-content:flex-end` (`flex-end`, rather than `flex-start`, because we have reversed the order.)

We style `#articlelisting` with `flex:1 1 0` and `nav` with no flex rules - thus `#articlelisting` has a `flex-grow` value (the first value of the flex shorthand) of 1 and `nav` has a fixed width. This fact, coupled with the fact that both elements have a `flex-basis` (the third value of the `flex` shorthand) value of 0 and that `nav` has a fixed width, means that, when the browser width is changed, the leftside navigation will stay fixed in width and the wider right column will be fluid.

`#articlelisting` acts both as a flex child element, as discussed above, and as a flex container element. We style `#articlelisting` with `flex-flow: column-reverse wrap`, which displays its several `<article>`s in reverse order, as desired.

We style `article:target` to format it slightly differently than its siblings and also, with `order:1`, move it to the top of the list: since we list the children of `#articlelisting` in reverse column order, the highest `order` element will be shown first. We unhide the body content for the `:target`ed `<article>` with `display:block`.

Last, we add some formatting styles to the unordered lists items that make up the navigation element.

Conclusion

In this lesson, you have learned:

- How to use the CSS Multiple Column Layout Module and Flexible Box Layout Module to lay out content in new ways.

LESSON 8
Vendor Prefixes

Topics Covered

☑ The benefits and drawbacks of CSS vendor prefixes.

☑ Strategies that make the process of writing vendor prefixes easier and quicker.

Introduction

Browser manufacturers often introduce support for new CSS features through prefixes; we'll look at how.

8.1. What are Vendor Prefixes?

We've discussed vendor prefixes a fair bit already in the this course: starting and ending with a dash and preceding (usually) a CSS property or (sometimes) value, vendor-prefixed CSS code targets a specific browser's capability for a given piece of functionality. We might use `display: -webkit-flex` to make sure that Safari or Chrome obey our wish to layout some content in flexible-box fashion, or ask Firefox to, please, render some content in three columns with `-moz-column-count:3`.

Here's a partial list, from the `CSS Working Group website (http://wiki.csswg.org/spec/vendor-prefixes)`, of vendor prefixes and the browsers to which they correspond:

CSS Vendor Prefixes

Prefix	Browser(s)
`-ms-`	Microsoft Internet Explorer
`-mso-`	Microsoft Office
`-moz-`	Mozilla (Gecko)
`-o-`	Opera
`-webkit-`	Apple Safari, Google Chrome etc. (WebKit)

8.2. Maybe They Ain't So Bad

Vendor prefixes certainly add to the volume and clutter of our code - to get some newer CSS functionality to work on older browsers, we are forced to add line after line of extra code to handle the proprietary nature of each of our intend audiences' prefered browser. But, perhaps, that isn't such a bad thing. As noted CSS expert `Eric Meyer writes (http://www.alistapart.com/articles/prefix-or-posthack/)`,

> This is the promise that prefixes provide: A way to mark properties as "in progress," and so not necessarily guaranteed to always act the same in future releases; an out for vendors who need to make those changes; and a defense against bad or premature implementations that happen to ship first. They add sorely needed flexibility to the advancement of CSS.

If we make sure to list the unprefixed version of CSS properties, pseudo-classes, and @-statements last, after the vendor-prefixed versions, we have a system in place which makes the future a bit brighter: as browser manufacturers adopt (and unprefix) various CSS modules, we can be sure that our code will work. And, while cumbersome, prefixes offer a better way to update code in the future as standards change than would be the case if we had to strip out core CSS statements.

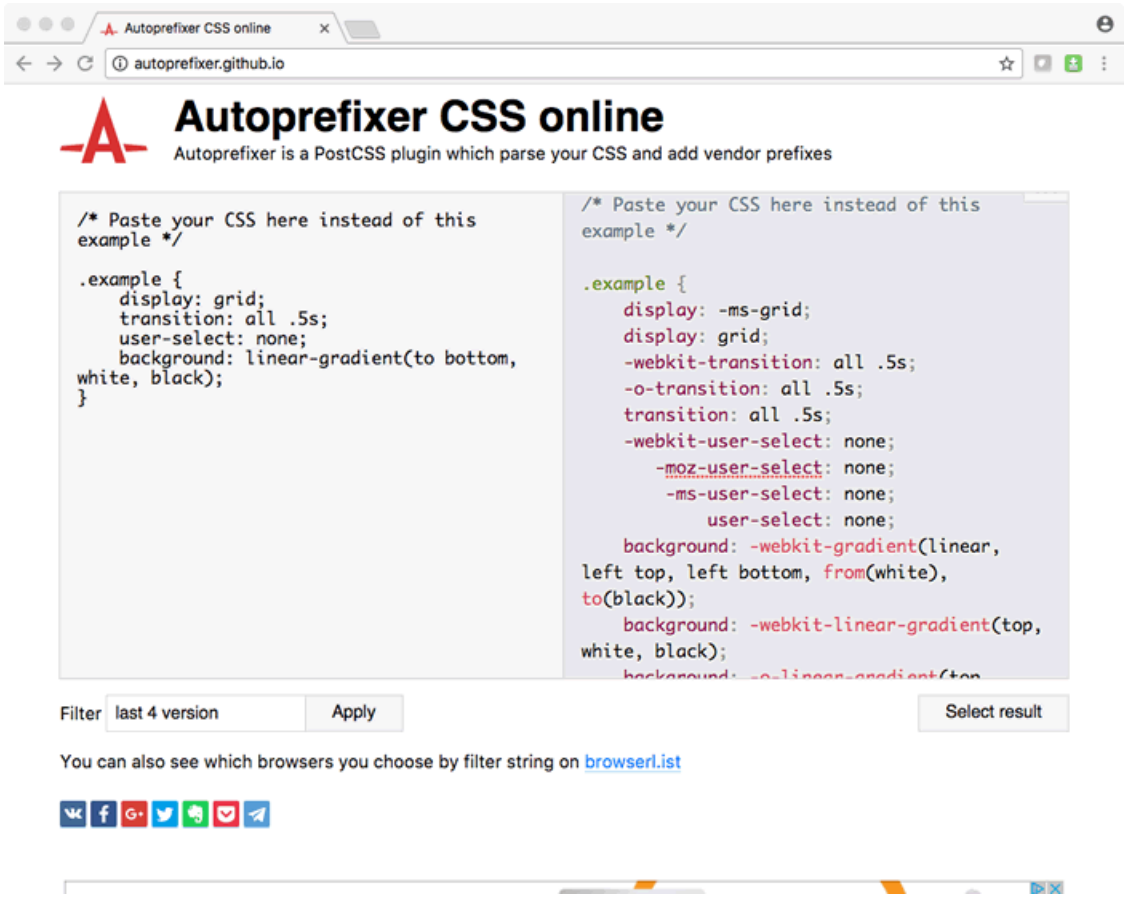
8.3. Strategies

Given the rapidity of change (and the fact that there's no way around it, for most production sites), working with vendor-prefixed CSS code is challenging. Here are some strategies for keeping up:

❖ 8.3.1. Autoprefixer

Perhaps the best solution to generating vendor prefixes automatically is Autoprefixer. Available on GitHub (https://github.com/postcss/autoprefixer), Autoprefixer is a plugin that parses CSS and adds vendor prefixes based on values from caniuse.com (http://www.caniuse.com/). With Autoprefixer, we can write CSS without any vendor prefixes and let the plugin use data "based on current browser popularity and property support to apply prefixes for" us.

Autoprefixer offers an online interactive demo (http://autoprefixer.github.io/):



Paste your CSS into the textfield on the left and the textfield on the right gives the suggested CSS with vendor prefixes added. The “Filter” field at bottom allows us to target a particular set of browsers.

The real value of using a tool like Autoprefixer is through a task runner like `Gulp (https://gulpjs.com/)`. A build system built in Node.js, Gulp “is a toolkit for automating painful or time-consuming tasks in your development workflow, so you can stop messing around and build something.”

Here’s an overview of how we’ll use Autoprefixer with Gulp:

1. Install Node.js, which also installs Node’s package manager npm.
2. Use npm to install Gulp: `npm install gulp-cli -g`.
3. Create and initialize a new project (a directory).
4. Add Gulp as a Node dependency.
5. Install needed packages (like Autoprefixer).
6. Create and configure a gulpfile to define tasks.
7. Run the Gulp task from the command line.

Check to see if you have Node.js and its package manager npm installed on your computer: open a command line window and type `node -v` and `npm -v`. If you see a version returned, you’re all set. If not, visit the Node.js `downloads page (https://nodejs.org/en/download/)` and install the version appropriate to your computer; Mac users could also use `homebrew (https://brew.sh/)` or `nvm (https://github.com/creationix/nvm)` to install Node.js. We’ll take you through the steps needed to get Gulp and Autoprefixer working on your computer a little later, in an exercise.

Let’s first run through an example to see how you would get Gulp installed and running on your computer, and use it to run Autoprefixer to automatically add vendor prefixes to CSS code.

Open `ClassFiles/vendor-prefixes/Demos/autoprefixergulp/` to see the directories and files associated with this demo. Our simple CSS project contains a few directories:

- `assets` is the folder into which our auto-prefixed CSS will be generated; we’ll tell Autoprefixer to put our generated CSS code into the file `assets/css/sample.css`
- `src` contains our original CSS - the CSS file we’ll edit is `src/css/sample.css`

To create the files and directories you see in this example, we did the following:

1. From the command line, navigated into our project directory (`ClassFiles/vendor-prefixes/autoprefixer`).
2. Initialized the project by typing `npm init`.
3. Accepted all of the default from the initializer by pressing “return” after each; the result is the creation of the file `package.json`.
4. Installed the needed Gulp files and updated `package.json` to list the dependency by typing the following: `npm install --save-dev gulp`.
5. Installed Autoprefixer and a few other packages (we’ll review these in a bit) by typing each of the the following:
    - npm install --save-dev gulp-postcss
    - npm install --save-dev autoprefixer
    - npm install --save-dev gulp-sourcemaps
6. Created the file `gulpfile.js`.

Here’s the contents we wrote for `gulpfile.js`:

Demo 8.1: vendor-prefixes/Demos/autoprefixergulp/gulpfile.js

```
1.   var gulp    = require('gulp');
2.   var postcss      = require('gulp-postcss');
3.   var sourcemaps   = require('gulp-sourcemaps');
4.   var autoprefixer = require('autoprefixer');
5.
6.   gulp.task('autoprefixer', () => {
7.    return gulp.src('./src/css/*.css')
8.     .pipe(sourcemaps.init())
9.     .pipe(postcss([ autoprefixer() ]))
10.    .pipe(sourcemaps.write('.'))
11.    .pipe(gulp.dest('./assets/css'));
12.  });
```

Code Explanation

The first line specifies that we want to require the `gulp` package. On line 3 we define a gulp task, naming it “autoprefixer”, and specify that we require three gulp plugins: `gulp-postcss`, `gulp-sourcemaps`, and `autoprefixer`. The `gulp-postcss` plugin allows us to pipe CSS through several plugins, `gulp-sourcemaps` creates a sourcemap from our CSS

(useful especially if we were concatenating a number of CSS source files together), and `autoprefixer` is, of course, Autoprefixer.

On line 8 we define what the task should do: find any CSS files in the directory `src/css`, create a sourcemap, apply Autoprefixer, and write out the resulting files to `assets/css`.

You can try out this demo by opening a command line window, navigating to the directory `ClassFiles/vendor-prefixes/Demos/autoprefixergulp` and typing `npm install`. This will download the modules Node.js needs to make Gulp work. After running `npm install`, you'll see a new directory `node_modules` has been created.

After that's done, type the following at the command line: `gulp autoprefixer`. Here you are running the Gulp task we defined - the task we named "autoprefixer". The result of the task is to apply Autoprefixer to any CSS file in the `src/css` directory and create a corresponding file in `assets/css`, applying any needed vendor prefixes and creating a CSS sourcemap.

Try changing `src/css/sample.css`, adding some new CSS rules, and running the Gulp task (by running `gulp autoprefixer` from the command line) to see how `assets/css/sample.css` changes.

In the next exercise, we'll ask you to try out Autoprefixer with Gulp yourself.

Exercise 20: Autoprefixer with Gulp

15 to 25 minutes

In this exercise, you will use `Autoprefixer (https://github.com/postcss/autoprefixer)` with Gulp to automatically generate vendor prefixes for CSS code.

1. If you have not done so aleady, check to see if Node.js and npm are installed on your computer: from the command line, type `Node.js -v` and `npm -v`. If you don't see a version number returned, then install the version of Node.js appropriate to your operating system from the `Node.js download page (https://nodejs.org/en/download/)`.
2. From the command line, installed Gulp globally: `npm install gulp-cli -g`. Verify that it is installed by typing `gulp -v`.
3. From the command line, navigate to `ClassFiles/vendor-prefixes/Exercises/`.
4. Enter `npm init`.
5. Choose appropriate values for each prompt (you might name the package "autoprefixer", for example); the defaults will work fine.
6. Check to ensure that the file `package.json` was created in the `Exercises` directory.
7. Add Gulp to the project: from the command line, enter `npm install --save-dev gulp`. This will add a dependency in `package.json`.
8. Install the needed libraries by entering the following from the command line:
   - npm install --save-dev gulp-postcss
   - npm install --save-dev autoprefixer
   - npm install --save-dev gulp-sourcemaps

   (Note that you can also install these all in a single command by listing them with a space between.)
9. We'll add one more item to our task: let's use `clean-css (https://github.com/jakubpawlowicz/clean-css)`, another popular library, to minify our CSS: clean-css will remove extraneous whitespace, compress identifiers, and remove unnecessary definitions, giving us a smaller (and thus faster to download) CSS file. We'll use the gulp plugin `gulp-clean-css (https://github.com/scniro/gulp-clean-css)`. Enter the following from the command line: `npm install gulp-clean-css --save-dev`.

10. Create file `gulpfile.js` in the `Exercises` directory.
11. Copy the contents of `gulpfile.js` from our earlier demo example into the `gulpfile.js` you just created.
12. Edit `gulpfile.js` to add the line `.pipe(cleanCSS())` just after the existing line `.pipe(postcss([ autoprefixer() ]))`; this will perform the minification of the CSS using cssnano.
13. Create a directory structure to mimic that of our earlier demo: `src` contains directory `css` in which goes the CSS file we will edit; `assets` contains directory `css` into which will go our generated (autoprefixed and minified) CSS.
14. Create a CSS file in `src/css` and add some CSS to it. (We copied some CSS from an earlier lesson in this course.)
15. From the command line, enter `gulp autoprefixer`
16. Inspect the generated CSS file in `assets/css` to see the results.

Solution: vendor-prefixes/Solutions/src/css/style.css

```
.container {
 width:300px;
 height:300px;
 margin-right:10px;
 position:relative;
 perspective:1000px;
 float:left;
}
#cube1:hover .cube {
 /* show back face */
 transform: translateZ(-150px) rotateX(-180deg);
}
#cube2:hover .cube {
 /* show right face */
 transform: translateZ(-150px) rotateY(-90deg);
}
#cube3:hover .cube {
 /* show bottom face */
 transform: translateZ(-150px) rotateX(-90deg);
}
.cube {
 transition-duration:1s;
 position: absolute;
 width: 100%;
 height: 100%;
 transform-style:preserve-3d;
 transform: translateZ(-150px);
}
.cube div {
 width: 298px;
 height: 298px;
 display: block;
 position: absolute;
 border: 1px solid black;
 line-height: 298px;
   font-size: 4em;
   font-weight: bold;
   text-align: center;
   color:white;
}
.cube .front {
 background-color:hsla(355, 37%, 39%, 0.8);
 transform: rotateY(0deg) translateZ(150px);
}
```

```
45.   .cube .back {
46.    background-color:hsla(355, 77%, 52%, 0.8);
47.    transform: rotateX(180deg) translateZ(150px);
48.   }
49.   .cube .right {
50.    background-color:hsla(206, 12%, 26%, 0.8);
51.    transform: rotateY(90deg) translateZ(150px);
52.   }
53.   .cube .left {
54.    background-color:hsla(210, 100%, 50%, 0.8);
55.    transform: rotateY(-90deg) translateZ(150px);
56.   }
57.   .cube .top {
58.    background-color:hsla(154, 100%, 13%, 0.8);
59.    transform: rotateX(90deg) translateZ(150px);
60.   }
61.   .cube .bottom {
62.    background-color:hsla(345, 100%, 25%, 0.8);
63.    transform: rotateX(-90deg) translateZ(150px);
64.   }
65.   footer {
66.    clear:both;
67.   }
```


Code Explanation

Here's our initial CSS file (taken from the the "Transitions and Transforms" lesson).

And you should find the generated CSS in `vendor-prefixes/Solutions/assets/css/style.css`. Note that vendor prefixes have been added and the file is compressed into a single line.

Conclusion

In this lesson, you have learned:

- How to use various tools to write cross-browser-correct CSS vendor prefixes.

LESSON 9
Embedding Media

Topics Covered

☑ How to use the HTML5 video tag to publish video.

☑ How to style video responsively.

☑ How to apply 2D transforms and other styles to the video tag.

Introduction

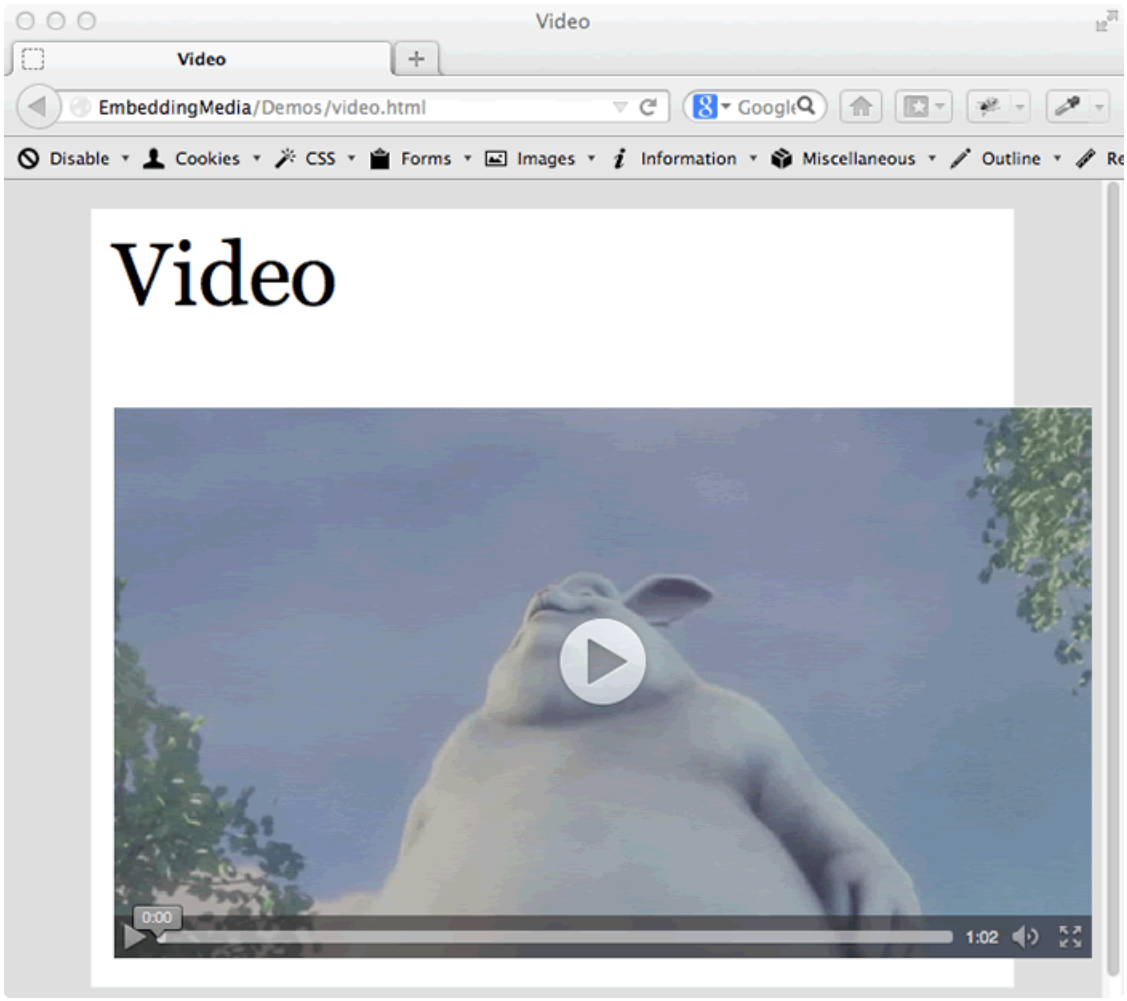
HTML and CSS make it easy to publish video that plays on all devices - we'll look at some best practices for making video look great.

9.1. Embedding Media

Nowadays, it's pretty easy to upload video from your phone to Youtube or Vimeo and embed on our sites. But housing your videos on YouTube or Vimeo isn't always a perfect option. Worries over intellectual property rights (others can embed my videos), loss of traffic (visitors view my videos on YouTube instead of my site), professionalism (worries that visitors might think my site less professional if videos are embedded), and other concerns might dictate that you host your videos on your own site.

The HTML5 `video` tag, along with appropriate choices for video file format, makes relatively easy the process of presenting videos to be playable regardless of the viewing device. We can apply a variety of styles to the video tag to achieve some cool effects.

First, let's look at the `video` tag. Open `ClassFiles/embedding-media/Demos/video.html` in a code editor and in a browser. We show the video with the HTML5 video tag - this works great on recent-model browsers, including mobile devices. Given that video is, like images, a fixed-width media, we get a problem when the video becomes too wide for its container:



(c) Copyright 2008, Blender Foundation / `peach.blender.org/` (`https://peach.blender.org/`), used with permission per `peach.blender.org/about/` (`https://peach.blender.org/about/`)

We can address this with CSS media queries. Here's the markup for the `<video>` tag:

```
<video width="480" height="270" controls poster="poster.png">
  <source type="video/mp4" src="http://clips.vorwaerts-gmbh.de/big_buck_bun ↵↵
ny.mp4"></source>
  <source type="video/webm" src="http://clips.vorwaerts-gmbh.de/big_buck_bun ↵↵
ny.webm"></source>
  <source type="video/ogg" src="http://clips.vorwaerts-gmbh.de/big_buck_bun ↵↵
ny.ogv"></source>
  <p>Your browser does not support the video tag</p>
</video>
```

Code Explanation

The `width` and `height` attributes set the width and height, of course. We'll address the video-too-wide issue by setting the CSS `max-width` of the video element to ensure a responsive handling of width. The presence of the `controls` attribute dictates that the player will show controls ("play", "pause", etc.) The `poster` attribute is the image to show on the player before video playing starts.

The `video` tag encloses a series of `source` tags, which specify alternative video files from which the browser may choose, depending on the media types or codes the browser supports.

In the example above, we offer browsers three options to pick from: `.mp4`, `.webm`, and `.ogg` versions of our video. Note that we're using open-source videos from the `Peach open movie project (https://peach.blender.org/)` - we kindly appreciate their sharing.

Last, the content inside the `video` tag but outside the `source` tags is what users will see if their browser does not support the `video` tag.

Open up `ClassFiles/embedding-media/Demos/video.html` to see the page or view the code.

We style the video using `max-width:100%` to ensure that the element scales with the device width, getting narrower if its container shrink smaller than its (the video's) width:

```
video {
  max-width: 100%;
  height: auto;
}
```

9.2. Video Formats

Specifics about video file formats is a bit beyond the scope of this course. What we really need is a list of best formats to use and some information on how to convert to those formats.

Flash video, often the format in which embedded video is presented for desktop browsers, generally won't work on mobile devices. Instead, we'll concentrate on three formats:

Video File Formats

Format	File Extension	Notes
MPEG 4	`.mp4`	Based on Apple older QuickTime format; supported natively by Chrome, Safari, and Internet Explorer 9+
Ogg	`.ogv`, `.ogg`	Open source; unencumbered by patents; supported natively by Firefox >3.5, Opera >10.50, Chrome >3.0
WebM	`.webm`	Supported natively by Chrome, Firefox, and Opera

Thus, to publish video - specifically, video able to be viewed on different desktop and smartphone browsers - we:

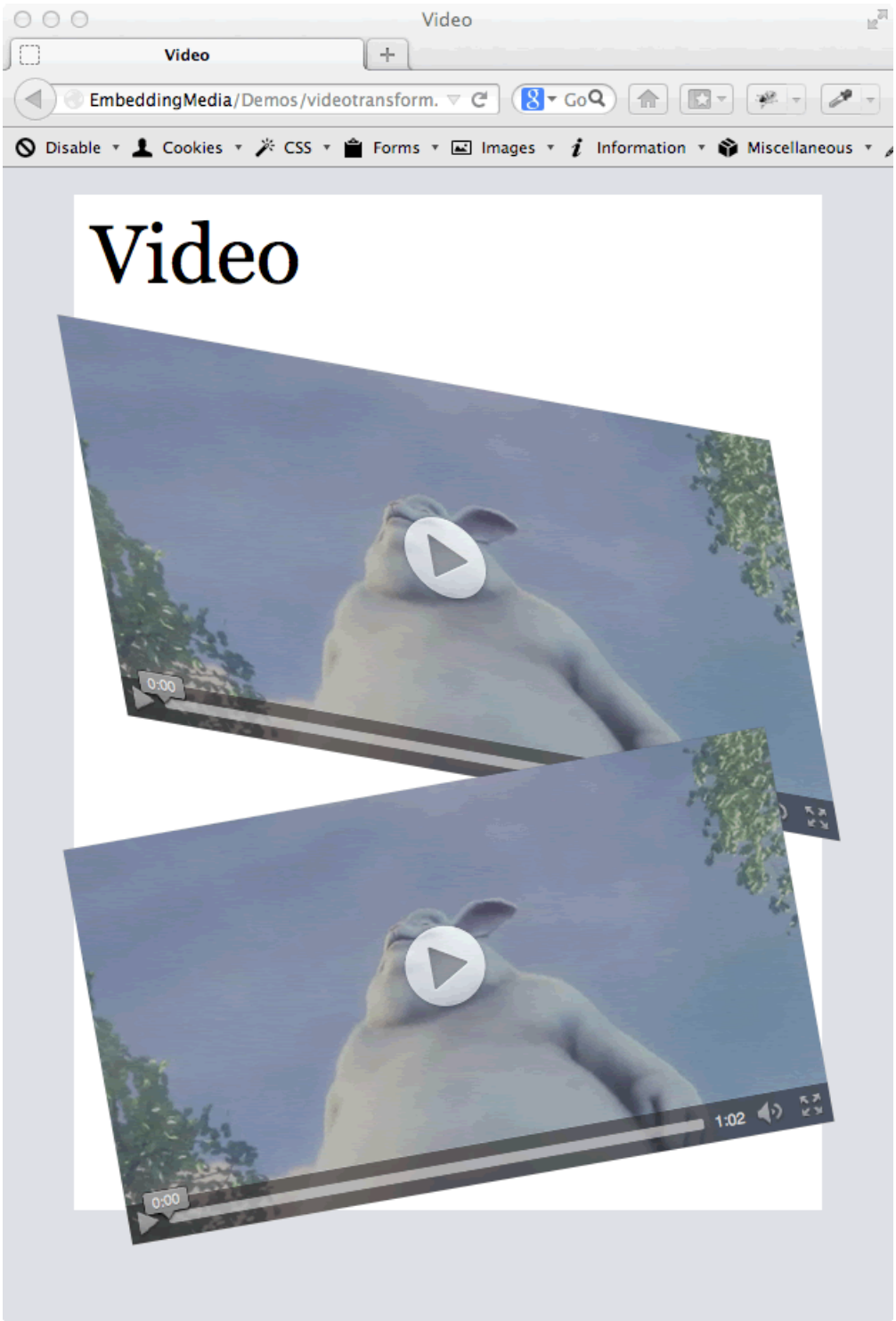
1. Take our original video file (the raw file from the camera).
2. Convert it to each of the file formats above.
3. Publish with the `video` tag.

As of this writing, Chrome and Firefox desktop browsers and Chrome and UC Browser on Andoid support the WebM format. Some developers find that using WebM in addition to mp4 is better for desktop users as the file tends to be smaller.

The user sees a video player with their browser - whether it be Safari on an iPhone, Internet Explorer on a Windows PC desktop, or anything else - picking the video format that works best.

9.3. Styling Video

We can apply many CSS styles to video, including (for those browsers that support them) 2D transforms:



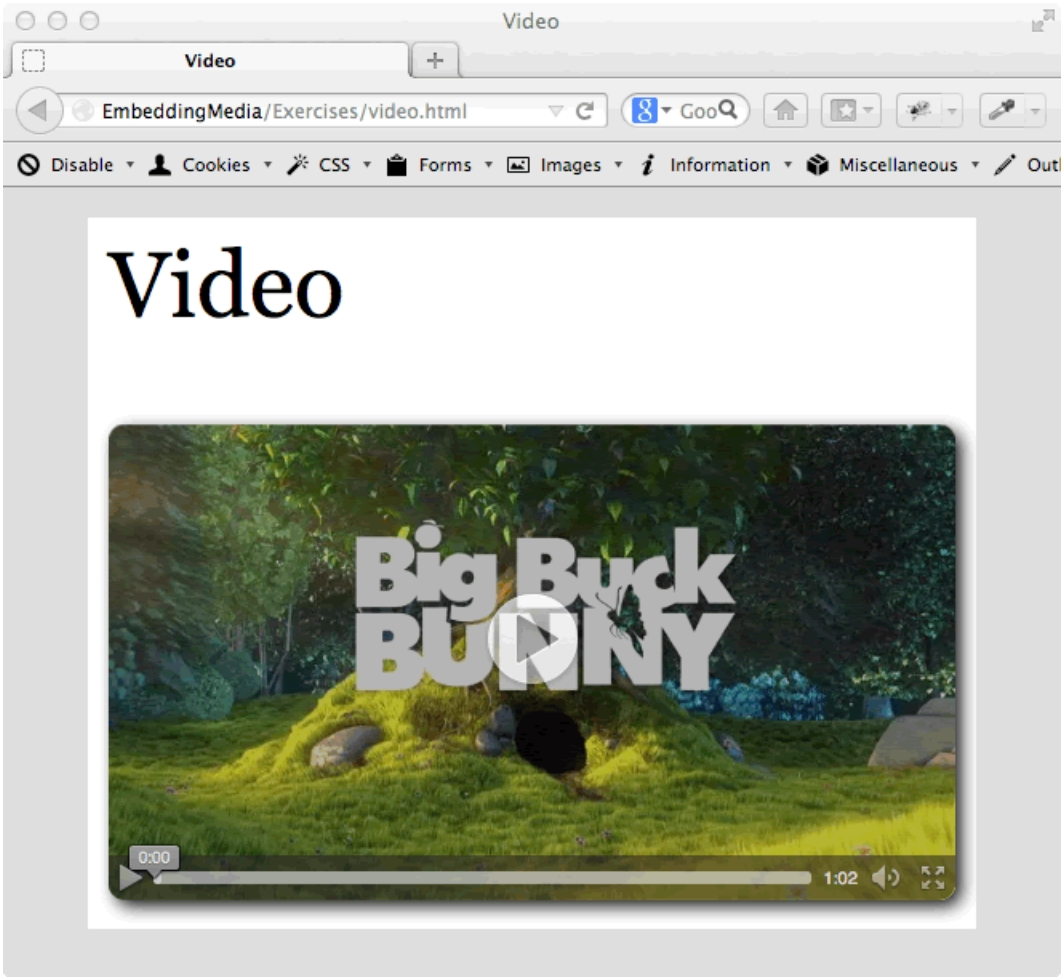
(c) Copyright 2008, Blender Foundation / peach.blender.org/ (https://peach.blender.org/), used with permission per peach.blender.org/about/ (https://peach.blender.org/about/)

The top video is skewed with the statement `transform: skew(10deg, 10deg)`. The bottom video is rotated by counter-clockwise by 10 degrees with the statement `transform: rotate(-10deg)`. Check out `ClassFiles/embedding-media/Demos/videotransform.html` to see the page in action.

Exercise 21: Styling Video

10 to 20 minutes

In this exercise, you will add some styles to the `<video>` tag, incorporating rounded corners and drop shadowing:



(c) Copyright 2008, Blender Foundation / `peach.blender.org/` (`https://peach.blender.org/`), used with permission per `peach.blender.org/about/` (`https://peach.blender.org/about/`)

1. Open `ClassFiles/embedding-media/Exercises/video.html` in a code editor and in a browser to view the page.
2. Add CSS to style the `<video>` element:

- Use `max-width:100%` to ensure the video shrinks to fit the space available to it.
- Add rounded corners to the video.
- Add some drop shadow to the video.

Solution: embedding-media/Solutions/video.html

```
1.  <!DOCTYPE html>
2.  <html>
3.  <head>
4.   <meta charset="utf-8">
5.   <title>Video</title>
6.   <meta name="viewport" content="width=device-width">
7.   <link rel="stylesheet" href="../Shared/reset.css">
8.   <link rel="stylesheet" href="../Shared/style.css">
9.   <style type="text/css">
10.   video {
11.    max-width: 100%;
12.    height: auto;
13.    border-radius: 10px;
14.    box-shadow: 4px 4px 10px black;
15.   }
16.  </style>
17. </head>
18. <body>
19.  <div id="main">
20.   <h1>Video</h1>
21.   <video controls poster="poster.png">
22.    <source type="video/mp4" src="http://clips.vorwaerts-gmbh.de/big_buck_bun ↵↵
          ny.mp4"></source>
23.    <source type="video/webm" src="http://clips.vorwaerts-gmbh.de/big_buck_bun ↵↵
          ny.webm"></source>
24.    <source type="video/ogg" src="http://clips.vorwaerts-gmbh.de/big_buck_bun ↵↵
          ny.ogv"></source>
25.    <p>Your browser does not support the video tag</p>
26.   </video>
27.  </div>
28. </body>
```


Code Explanation

We use `max-width:100%` and `height:auto`, telling the `<video>` element to shrink up if its container is smaller than the video's physical width allows.

We use `border-radius: 10px` and `box-shadow: 4px 4px 10px black` to add rounded corners and drop shadowing, respectively, to the video.

Conclusion

In this lesson, you have learned:

- How to use the HTML5's video tag to publish video viewable on more browsers.
- How to style video responsively, so as not to break the page's design.
- How to apply 2D transforms and other styles to the video element.

LESSON 10
Accessibility Features

Topics Covered

☑ How to use CSS to specify volume, pitch, voice, and other aspects of screen readers employed to render content aurally.

Introduction

Screen readers and other assistive technology are used to “speak” web content for users who have limited or no vision; the CSS Speech Module offers a way to control the rendering of our content as speech.

10.1. Accessibility Features

❖ 10.1.1. Web Accessibility

Many users employ assistive technologies to improve their access to web content; screen readers, for instance, speak aloud the text (and alt text, for images) on a given page, giving folks with no or impaired vision the same access to content as that of sighted users. For years, we as web developers and designers have - or should have - coded our pages to be friendly to these assistive technologies. These coding tactics included ensuring that all image content included appropriate alt text, using `<label>` tags and `for` attributes for form elements, and adding “skip nav” links to allow folks using screen readers to more easily navigate pages. The W3C’s `Web Accessibility Initiative (http://www.w3.org/WAI/)` site is a great place to find more information.

❖ 10.1.2. The CSS Speech Module

CSS’s Speech Module is a relatively recent addition to the set of tools we can use to render content in a manner friendly to those using screen readers and other accessibility tools. The module offers, as the `W3C’s Speech Module site (http://www.w3.org/TR/css3-speech/)` declares, ways to define “aural properties that enable authors to declaratively control the rendering of documents via speech synthesis, and using optional audio cues.”

Unfortunately, at the time of this writing, we have no easy way to test these properties; no current browser or free/low-cost screen reader offers a way for you to test out these properties. Mac's VoiceOver, Opera's Voice tool, Firefox's Fire Vox, and other technologies - all of which are useful in their own right - either do not yet adhere to the CSS Speech Module properties, do not work with current operating systems/browser versions, or both.

So, with our apologies, we'll give a brief overview of the Speech Module properties but no demos or exercises. Hopefully, the module will grow quickly in popularity and adoption.

`voice-volume`

The `voice-volume` property suggests a volume for aural rendering of the designated content, with values wither in absolute decibels or one of the following:

- `silent`
- `x-soft`
- `soft`
- `medium` (the default)
- `loud`
- `x-loud`

Note that a value of `silent` suggests to screen readers to render, silently, the text content aurally - thus taking up the duration (of silence) that would have been used to speak the content.

```
<p style="voice-volume:soft">This text is soft</p>
      <p style="voice-volume:x-loud">This text is extra loud</p>
      <p style="voice-volume:-3.0dB">This text should be three decibels softer than the
 default</p>
```

`voice-balance`

The `voice-balance` property suggests the distribution of the sound across a left-right access, similar to the "fade" control on a stereo. Values can be a number in the range -100 to 100, inclusive, with -100 representing "left", 0 representing "center", and 100 representing "right". Values can also be:

- `left` (same as `-100`)

- `center` (same as `0`)
- `right` (same as `100`)
- `leftwards` (decrease inherited value by 20)
- `rightwards` (increase inherited value by 20)

speak

The `speak` property determines whether or not to render text aurally, with possible values:

- `auto`
- `none`
- `normal`

A value of `auto` equates to `none` if the element to which it is applied has a `display` value of `none`.

Note that `speak:none` and `voice-volume:silent` correspond loosely to `display:none` and `visibility:hidden`. In the first instances (`speak:none` and `display:none`), the element does not take up any duration or space; in the second (`voice-volume:silent` and `visibility:hidden`), the element does take up duration/space, but isn't heard or seen.


```
<p style="speak:none">This text won't be rendered aurally</p>
      <p style="speak:normal">This text will be rendered aurally</p>
```

speak-as

The `speak-as` property determines the manner in which text gets rendered aurally: whether to read "normally," to spell out individual characters or digits, and how to handle punctuation. Valid values are:

- `normal` (punctuation rendered naturally as appropriate pauses)
- `spell-out` (spells out text one letter at a time)
- `digits` (speak numbers one digit at a time)
- `literal-punctuation` (speak the name of each punctuation character)
- `no-punctuation` (punctuation is neither spoken nor rendered as pauses)

```
<p style="speak-as:normal">This text would be rendered normally</p>
      <p style="speak-as:spell-out">USA</p>
      <p style="speak-as:digits">twelve 23</p>
      <p style="speak-as:literal-punctuation">I am happy - really happy.</p>
      <p style="speak-as:no-punctuation">One. Two. Three.</p>
```

In the examples above, the first text would be read normally. The second example would be rendered aurally as “you-es-ay”. The third would be rendered aurally as “one two two three”. The fourth would be read as “I am happy dash really happy period”. The last would be read without pauses for the periods.

The Aural Box Model

The Speech Module uses a conceptual formatting model for aural media based on a sequence of sounds and silence that render in a nested fashion. As a speech device reads each element on the page, the aural box model defines the order of properties as follows:

- `pause-before`
- `cue-before`
- `rest-before`
- [the element]
- `rest-after`
- `cue-after`
- `pause-after`

The aural formatting model is similar in some ways to CSS’s box model: `rest` corresponds to `padding`, `cue` corresponds to `border`, and `pause` corresponds to `margin`.

`pause`

The `pause` property allows us to specify pauses before and/or after a given element; alternately, we can use `pause-before` or `pause-after` instead of the shorthand `pause` property. `pause` takes one or two values - if both are given, they set the before and after properties, respectively; if just one is given, it is applied to both properties.

`pause`, `pause-before`, and `pause-after` accept a value of time duration (specified as seconds or milliseconds - e.g., “2s” or “100ms”) or one of the following values:

- `none` (no prosodic break is produced by the speech processor)
- `x-weak`
- `weak`
- `medium`
- `strong`
- `x-strong`

The weak/medium/strong values are implementation dependent.

The `rest` property - a shorthand for specifying both `rest-before` and `rest-after` in one fell swoop - works exactly the same as `pause`, and takes the same range of possible values.

The cue property is used to specify the rendering of an "auditory icon" - a bell sound, perhaps, or chime. The property accepts a URI, specifying the path to the desired cue sound file, and (optionally) the decibel level (relative to the current `voice-volume` value) for the cue sound.

Here's an example showing the use of `pause`, `cue`, and `rest`:


```
<p style="pause:20ms 3s; cue:url(bell.wav) url(chime.au);rest:20ms 2s">This text has a 20-millisecond pause before it, the cue bell.wav before it, and a 20-millisecond rest before it; it has a 2-second rest after it, the cue chime.au after it, and a 3-second pause after it</p>
```

`voice-family`

`voice-family` is similar to `font-family`: it specifies the voice to use to read the given content. Valid values include:

- A named voice
- A gender: `male`, `female`, or `neutral`
- An age: `child`, `young`, or `old`

```
<p style="voice-family:young female">This text will be read by a young female voice</p>
```

voice-rate

The `voice-rate` property controls the rate at which content is rendered aurally. The property can be specified by keyword or by percentage, with valid values of:

- `normal` (the default)
- `x-slow`, `slow`, `medium`, `fast` or `x-fast`
- a percentage, relative to the given element's inherited `voice-rate`: 50% specifies one-half of the inherited rate; 200% specifies twice the inherited rate

```
<p style="voice-rate:x-fast">This text will be read quite quickly</p>
```

voice-pitch

The `voice-pitch` property specifies the baseline pitch of the generated speech output; note that the pitch depends upon the `voice-family` property's value (a male voice would have a lower pitch than that of a female voice) and on the speech synthesis processor in use. Valid values include absolute or relative frequency, percentage, or keyword:

- `100Hz` specifies 100 Hertz higher than the inherited value
- `-50Hz` specifies 50 Hertz lower than the inherited value
- `210Hz absolute` specifies a pitch of 210 Hertz
- `20%` or `-30%` specifies a pitch 20% higher or 30% lower, respectively, than the inherited value
- keyword `x-low`, `low`, `medium`, `high`, `x-high`

```
<p style="voice-pitch:110Hz absolute">This text should be read at a baseline pitch of
110 Hertz</p>
```

voice-range

The `voice-range` property specifies the variability in baseline pitch; as the `W3C specification (http://www.w3.org/TR/css3-speech/#voice-range)` states, "how much the fundamental frequency may deviate from the average pitch of the speech output." As with `voice-pitch`, the implementation of `voice-range` is dependent upon both the current voice

and the speech processor in use. Values can be specified as relative or absolute frequency, percentage, or keyword:

- `90Hz` specifies 90 Hertz higher than the inherited value
- `-40Hz` specifies 40 Hertz lower than the inherited value
- `180Hz absolute` specifies a pitch of 180 Hertz
- `10%` or `-40%` specifies a pitch 10% higher or 40% lower, respectively, than the inherited value
- keyword `x-low`, `low`, `medium`, `high`, `x-high`

```
<p style="voice-range:x-high">This text should be read at with an extra-high voice range</p>
```

voice-stress

The `voice-stress` property defines the strength of emphasis which, per the `W3C's specification (http://www.w3.org/TR/css3-speech/#voice-stress)`, "is normally applied using a combination of pitch change, timing changes, loudness and other acoustic differences. The precise meaning of the values therefore depend on the language being spoken." Valid values are:

- `normal` - the default
- `none` - prevents synthesizer from emphasizing text which would normally be emphasized
- `moderate`
- `strong`
- `reduced`

```
<p style="voice-stress:strong">The text should be read with extra emphasis!</p>
```

voice-duration

The `voice-duration` property defines how long it should to render the specified element's content; this does not include `pause`, `cue`, and `rest` settings. Valid values are `auto` or a time duration in absolute time units.

```
<p style="voice-duration:5s">This text should be read over the course of five seconds</p>
```

Conclusion

In this lesson, you have learned:

- How to control aural presentations of content through the CSS Speech Module.

LESSON 11
Media Queries

Topics Covered

☑ What are media queries?

☑ How to use media queries.

☑ Using media queries to target different browser widths.

☑ Using media queries to target different device widths.

Introduction

CSS media queries offer fine control over style rules, allowing us to target differing browser widths and device widths to change the manner in which we present content.

11.1. Media Queries

❖ 11.1.1. Targeting Widths and Devices with CSS Media Queries

Responsive web design - that is, the process of adapting the presentation and layout of web pages to better fit the user's device size and other properties - is a given in web design these days. The proliferation of smartphones, tablets, and other non-desktop devices means that more of the traffic to our sites comes from iPhones, Android phones, Kindles, iPads and other tablets, and other devices. For better or for worse, a design of one size no longer fits all - we must adapt our designs to fit smaller screens.

You may already be familiar with CSS media types - a CSS2 specification that offered a way to include different style sheets based on the media type (screen, print, etc.) being employed. A print-specific style sheet, for example, might present a page without background images, with a print-friendly font face, and with margins appropriate for printing.

CSS Level 3 extends this concept with the media query - a way to apply CSS rules selectively based on both the type of media and the physical properties of the device (browser, phone) being used to access the page.

Each media query comprises a media type (e.g., “screen”) and zero or more logical expressions - a condition evaluating to true or false based on the conditions of particular media features. We can test our user’s device for screen width, device width, orientation (“portrait” or “landscape”), and other features.

Media Query Features

Feature	Possible Values	Min/Max?	Explanation
`color`	int	yes	bits per color component
`color-index`	int	yes	number of entries in color lookup table
`device-aspect-ratio`	int/int	yes	aspect ratio
`device-height`	length (pixels)	yes	height of the output device
`device-width`	length (pixels)	yes	width of the output device
`grid`	int	no	true for a grid-based device
`height`	length (pixels)	yes	height of the rendering surface
`monochrome`	int	yes	bits per pixel in a monochrome frame buffer
`resolution`	“dpi” or “dpcm”	yes	resolution
`scan`	“progressive” or “interlaced”	no	scanning process of “tv” media types
`width`	length (pixels)	yes	width of the rendering surface

Let’s look at a quick example.

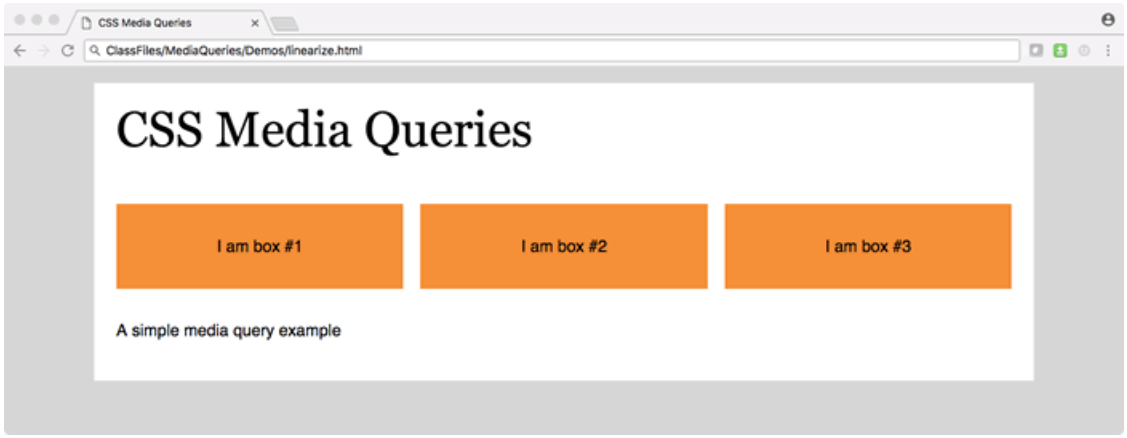
Demo 11.1: media-queries/Demos/linearize.html

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
 <title>CSS Media Queries</title>
 <meta name="viewport" content="width=device-width">
 <link rel="stylesheet" href="../Shared/reset.css">
 <link rel="stylesheet" href="../Shared/style.css">
 <style>
  .container {
   display: grid;
   grid-template-columns: auto auto auto;
   grid-column-gap: 20px;
  }
  .box {
   padding:40px 2%;
   background-color:hsl(30, 90%, 60%);
   text-align:center;
   margin-bottom:10px;
  }
  .box p {
   margin-bottom:0;
  }
  @media screen and (max-width: 768px) {
   /* CSS for browsers smaller than 768 pixels: */
   h1 {
    font-size:1.5em;
   }
   .container {
    grid-template-columns: auto;
   }
   .box {
    border:1px solid hsl(0,50%, 0%);
   }
  }
  footer {
   clear:left;
  }
 </style>
</head>
<body>
 <div id="main">
  <h1>CSS Media Queries</h1>
  <div class="container">
```

```
45.      <div class="box">
46.       <p>I am box #1</p>
47.      </div>
48.      <div class="box">
49.       <p>I am box #2</p>
50.      </div>
51.      <div class="box">
52.       <p>I am box #3</p>
53.      </div>
54.     </div>
55.     <footer>
56.      <p>A simple media query example</p>
57.     </footer>
58.    </div>
59.   </body>
60.   </html>
```

Code Explanation

Open `ClassFiles/media-queries/Demos/linearlize.html` in a desktop browser. At a typical browser width (about 850 pixels in this first screenshot), the page looks like this:



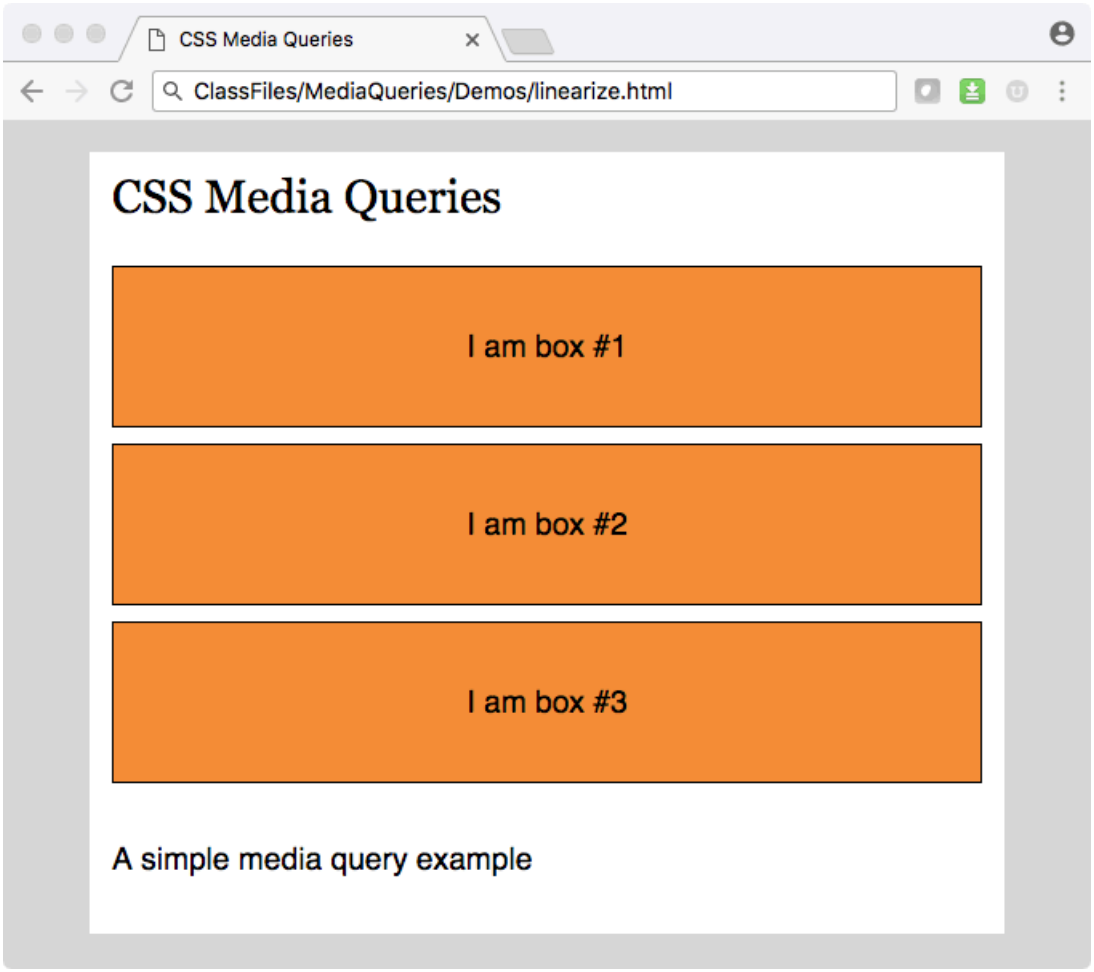
We style the three boxes using grid layout (we'll cover this in more detail later in the course). The styles we give the `.container` element:

```
.container {
  display: grid;
  grid-template-columns: auto auto auto;
  grid-column-gap: 20px;
}
```

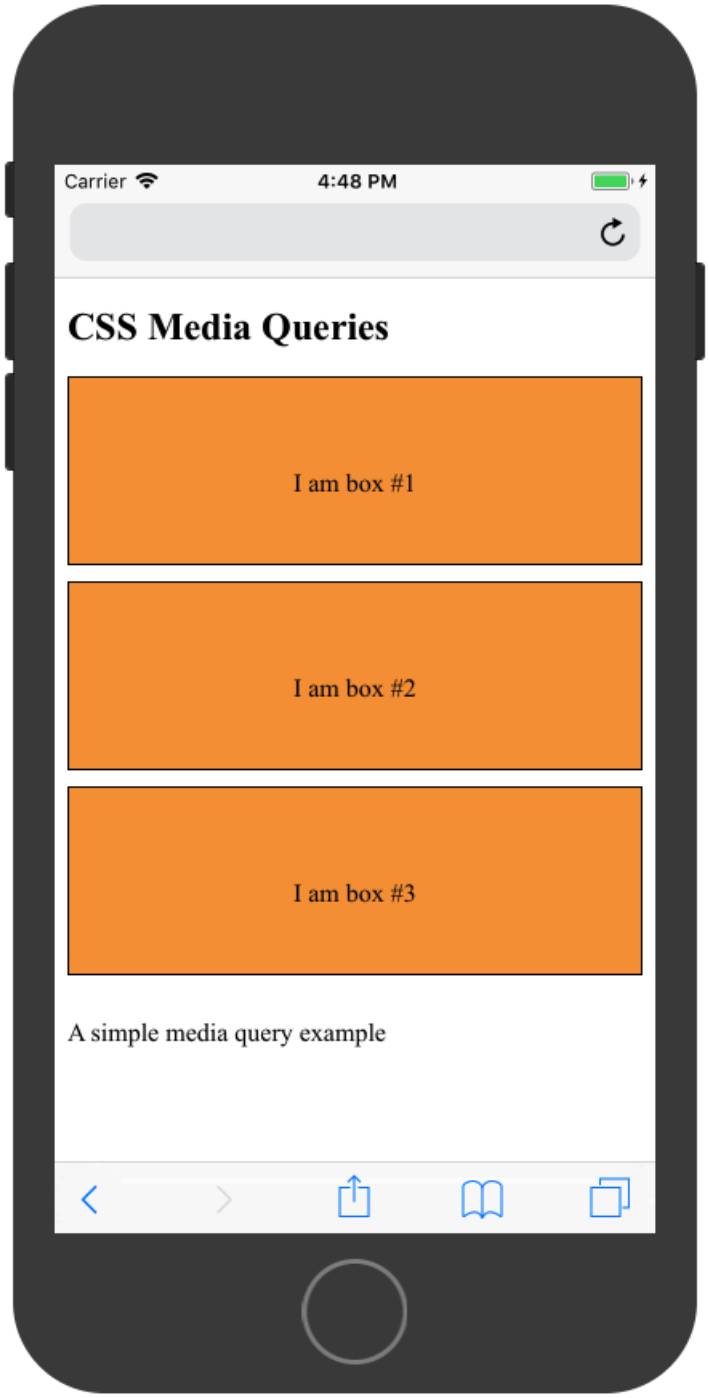
set the element's `display` to `grid` and set the child elements (each of the "box" `div`s to be 1/3 of the total width, with a 20px gutter between the elements.

If we view this same page on a smartphone or tablet, or if we resize the browser to a narrower width, the page might not work so well: the three-across arrangement of the `<div>`s would be pretty small if viewed on an iPhone, for instance.

To address this, we can turn to a common design pattern: take a horizontal layout, like we have here, and linearize it - arrange the elements vertically instead of horizontally. Notice the media query: the CSS statement `@media screen and (max-width: 768px)`. This targets browser widths of 768 pixels or less - meaning that, if we were to drag the browser to a narrower width (narrower than 768 pixels), the page would look like this:



Even better, the page now works well on a smartphone - here's a view from an iPhone:



The important code here, as mentioned above, is the media query. For any browser width smaller than 768 pixels, we effect the following changes:

- Size the `<h1>` title a little (`1.5em`) smaller

- Set the grid elements to display full width, instead of three across, and add a border.

As the table of media query features suggests, we can target many aspects of user's devices: the width or height of the browser, the width or height of the device itself, the color depth, the resolution, etc. We can add multiple media queries to target different browser widths, so that folks using a really wide browser get one view, medium-sized browser get another view, and smaller (smartphone, say) widths get yet another view - the same content for each, but optimized for each user's particular viewing environment.

❖ 11.1.2. Responsive Images & the `viewport` Metatag

One challenge in rending responsive designs is images and other fixed-width media. A flexible design - one that linearizes or collapses on small-width browsers or devices - tends to get mucked up by images which, in the absence of a little extra work on our part, stubbornly refuse to shrink up to fit smaller screens. Luckily, CSS makes this pretty easy to handle. The `max-width` property (a CSS2 specification), when set to `100%`, does exactly what we need here - it shrinks the image or element to which it is applied to the width of its container element. Images retain their inherent width when their parent container allows, and shrink (maintaining their aspect ratio) when the parent becomes too small. Here's a quick example:

Demo 11.2: media-queries/Demos/max-width.html

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
 <title>CSS Responsive Images</title>
 <meta name="viewport" content="width=device-width">
 <link rel="stylesheet" href="../Shared/reset.css">
 <link rel="stylesheet" href="../Shared/style.css">
 <style>
  .box {
   padding:5% 2%;
   width:43%;
   margin-right:5%;
   float:left;
   background-color:hsl(30, 90%, 60%);
   text-align:center;
  }
  .box:nth-child(2n+1) {
   margin-right:0;
  }
  #box2 img {
   /* ensure that the image shrinks up when its container is small: */
   max-width: 100%;
  }
  footer {
   clear:left;
  }
 </style>
</head>
<body>
 <div id="main">
  <h1>CSS Responsive Images</h1>
  <div class="box" id="box1">
   <p>Chicago Skyline</p>
   <img src="chicago.jpg" alt="Chicago skyline">
  </div>
  <div class="box" id="box2">
   <p>Chicago Skyline</p>
   <img src="chicago.jpg" alt="Chicago skyline">
  </div>
  <footer>
   <p>CSS <code>max-width</code></p>
  </footer>
 </div>
```

```
45.   </body>
46.   </html>
```

Code Explanation

Two `<div>`s, each containing a 300-pixel-wide photo of the Chicago skyline, float left; each is roughly 1/2 of the width of the containing `<div>`, sized (`width:43%`) by percentage.

At wider browser widths, the images look good - each fits nicely into its respective orange box:

But if we resize the browser, making it more narrow, the fixed width of the images would break the design. We fix one of the images - the right photo (`.box2 img`) gets `max-width: 100%;`; we don't apply the same CSS to the left photo. Note the effect when we view the page on a narrower browser:

Using percentage widths is, in general, a good idea when designing responsively. A fluid (percentage-based) design ensures that the elements shrink or expand to fit the available space. See Ethan Marcotte's `excellent article on A List Apart (http://www.alistapart.com/articles/responsive-web-design/)` for further reading.

The last key tool in the responsive web designer's kit is the `viewport` metatag. The astute observer will note that we've used this through the course so far on all demos and exercises; the effect of this line of code, in the `<head>` of the page, is to scale the content to fit the device. Not CSS, of course, but important to note when designing for phones and tablets.

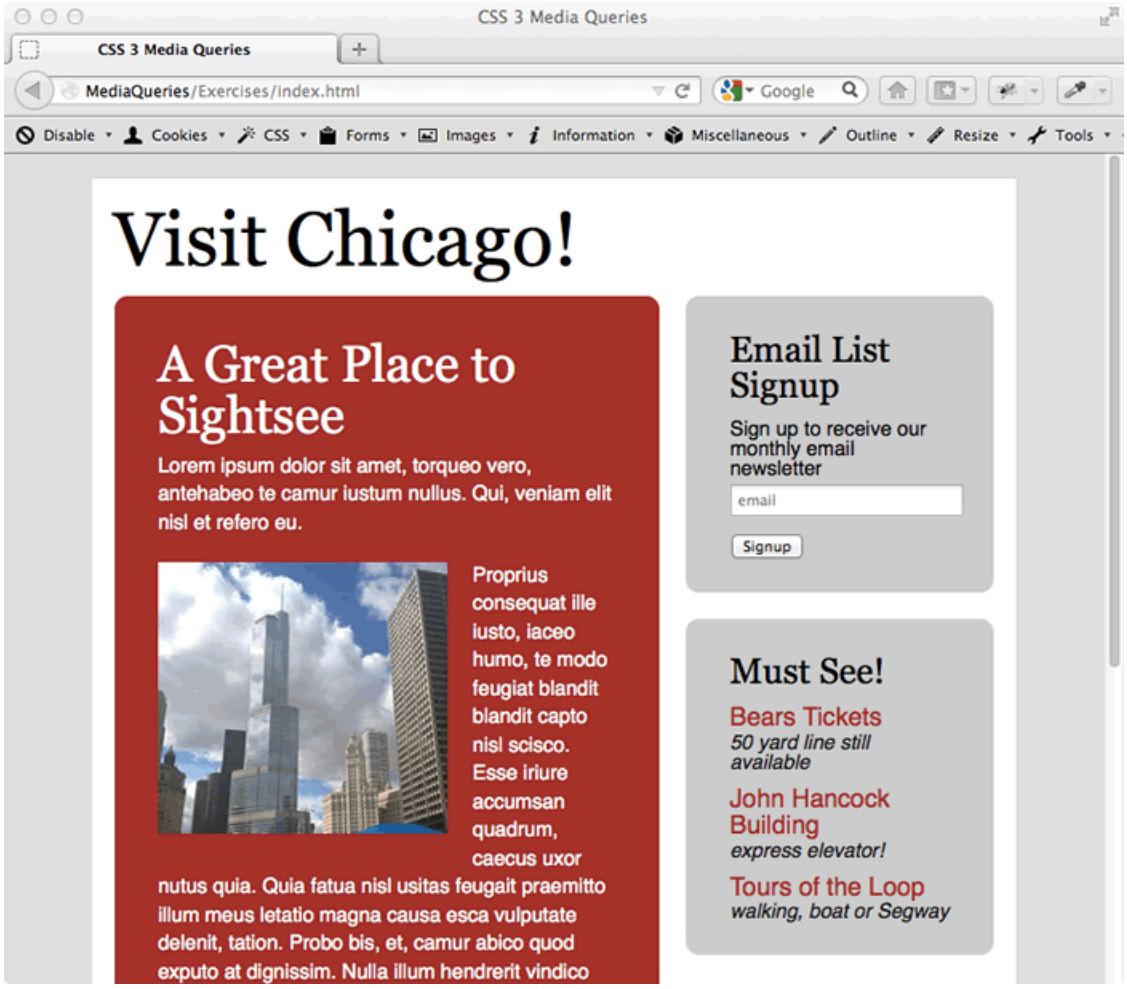
Let's have you try out these concepts in an exercise.

Exercise 22: Responsive Design

25 to 35 minutes

In this exercise, you will render a responsive design, offering three different visual looks for users with smartphones/narrow-width-browsers, for medium-width-browsers, and for wide-width-browsers.

1. Open `ClassFiles/media-queries/Exercises/index.html` in a file editor and in a browser.
2. First, render the design for "medium" width browsers, between 768 and 1200 pixels wide - the content should look as below:



   A. The main content ("A Great Place to Sightsee"), marked up as an `<article>`, should occupy about 60% of the left of the container.

B. The main content should have a dark red background color and white text color.

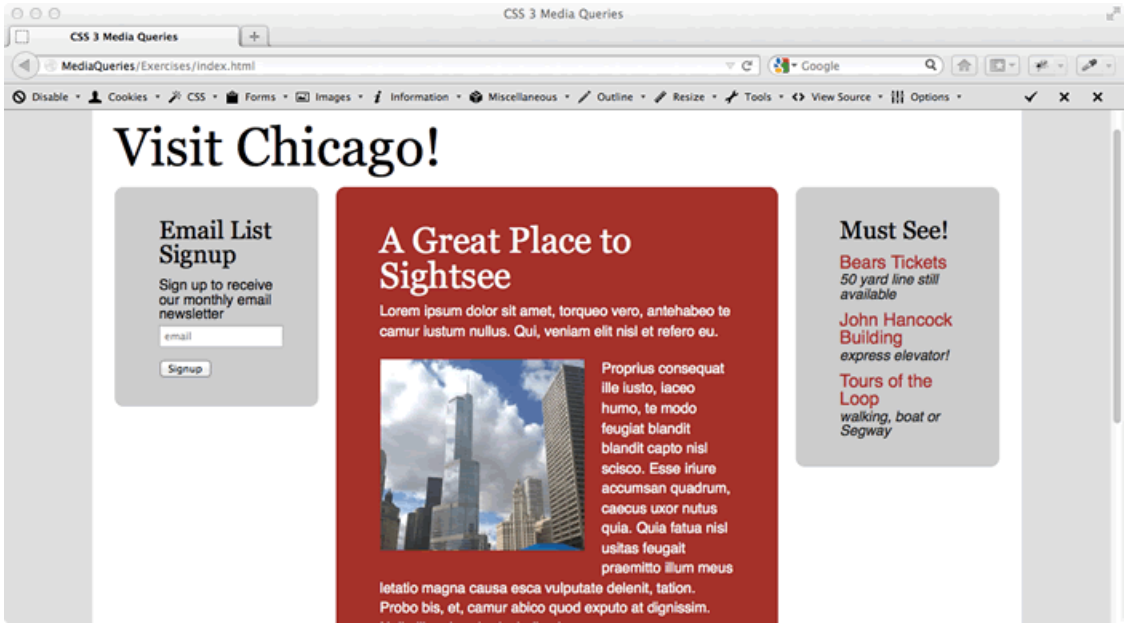
C. Be sure to handle flexible width for any images inside the main content; float the image to wrap text around it.

D. The “Email List Signup” and “Must See!” elements should occupy the right column.

E. Style the “Must See!” link details as shown.

F. All content containers should have rounded corners.

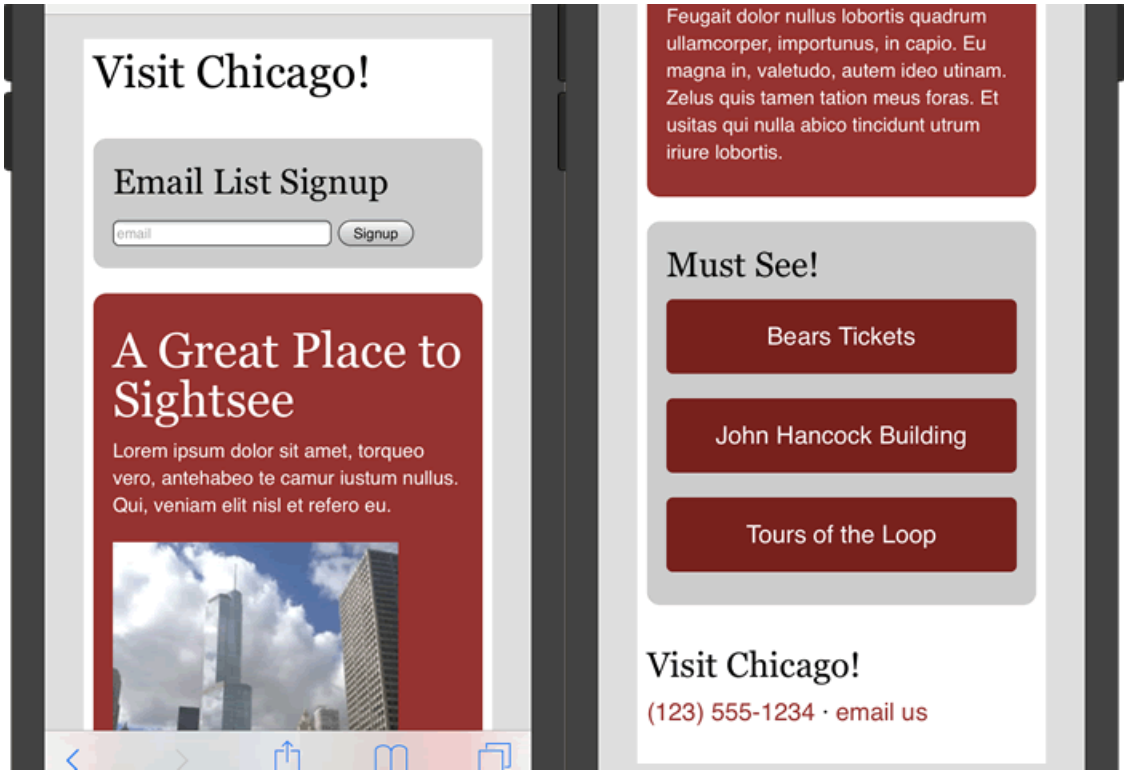
3. In browsers wider than 1200 pixels, the design should change - we’ll now arrange the elements side-by-side, with the email-signup element to the left of the main content:



A. Use a media query with `min-width: 1200px` to target browsers of wide widths.

B. Float the email-signup to the left, and update each of the three main elements’ widths to ensure they fit into the available space.

4. In browsers narrower than 768 pixels (including smartphones), the design should also change - in manner similar to the example we saw earlier, change the presentation of the three main elements to show stacked vertically, instead of floated horizontally:



A. Use a media query with `max-width: 768px` to target browsers of narrow widths/smartphones.

B. Linearize the design, showing “Email List Signup” at top, then “A Great Place to Sightsee”, then “Must See” at bottom; update each element’s widths and margins accordingly.

C. Hide elements classed as “detail” in the email-signup and the “Must See” sections; this is a common design pattern, to hide some content on smaller screens.

D. Make the links in the “Must See” section more button-like: give them a padding, background color, and rounded corners, as shown.

Solution: media-queries/Solutions/index.html

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
 <title>CSS Media Queries</title>
 <meta name="viewport" content="width=device-width">
 <link rel="stylesheet" href="../Shared/reset.css">
 <link rel="stylesheet" href="../Shared/style.css">
 <style>
  /* default styles */
  a {
   color:hsl(0, 60%, 40%);;
  }
  h1 {
   margin-bottom:0.25em;
  }
  article, aside {
   border-radius: 10px;
  }
  article {
   padding:2% 5%;
   width:52%;
   margin-right:3%;
   float:left;
   background-color:hsl(0, 60%, 40%);
   color:#fff;
  }
  article p {
   font-size:0.8em;
   line-height:1.4em
  }
  img.pageimg {
   max-width:100%;
   float:left;
   margin:0 1em 1em 0;
  }
  aside#signup {
   float:right;
   width:25%;
   padding:2% 5%;
   background-color:hsl(0, 0%, 80%);
   margin-bottom:1em;
  }
  aside#signup p.details {
```

```
45.      font-size:0.8em;
46.      margin:0;
47.     }
48.     aside#signup input#email {
49.      width:100%;
50.     }
51.     aside#mustsee {
52.      float:right;
53.      width:25%;
54.      padding:2% 5%;
55.      background-color:hsl(0, 0%, 80%);
56.     }
57.     aside#mustsee ul {
58.      list-style: none;
59.      margin:0;
60.     }
61.     aside#mustsee ul li a {
62.      display:block;
63.     }
64.     aside#mustsee ul li a.details {
65.      font-size:0.8em;
66.      font-style:italic;
67.      color:hsl(0, 0%, 10%);
68.      margin:0.1em 0 0.7em 0;
69.     }
70.     footer {
71.      padding-top:1em;
72.      clear:both;
73.     }
74.
75.     /* styles for small */
76.     @media screen and (max-width: 768px) {
77.      h1 {
78.       font-size:1.8em;
79.       margin-bottom:1em;
80.      }
81.      aside#signup {
82.       /* un-float the #signup element: */
83.       float:none;
84.       width:90%;
85.       margin:0 auto 1em auto;
86.       padding:2% 5%;
87.       background-color:hsl(0, 0%, 80%);
88.      }
89.      aside#signup input#email {
```

```
90.       width:60%;
91.      }
92.      article {
93.       padding:2% 5%;
94.       width:90%;
95.       margin:0 auto 1em auto;
96.       float:none;
97.       background-color:hsl(0, 60%, 40%);
98.       color:#fff;
99.      }
100.     aside#mustsee {
101.      /* un-float the #mustsee element: */
102.      float:none;
103.      width:90%;
104.      padding:2% 5%;
105.      background-color:hsl(0, 0%, 80%);
106.     }
107.     aside#mustsee ul li a {
108.      background-color:hsl(0, 70%, 30%);
109.      border-radius:5px;
110.      padding:1em;
111.      margin-bottom:1em;
112.      text-align: center;
113.      color:hsl(0,50%,100%);
114.     }
115.     .details {
116.      /* don't show details: */
117.      display:none !important;
118.     }
119.    }
120.
121.    /* styles for large */
122.    @media screen and (min-width: 1200px) {
123.     /* we arrange the elements three-across: */
124.     aside#signup {
125.      float:left;
126.      width:13%;
127.      margin-right:2%;
128.      padding:2% 5%;
129.      background-color:hsl(0, 0%, 80%);
130.      margin-bottom:1em;
131.     }
132.     article {
133.      padding:2% 5%;
134.      width:40%;
```

```
135.      margin-right:2%;
136.      float:left;
137.      background-color:hsl(0, 60%, 40%);
138.      color:#fff;
139.     }
140.     aside#mustsee {
141.      float:right;
142.      width:13%;
143.      padding:2% 5%;
144.      background-color:hsl(0, 0%, 80%);
145.     }
146.    }
147.   </style>
148.  </head>
149.  <body>
150.   <div id="main">
151.    <h1>Visit Chicago!</h1>
152.    <aside id="signup">
153.     <h3>Email List Signup</h3>
154.     <p class="details">Sign up to receive our monthly email newsletter</p>
155.     <form action="#" method="post">
156.      <input type="email" name="email" id="email" placeholder="email">
157.      <input type="submit" value="Signup">
158.     </form>
159.    </aside>
160.    <article>
161.     <h2>A Great Place to Sightsee</h2>
162.     <p>Lorem ipsum dolor sit amet, torqueo vero, antehabeo te camur iustum nullus.
            Qui, veniam elit nisl et refero eu.</p>
163.     <img src="chicago.jpg" alt="Chicago Skyline" class="pageimg">
164.     <p>Proprius consequat ille iusto, iaceo humo, te modo feugiat blandit blandit
            capto nisl scisco. Esse iriure accumsan quadrum, caecus uxor nutus
           quia. Quia fatua nisl usitas feugait praemitto illum meus letatio magna
            causa esca vulputate delenit, tation. Probo bis, et, camur abico quod
            exputo at dignissim. Nulla illum hendrerit vindico importunus, esse
           tamen valetudo duis vel occuro validus exerci.</p>
-------Line 165 Omitted-------
166.    </article>
167.    <aside id="mustsee">
168.     <h3>Must See!</h3>
169.     <ul>
170.      <li><a href="#">Bears Tickets</a><a href="#" class="details">50 yard line
           still available</a></li>
171.     <li><a href="#">John Hancock Building</a><a href="#" class="details">express
            elevator!</a></li>
```

```
172.      <li><a href="#">Tours of the Loop</a><a href="#" class="details">walking,
               boat or Segway</a></li>
173.     </ul>
174.    </aside>
175.    <footer>
176.     <h3>Visit Chicago!</h3>
177.     <p>(123) 555-1234 &middot; <a href="#">email us</a></p>
178.    </footer>
179.   </div>
180.  </body>
181.  </html>
```

Code Explanation

For the medium-width browser design - that is, the default style - we style the page to float the `<article>` left and the two `<aside>`s to the right. The `article` gets a total width of 62% (52% plus right and left padding of 5% each), along with a right margin of 3%. The two asides take up the remaining width. We style the other aspects of each element to match the specification as shown in the screenshot.

For the large-width browser design, we use `@media screen and (min-width: 1200px)`. We now float `aside#signup` left (instead of right) and give each of the elements a slightly smaller width, to accommodate the fact that there are three (instead of two, in the default case) elements taking up the horizontal width.

Last, for the small-width/mobile design, we use `@media screen and (max-width: 768px)` to target widths less than 768 pixels. Each code element is unfloated (`float:none`) and given a width of 90%, along with some padding. We use `display:none` to hide the email-list-signup explanatory text (`p.details`) and the extra info (`a.details`) for each of the three events; note that we need to use `!important` for the latter, as `display:block` is applied to an id for these links, and would thus take precedence if we didn't use `!important`.

Conclusion

In this lesson, you have learned:

- How to use media queries to modify CSS rules selectively.
- How to target different browser-width breakpoints with media queries.
- How to target different device viewport widths with media queries.

LESSON 12
Sass

Topics Covered

☑ How CSS preprocessors like Sass make developing CSS easier and faster.

☑ How to install and use Sass.

Introduction

Sass offers features not found in CSS - using Sass's variables, nesting, inheritance, and other components make writing and maintaining CSS quicker and easier.

12.1. Preprocessors

CSS preprocessors/precompilers have grown in popularity in recent years, for good reason. Working from a sort of CSS metalanguage, preprocessors offer a more convenient syntax; the ability to define variables, mixins, and nesting (all of which we'll look at below); and generally simplify the chore of writing lots and lots of CSS. All generate syntactically valid CSS. We'll look here at `Sass (http://sass-lang.com/)` (Syntactically Awesome Stylesheets); `Less (http://lesscss.org/)` is another popular alternative.

Using Sass means writing Sass code in a `.scss` file and then compiling that code (using the Ruby-based compiler) to generate syntactically valid, perfectly ordinary CSS code in a `.css` file. Mac users likely have Ruby already installed (it comes with macOS); Windows users can download the `RubyInstaller (http://rubyinstaller.org/downloads/)` to install Ruby. Once Ruby is installed, you can download and install Sass (for free) by typing `gem install sass` (`sudo gem install sass` and type admin password, if prompted, on a Mac) from a command prompt. Let's look at a simple example.

Open the three files in `ClassFiles/sass/Demos/sass/`. The HTML file is pretty simple - a div containing an `h2` title and some paragraph text - and is styled via an external style sheet, `sassstyle.css:`, which has the following code:

```
#example {
  border: 1px solid #d98c8c; }
#example h2 {
  color: #dbc7bd; }
```

This CSS file was generated by compiling the Sass style sheet `sassstyle.scss`:

```
$border-color: hsl(0, 50%, 70%);
$h2-color: hsl(20, 30%, 80%);
#example {
  border:1px solid $border-color;
  h2 {
    color:$h2-color;
  }
}
```


You can write Sass in a couple of different ways. The example here shows SCSS ("Sassy CSS"); it looks very much like CSS - in fact, any valid CSS is valid SCSS. The other, and older, syntax - often referred to as "indented syntax" or just "Sass" - uses indentation instead of braces to indicate nesting and uses newlines instead of semicolons to indicate the end of properties. Which you use is up to you; we'll use the newer syntax here.

At the start of `sassstyle.scss`, we define two variables, `$border-color` and `$h2-color`. While not a big deal in this example, you can see the value CSS variables brings to code organization, making it easier to change a color, width, or other value throughout thousands of lines of code by changing the value of one variable.

The Sass file also shows the simplicity of nesting: the styles for the `<h2>` are defined within the open/close braces that define the styles for `#example` - the generated output lists the styles correctly, with `#example h2`.

There's lots more Sass can do: function, interpolation (using `#{$i}` to display the actual value of Sass variable `$i`), mixins (allowing re-use of styles, optionally with arguments), and other features of the language offer a cleaner, quicker way to write and maintain complex CSS code. We can even write a for loop:

```
@for $i from 1 through $number-of-items {
  ul li:nth-child(#{$i}) {
    background-color: darken($key-color, $i * 5);
  }
}
```


This code would write multiple statements using the `:nth-child` selector to set the background color of a list item darker and darker, making use of Sass's `darken` function (which darkens a color, its first parameter, by the percentage given in its second parameter.)

The next exercise will give you a chance to try out Sass.

Exercise 23: Sass

15 to 25 minutes

In this exercise, you will use Sass to generate CSS to style a simple page.

1. If needed, install Ruby on your computer; `RubyInstaller (http://rubyinstaller.org/downloads/)` is your best bet if running Windows.
2. If you haven't yet, install the Sass Ruby gem: type `gem install sass` (`sudo gem install sass` and type admin password, if prompted, on a Mac) from the command prompt.
3. From a command prompt, navigate to the directory `ClassFiles/sass/Exercises/sass/`.
4. Type the following command from the command prompt and hit enter: `sass --watch sassstyle.scss:sassstyle.css`. This tells Sass to watch file `sassstyle.scss` for changes and, for each change, to compile the results of `sassstyle.scss` into `sassstyle.css`.
5. Open `ClassFiles/sass/Exercises/sass/index.html`. Note that the `index.html` file links to `sassstyle.css`; as you change `sassstyle.scss`, `sassstyle.css` will be updated and `index.html` will display the changes.
6. Open `ClassFiles/sass/Exercises/sass/sassstyle.scss` in a code editor and add code to perform the following:
    - Define a Sass variable `$container-width`, give it an appropriate value, and use it to define the width of the `#container` element.
    - Define a Sass variable `$number-of-items` and give it value `5` - this represents the number of list items.
    - Define a Sass variable `$key-color` and give it a color value (`#c00`, for example) of your choosing.
    - Add Sass code to define the style of the paragraph tag nested in `#container`: align the text in the center, add some top padding, and clear floated elements (to clear the list items, which will be floated above).
    - Give the `<ul>` a `list-style` of `none`.
    - Float each list item left and give each a `width` that is slightly less than the result of dividing the width of `#container` by the number of items.

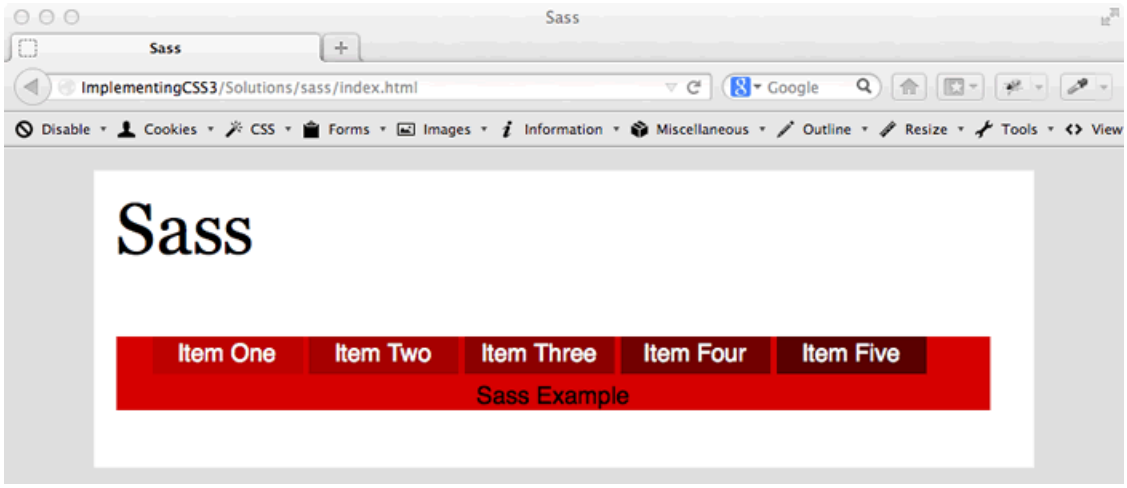
- Use a Sass `for` loop to make each successive list item slightly darker in background color.

Solution: sass/Solutions/sass/sassstyle.scss

```
1.  $container-width:700px;
2.  $number-of-items:5;
3.  $key-color:#c00;
4.
5.  #container {
6.   width: $container-width;
7.   background-color: $key-color;
8.   p {
9.    clear:both;
10.   text-align:center;
11.   padding-top:10px;
12.  }
13. }
14.
15. ul {
16.  list-style: none;
17. }
18.
19. ul li {
20.  float: left;
21.  margin-right: 5px;
22.  width: $container-width/$number-of-items - 20px;
23.  padding:5px 0;
24.  text-align:center;
25.  color:white;
26. }
27.
28. @for $i from 1 through $number-of-items {
29.  ul li:nth-child(#{$i}) {
30.   background-color: darken($key-color, $i * 5);
31.  }
32. }
```


Code Explanation

Our page should look something like this:



In `sassstyle.scss`, we define three variables to hold the width of the container, the number of elements, and our desired color, respectively.

We nest the styles for `#container p` inside those of `#container` in the Sass file - the result comes out in the `sassstyles.css` just like we want.

We give each list item a width of `$container-width/$number-of-items - 20px`, referring to the variables we created earlier.

Last, we use a Sass `for` loop to make each list item's background color darker and darker.

Conclusion

In this lesson, you have learned

- How CSS preprocessors like Sass make developing CSS easier and faster.
- How to install and use Sass.

LESSON 13
Frameworks

Topics Covered

☑ How CSS responsive frameworks make building sites faster and easier.

☑ How to use the Bootstrap framework.

☑ How to use the Foundation framework.

☑ How to use the UIKit framework.

Introduction

Responsive CSS frameworks like Bootstrap, Foundation, and UIkit make the process of building and maintaining websites for a variety of devices quicker and easier.

13.1. Frameworks

Responsive frameworks like Bootstrap, Foundation, UIkit, and others make easier the process of building pages optimized for different device widths - to present a different layout for browsers on desktops, tablet, phones, and other devices. Including the framework CSS and JavaScript resources in our pages means that we can write HTML markup to leverage the grid system of the framework.

For instance, we might present content for visitors using a desktop browser in two equal-width columns (each taking up 50% of the horizontal width of the page), but display that same content as two stacked (one on top of the other, each at 100% width) elements for those viewing our page on a phone.

Most frameworks work with a 12-column grid system: we can set one element to be 6/12 (half) width for wide viewports, 4/12 (one-third) width for medium viewports, and 12/12 (100%) for narrow viewports. We use CSS classes to control the display of elements' widths. Container and "row" elements (`div`s or other HTML tags) wrap the individual "cell" elements.

We can (and usually do) write our own CSS when using frameworks, to customize colors, typography, and other aspects of our pages' styles. But the CSS provided by the framework goes a long way toward achieving our page layout, even before we add our own customization.

Frameworks can be installed and maintained on our pages in various ways. Most provide CDNs (content delivery networks) from which we can link needed CSS and JavaScript files. Many work with popular package managers like npm, RubyGems, and Composer. Most frameworks allow us to control settings like font, page width, and column gutters through CSS preprocessors like Sass and Less.

In our examples below, we use the simplest way of installing the frameworks: we download the needed CSS and JavaScript files and include them in our sample pages.

13.2. Bootstrap

The most popular of the frameworks, `Bootstrap (https://getbootstrap.com/)` was originally developed at Twitter and released as an open source project in 2011. The grid system is based on a 12-column grid and defaults to the following responsive breakpoints:


```
// Extra small devices (portrait phones, less than 576px)
// No media query since this is the default in Bootstrap

// Small devices (landscape phones, 576px and up)
@media (min-width: 576px) { ... }

// Medium devices (tablets, 768px and up)
@media (min-width: 768px) { ... }

// Large devices (desktops, 992px and up)
@media (min-width: 992px) { ... }

// Extra large devices (large desktops, 1200px and up)
@media (min-width: 1200px) { ... }
```

When using Bootstrap, we use an HTML element with class `container` that wraps elements with class `row`, which in turn wrap cell elements. These elements have classes like `col`, `col-8`, `col-sm-6`, etc. to control the width of the elements. We'll review these in the example below.

Visit the `Bootstrap documentation (https://getbootstrap.com/docs/4.0/layout/overview/)` for more info on layout details.

❖ 13.2.1. Setting up Bootstrap

To download Boostrap, visit `https://getbootstrap.com/docs/4.0/getting-started/download/` and choose the “Compiled CSS and JS” option.

1. Extract the zipped download.
2. Save the `css` and `js` directories to a new directory.
3. Create a new HTML file (like `index.html`) in the root of the new directory.
4. Add a CSS link to `css/bootstrap.min.css` in the `head` of the HTML file.
5. Add JavaScript links to jQuery and to `js/bootstrap.min.js` just before the closing `body` tag.

The following example shows how to setup a first Bootstrap page and how to use Bootstrap.

Demo 13.1: frameworks/Demos/Bootstrap/index.html

```
1.   <!doctype html>
2.   <html lang="en">
3.   <head>
4.    <meta charset="utf-8">
5.    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-
            fit=no">
6.    <meta name="description" content="">
7.    <meta name="author" content="">
8.
9.    <title>Grid Template for Bootstrap</title>
10.
11.   <!-- Bootstrap core CSS -->
12.   <link href="css/bootstrap.min.css" rel="stylesheet">
13.   <style>
14.    body {
15.     padding-top: 2rem;
16.     padding-bottom: 2rem;
17.    }
18.
19.    h3 {
20.     margin-top: 2rem;
21.    }
22.
23.    .row {
24.     margin-bottom: 1rem;
25.    }
26.    .row .row {
27.     margin-top: 1rem;
28.     margin-bottom: 0;
29.    }
30.    [class*="col-"] {
31.     padding-top: 1rem;
32.     padding-bottom: 1rem;
33.     background-color: rgba(86, 61, 124, .15);
34.     border: 1px solid rgba(86, 61, 124, .2);
35.    }
36.
37.    hr {
38.     margin-top: 2rem;
39.     margin-bottom: 2rem;
40.    }
41.   </style>
42.  </head>
43.
```

```
<body>
 <div cldass="container">

  <h1>Bootstrap grid examples</h1>
  <p class="lead">Basic grid layouts to get you familiar with building within
        the Bootstrap grid system.</p>

  <h3>Five grid tiers</h3>
  <p>There are five tiers to the Bootstrap grid system, one for each range of
        devices we support. Each tier starts at a minimum viewport size and
        automatically applies to the larger devices unless overridden.</p>

  <div class="row">
   <div class="col-4">.col-4</div>
   <div class="col-4">.col-4</div>
   <div class="col-4">.col-4</div>
  </div>

  <div class="row">
   <div class="col-sm-4">.col-sm-4</div>
   <div class="col-sm-4">.col-sm-4</div>
   <div class="col-sm-4">.col-sm-4</div>
  </div>

  <div class="row">
   <div class="col-md-4">.col-md-4</div>
   <div class="col-md-4">.col-md-4</div>
   <div class="col-md-4">.col-md-4</div>
  </div>

  <div class="row">
   <div class="col-lg-4">.col-lg-4</div>
   <div class="col-lg-4">.col-lg-4</div>
   <div class="col-lg-4">.col-lg-4</div>
  </div>

  <div class="row">
   <div class="col-xl-4">.col-xl-4</div>
   <div class="col-xl-4">.col-xl-4</div>
   <div class="col-xl-4">.col-xl-4</div>
  </div>

  <h3>Three equal columns</h3>
  <p>Get three equal-width columns <strong>starting at desktops and scaling to
        large desktops</strong>. On mobile devices, tablets and below, the
        columns will automatically stack.</p>
```

```
85.     <div class="row">
86.      <div class="col-md-4">.col-md-4</div>
87.      <div class="col-md-4">.col-md-4</div>
88.      <div class="col-md-4">.col-md-4</div>
89.     </div>
90.
91.     <h3>Three unequal columns</h3>
92.     <p>Get three columns <strong>starting at desktops and scaling to large desk ↵↵
            tops</strong> of various widths. Remember, grid columns should add up
             to twelve for a single horizontal block. More than that, and columns
             start stacking no matter the viewport.</p>
93.     <div class="row">
94.      <div class="col-md-3">.col-md-3</div>
95.      <div class="col-md-6">.col-md-6</div>
96.      <div class="col-md-3">.col-md-3</div>
97.     </div>
98.
99.     <h3>Two columns</h3>
100.    <p>Get two columns <strong>starting at desktops and scaling to large desk ↵↵
            tops</strong>.</p>
101.    <div class="row">
102.     <div class="col-md-8">.col-md-8</div>
103.     <div class="col-md-4">.col-md-4</div>
104.    </div>
105.
106.    <h3>Full width, single column</h3>
107.    <p class="text-warning">No grid classes are necessary for full-width ele ↵↵
            ments.</p>
108.
109.    <hr>
110.
111.    <h3>Two columns with two nested columns</h3>
112.    <p>Per the documentation, nesting is easy—just put a row of columns within an
             existing column. This gives you two columns <strong>starting at desk ↵↵
            tops and scaling to large desktops</strong>, with another two (equal
            widths) within the larger column.</p>
113.    <p>At mobile device sizes, tablets and down, these columns and their nested
            columns will stack.</p>
114.    <div class="row">
115.     <div class="col-md-8">
116.      .col-md-8
117.      <div class="row">
118.       <div class="col-md-6">.col-md-6</div>
119.       <div class="col-md-6">.col-md-6</div>
120.      </div>
121.     </div>
122.     <div class="col-md-4">.col-md-4</div>
```

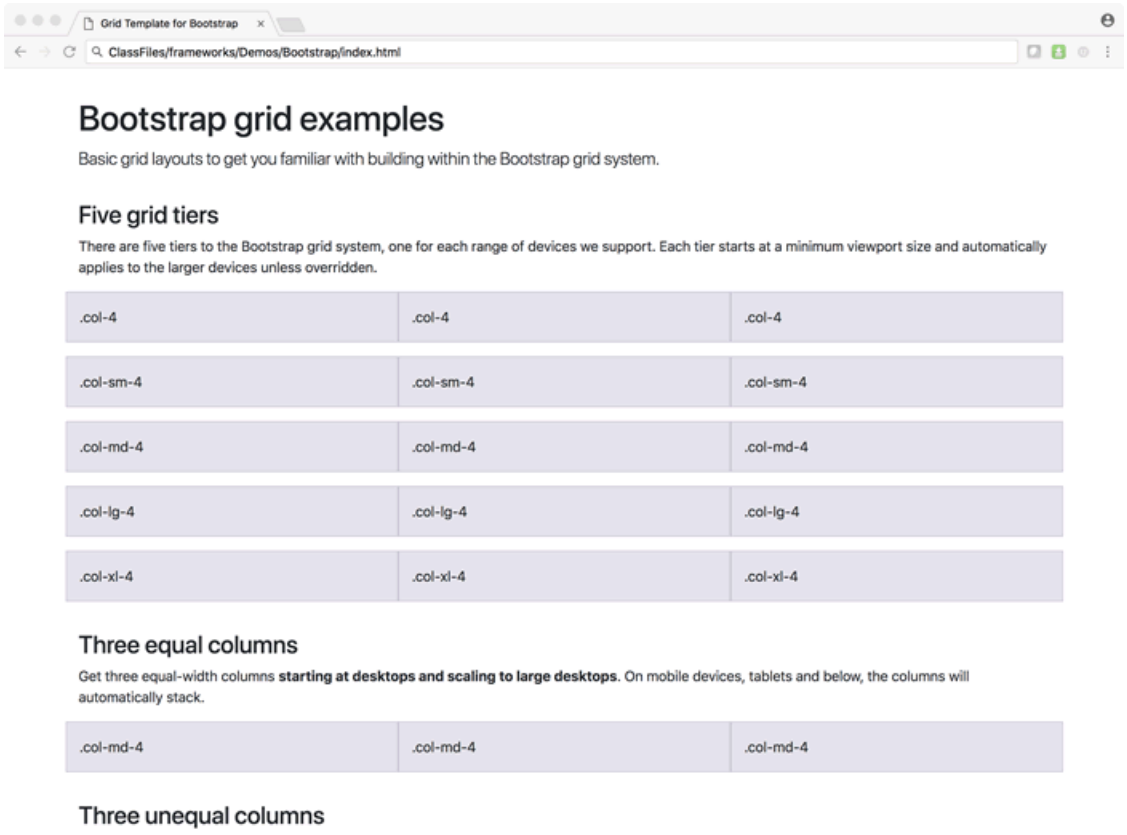
```
123.    </div>
124.
125.    <hr>
126.
127.    <h3>Mixed: mobile and desktop</h3>
128.    <p>The Bootstrap v4 grid system has five tiers of classes: xs (extra small),
            sm (small), md (medium), lg (large), and xl (extra large). You can use
             nearly any combination of these classes to create more dynamic and
            flexible layouts.</p>
129.    <p>Each tier of classes scales up, meaning if you plan on setting the same
            widths for xs and sm, you only need to specify xs.</p>
130.    <div class="row">
131.     <div class="col-12 col-md-8">.col-12 .col-md-8</div>
132.     <div class="col-6 col-md-4">.col-6 .col-md-4</div>
133.    </div>
134.    <div class="row">
135.     <div class="col-6 col-md-4">.col-6 .col-md-4</div>
136.     <div class="col-6 col-md-4">.col-6 .col-md-4</div>
137.     <div class="col-6 col-md-4">.col-6 .col-md-4</div>
138.    </div>
139.    <div class="row">
140.     <div class="col-6">.col-6</div>
141.     <div class="col-6">.col-6</div>
142.    </div>
143.
144.    <hr>
145.
146.    <h3>Mixed: mobile, tablet, and desktop</h3>
147.    <p></p>
148.    <div class="row">
149.     <div class="col-12 col-sm-6 col-lg-8">.col-12 .col-sm-6 .col-lg-8</div>
150.     <div class="col-6 col-lg-4">.col-6 .col-lg-4</div>
151.    </div>
152.    <div class="row">
153.     <div class="col-6 col-sm-4">.col-6 .col-sm-4</div>
154.     <div class="col-6 col-sm-4">.col-6 .col-sm-4</div>
155.     <div class="col-6 col-sm-4">.col-6 .col-sm-4</div>
156.    </div>
157.
158.    <h3>Components</h3>
159.    <div class="row">
160.     <div class="col-12">
161.      <h2>Buttons</h2>
162.      <button type="button" class="btn btn-primary">Primary</button>
163.      <button type="button" class="btn btn-secondary">Secondary</button>
164.      <button type="button" class="btn btn-success">Success</button>
```

```
165.      <button type="button" class="btn btn-danger">Danger</button>
166.      <button type="button" class="btn btn-warning">Warning</button>
167.      <button type="button" class="btn btn-info">Info</button>
168.      <button type="button" class="btn btn-light">Light</button>
169.      <button type="button" class="btn btn-dark">Dark</button>
170.     </div>
171.     <div class="col-12">
172.      <h2>Forms</h2>
173.      <form>
174.       <div class="form-group">
175.        <label for="exampleInputEmail1">Email address</label>
176.       <input type="email" class="form-control" id="exampleInputEmail1" aria-de ↵↵
              scribedby="emailHelp" placeholder="Enter email">
177.        <small id="emailHelp" class="form-text text-muted">We'll never share your
               email with anyone else.</small>
178.       </div>
179.       <div class="form-group">
180.        <label for="exampleInputPassword1">Password</label>
181.        <input type="password" class="form-control" id="exampleInputPassword1"
              placeholder="Password">
182.       </div>
183.       <div class="form-check">
184.        <input type="checkbox" class="form-check-input" id="exampleCheck1">
185.        <label class="form-check-label" for="exampleCheck1">Check me out</label>
186.       </div>
187.       <button type="submit" class="btn btn-primary">Submit</button>
188.      </form>
189.     </div>
190.    </div>
191.   </div> <!-- /container -->
192.   <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-
              KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN"
              crossorigin="anonymous"></script>
193.   <script src="js/bootstrap.min.js"></script>
194.  </body>
195.  </html>
```

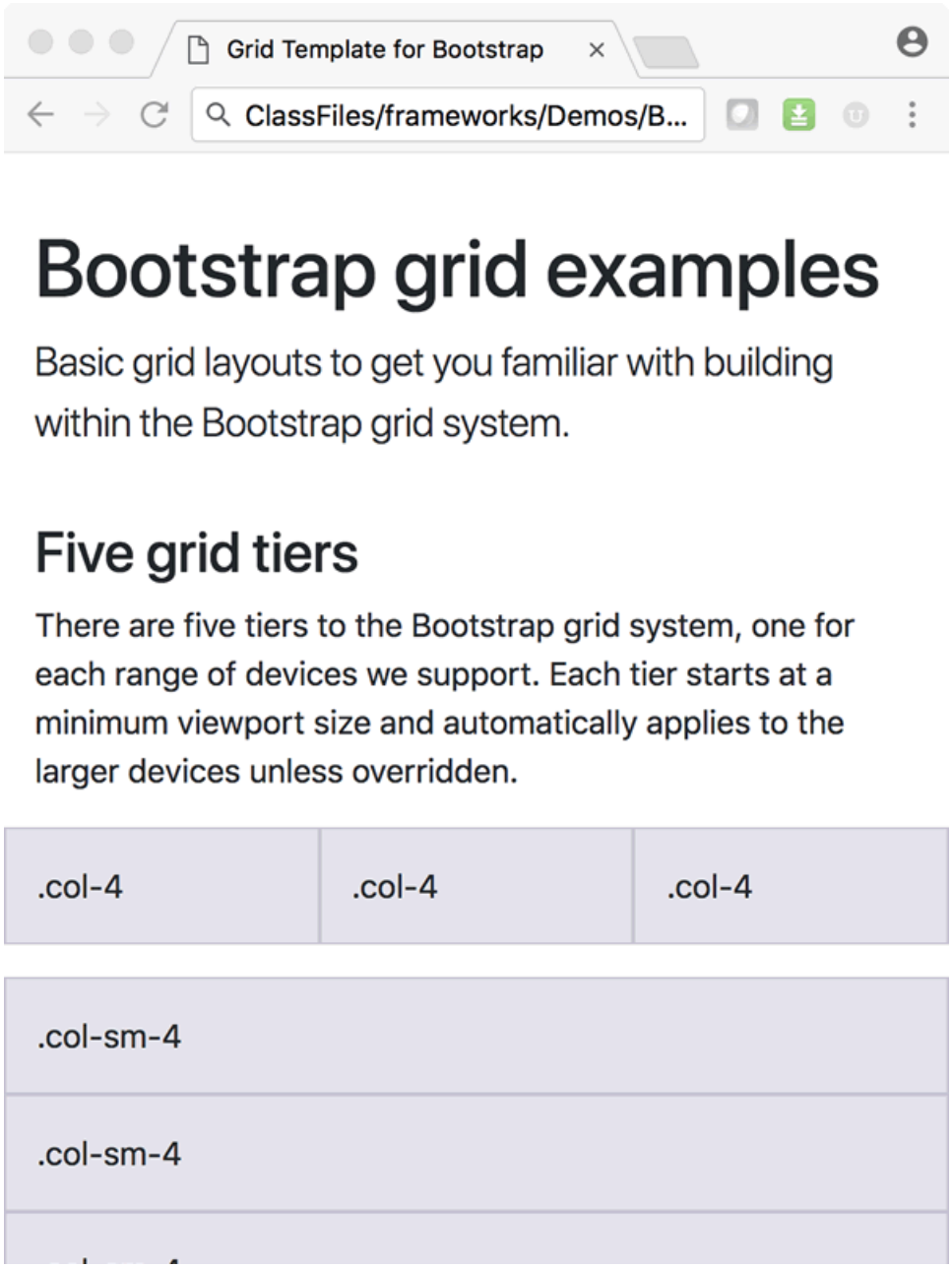

Code Explanation

Open `ClassFiles/frameworks/Demos/Bootstrap/index.html` in a file editor and in a browser to view the page.

. Here's the wide desktop view:



And here's the view for a narrower browser:



We include the needed CSS and JavaScript files (linking to jQuery) in our page. Note that we've added some custom CSS in the `head` of the document, to make the grid cells easier to see.

The outermost tag in our page is a `div` with class `container`; this has the result of centering the contained content - the width of page's content never exceeds `1140px`, regardless of how wide the browser is sized.

The top set of rows (with title “Five grid tiers”) demonstrates how we can use grid classes.

The first row’s three elements, each with class `col-4`, always get one-third width (4/12), regardless of the browser width.

The second row of three elements, each with class `col-sm-4`, display each as one-third width for browsers at the “small” breakpoint or wider (hence the "`sm`" identifier in the class name): the elements stack with full width at widths below `576px`.

The third row of three elements, each with class `col-md-4`, work the same way as the previous example - but, since the class name includes "`md`", the elements stack at browser widths below `768px`.

The next two rows of three elements demonstrate this same concept for classes `col-lg-4` (stacking at widths below `992px`) and `col-xl-4` (stacking at widths below `1200px`).

We can combine class names to control the widths of elements at different breakpoints: see the set of rows in the “Mixed: mobile and desktop” to see this in action.

Like all frameworks, Bootstrap offers a wide range of pre-styled components. At the bottom of our demo page, we show some buttons and some form elements. Visit `https://getbootstrap.com/docs/4.0/components/ (mailto:https://getbootstrap.com/docs/4.0/components/)` for more details.

❖ 13.2.2. Display Utility Classes

Bootstrap includes a set of classes we can apply to HTML elements that control on which view widths those elements display. We can use a class of `d-none` to hide content on all widths, `d-none d-sm-block` to hide only on extra small, `d-sm-none d-md-block` to hide only on small, `d-block` to display always, `d-block d-sm-none` to display only on extra small, etc. These utility classes mean we can control visibility of elememnts by applying classes to our markup and avoid having to write media queries in CSS. See the `Bootstrap documentation (https://getbootstrap.com/docs/4.0/utilities/display/)` for more information.

13.3. Foundation

The `Foundation (https://foundation.zurb.com/)` framework is “a family of responsive front-end frameworks that make it easy to design beautiful responsive websites, apps and emails that look amazing on any device”. The framwork offers versions for websites, email campaigns, and products/apps.

Foundation’s default grid system is the “XY Grid”. Built using Flexbox, the grid allows us to craft various layouts for larger and smaller device widths. We’ll delve into the details with the example below.

❖ 13.3.1. Setting Up Foundation

Foundation offers the ability to download the complete set of elements, a lightweight version, a customized version, or a Sass version. We cover downloading the “Complete” version below:

1. `Download Foundation (https://foundation.zurb.com/sites/download.html/).`
2. Extract the zipped archive to a new directory.
3. CSS and JavaScript Foundation resources live in the `css` and `js` directories, respectively.
4. Link to `css/foundation.css` (or the minified `css/foundation.min.css`) in the head of all HTML files; link to needed JavaScript files (`jquery.js`, `what-input.js`, and `foundation.js`) before the closing body tag.
5. Include your own CSS file (for custom styling) and JavaScript file (for custom JS) in your HTML files.
6. The downloaded Foundation resources include a sample `index.html` file demonstrating the needed style and script links, along with sample markup and a wide range of elements.

The Foundation documentation offers a set of `HTML templates (https://foundation.zurb.com/templates.html)` from which you can get started.

Let’s take a look at how the markup works and some of the elements Foundation offers.

Demo 13.2: frameworks/Demos/Foundation/index.html

```
1.  <!doctype html>
2.  <html class="no-js" lang="en" dir="ltr">
3.    <head>
4.      <meta charset="utf-8">
5.      <meta http-equiv="x-ua-compatible" content="ie=edge">
6.      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7.      <title>Foundation for Sites</title>
8.      <link rel="stylesheet" href="css/foundation.css">
9.      <link rel="stylesheet" href="css/app.css">
10.   </head>
11.   <body>
12.     <div class="grid-container">
13.       <div class="grid-x grid-padding-x">
14.         <div class="large-12 cell">
15.           <h1>Welcome to Foundation</h1>
16.         </div>
17.       </div>
18.
19.       <div class="grid-x grid-padding-x">
20.         <div class="large-12 cell">
21.           <div class="callout">
22.             <h3>We're stoked you want to try Foundation! </h3>
23.             <p>To get going, this file (index.html) includes some basic styles
            you can modify, play around with, or totally destroy to get going.</p>
24.             <p>Once you've exhausted the fun in this document, you should check
             out:</p>
25.             <div class="grid-x grid-padding-x">
26.               <div class="large-4 medium-4 cell">
27.                 <p><a href="http://foundation.zurb.com/docs">Foundation Documen ↵↵
            tation</a><br />Everything you need to know about using the frame ↵↵
            work.</p>
28.               </div>
29.               <div class="large-4 medium-4 cell">
30.                 <p><a href="http://zurb.com/university/code-skills">Foundation
            Code Skills</a><br />These online courses offer you a chance to better
             understand how Foundation works and how you can master it to create
            awesome projects.</p>
31.               </div>
32.               <div class="large-4 medium-4 cell">
33.                 <p><a href="http://foundation.zurb.com/forum">Foundation Fo ↵↵
            rum</a><br />Join the Foundation community to ask a question or show
            off your knowlege.</p>
34.               </div>
35.             </div>
36.             <div class="grid-x grid-padding-x">
```

```
37.                 <div class="large-4 medium-4 medium-push-2 cell">
38.                   <p><a href="http://github.com/zurb/foundation">Foundation on
               Github</a><br />Latest code, issue reports, feature requests and
               more.</p>
39.                 </div>
40.                 <div class="large-4 medium-4 medium-pull-2 cell">
41.                   <p><a href="https://twitter.com/ZURBfoundation">@zurbfounda ↵↵
               tion</a><br />Ping us on Twitter if you have questions. When you build
                something with this we'd love to see it (and send you a totally boss
                sticker).</p>
42.                 </div>
43.               </div>
44.             </div>
45.           </div>
46.         </div>
47.
48.         <div class="grid-x grid-padding-x">
49.           <div class="large-8 medium-8 cell">
50.             <h5>Here's your basic grid:</h5>
51.             <!-- Grid Example -->
52.
53.             <div class="grid-x grid-padding-x">
54.               <div class="large-12 cell">
55.                 <div class="primary callout">
56.                   <p><strong>This is a twelve cell section in a grid-x.</strong>
               Each of these includes a div.callout element so you can see where the
                cell are - it's not required at all for the grid.</p>
57.                 </div>
58.               </div>
59.             </div>
60.             <div class="grid-x grid-padding-x">
61.               <div class="large-6 medium-6 cell">
62.                 <div class="primary callout">
63.                   <p>Six cell</p>
64.                 </div>
65.               </div>
66.               <div class="large-6 medium-6 cell">
67.                 <div class="primary callout">
68.                   <p>Six cell</p>
69.                 </div>
70.               </div>
71.             </div>
72.             <div class="grid-x grid-padding-x">
73.               <div class="large-4 medium-4 small-4 cell">
74.                 <div class="primary callout">
75.                   <p>Four cell</p>
```

```
76.               </div>
77.             </div>
78.             <div class="large-4 medium-4 small-4 cell">
79.               <div class="primary callout">
80.                 <p>Four cell</p>
81.               </div>
82.             </div>
83.             <div class="large-4 medium-4 small-4 cell">
84.               <div class="primary callout">
85.                 <p>Four cell</p>
86.               </div>
87.             </div>
88.           </div>
89.
90.           <hr />
91.
92.           <h5>We bet you'll need a form somewhere:</h5>
93.           <form>
94.             <div class="grid-x grid-padding-x">
95.               <div class="large-12 cell">
96.                 <label>Input Label</label>
97.                 <input type="text" placeholder="large-12.cell" />
98.               </div>
99.             </div>
100.            <div class="grid-x grid-padding-x">
101.              <div class="large-4 medium-4 cell">
102.                <label>Input Label</label>
103.                <input type="text" placeholder="large-4.cell" />
104.              </div>
105.              <div class="large-4 medium-4 cell">
106.                <label>Input Label</label>
107.                <input type="text" placeholder="large-4.cell" />
108.              </div>
109.              <div class="large-4 medium-4 cell">
110.                <div class="grid-x">
111.                  <label>Input Label</label>
112.                  <div class="input-group">
113.                    <input type="text" placeholder="small-9.cell" class="input-
          group-field" />
114.                    <span class="input-group-label">.com</span>
115.                  </div>
116.                </div>
117.              </div>
118.            </div>
119.            <div class="grid-x grid-padding-x">
120.              <div class="large-12 cell">
```

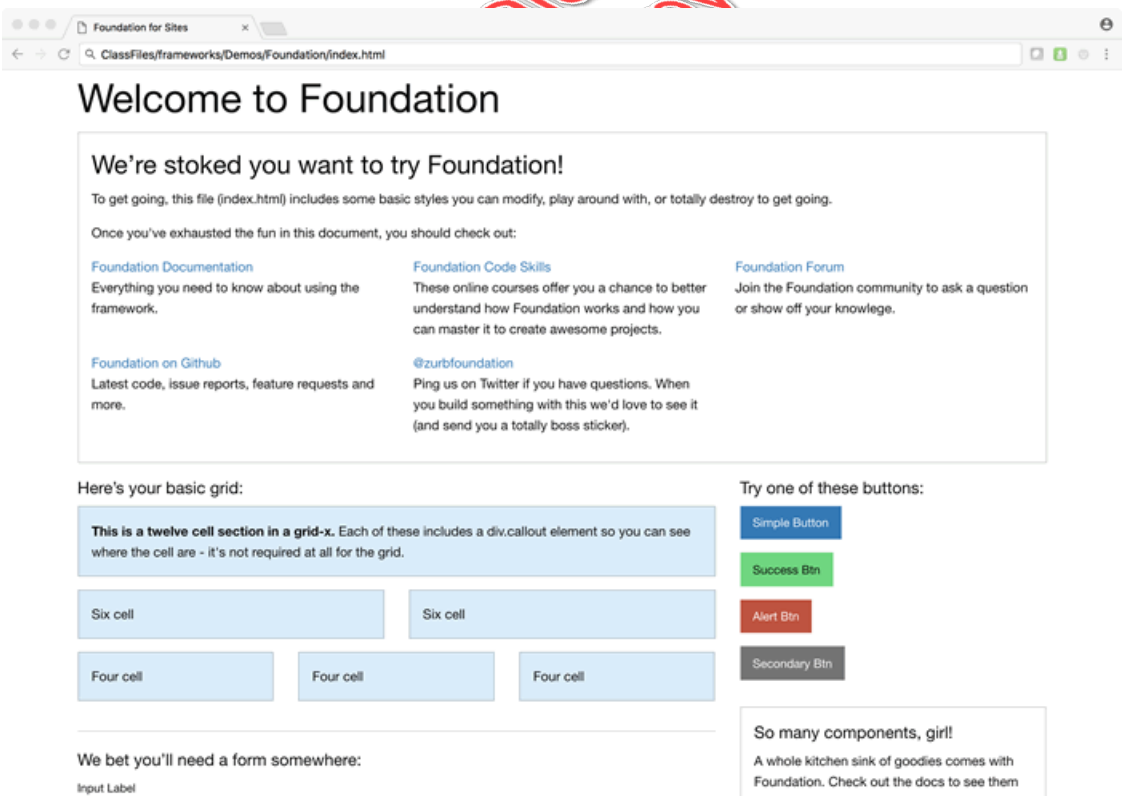
```
121.                 <label>Select Box</label>
122.                 <select>
123.                   <option value="husker">Husker</option>
124.                   <option value="starbuck">Starbuck</option>
125.                   <option value="hotdog">Hot Dog</option>
126.                   <option value="apollo">Apollo</option>
127.                 </select>
128.               </div>
129.             </div>
130.             <div class="grid-x grid-padding-x">
131.               <div class="large-6 medium-6 cell">
132.                 <label>Choose Your Favorite</label>
133.                <input type="radio" name="pokemon" value="Red" id="pokemonRed"><la ↵↵
             bel for="pokemonRed">Radio 1</label>
134.                 <input type="radio" name="pokemon" value="Blue" id="pokemon ↵↵
             Blue"><label for="pokemonBlue">Radio 2</label>
135.               </div>
136.               <div class="large-6 medium-6 cell">
137.                 <label>Check these out</label>
138.                <input id="checkbox1" type="checkbox"><label for="checkbox1">Check ↵↵
             box 1</label>
139.                <input id="checkbox2" type="checkbox"><label for="checkbox2">Check ↵↵
             box 2</label>
140.               </div>
141.             </div>
142.             <div class="grid-x grid-padding-x">
143.               <div class="large-12 cell">
144.                 <label>Textarea Label</label>
145.                 <textarea placeholder="small-12.cell"></textarea>
146.               </div>
147.             </div>
148.           </form>
149.         </div>
150.
151.         <div class="large-4 medium-4 cell">
152.           <h5>Try one of these buttons:</h5>
153.           <p><a href="#" class="button">Simple Button</a><br/>
154.           <a href="#" class="success button">Success Btn</a><br/>
155.           <a href="#" class="alert button">Alert Btn</a><br/>
156.           <a href="#" class="secondary button">Secondary Btn</a></p>
157.           <div class="callout">
158.             <h5>So many components, girl!</h5>
159.             <p>A whole kitchen sink of goodies comes with Foundation. Check out
              the docs to see them all, along with details on making them your
             own.</p>
```

```
160.            <a href="http://foundation.zurb.com/sites/docs/" class="small but ↵↵
               ton">Go to Foundation Docs</a>
161.            </div>
162.          </div>
163.        </div>
164.      </div>
165.
166.      <script src="js/vendor/jquery.js"></script>
167.      <script src="js/vendor/what-input.js"></script>
168.      <script src="js/vendor/foundation.js"></script>
169.      <script src="js/app.js"></script>
170.    </body>
171. </html>
```

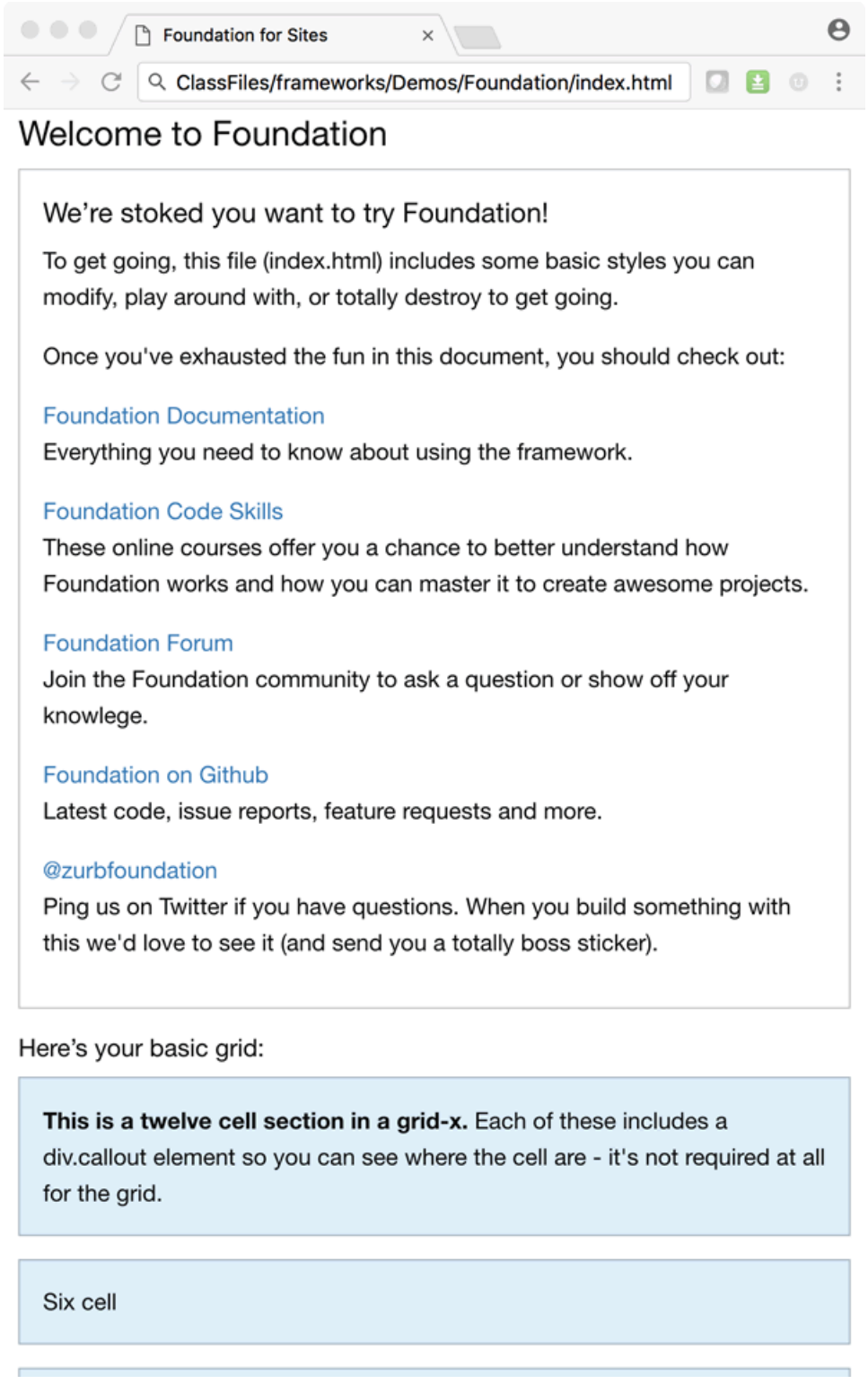
Code Explanation

Open `ClassFiles/frameworks/Demos/Foundation/index.html` in a file editor and in a browser to view the page.

The markup for this page comes from the default sample file that Foundation gives us when we download the framework files. Here's the wide desktop view:



And here's the view if we drag the browser narrower to approximate a phone's width:



The outermost element of the page is `<div class="grid-container">` - in Foundation, a `grid-container` class element contains the width of the contained elements (`1200px` max width, by default, but we can change this setting); in the absence of this parent container element, the grid elements default to the full width of the page.

Elements with class `grid-x` define grid elements - rows, effectively - and we can use `grid-margin-x` or `grid-padding-x` to add margin or padding to the grid cells.

The grid cells themselves are defined using class `cell` - with no other classes applied, a `cell` element spans the full width. We can use classes like `small-6`, `medium-6`, `large-4`, etc., to define how the cells take up width in the 12-column grid system. We can use multiple classes to define different layouts on different-width devices.

Like most frameworks, Foundation gives us predefined styles for a number of elements; the demo file we see here offers examples of how forms, buttons, and other elements show. The `Foundation docs (https://foundation.zurb.com/sites/docs/)` list all of the available elements and details on how to use them.

13.4. UIkit

The `UIkit (https://getuikit.com/)` framework grew out of Apple's Cocoa Touch UI framework. UIkit is a "lightweight and modular front-end framework for developing fast and powerful web interfaces"

❖ 13.4.1. Setting Up UIkit

1. `Download UIKit (https://getuikit.com/download).`
2. Extract the downloaded zip file.
3. Save the `css` and `js` directories to a new project (a new directory) and add an HTML file like `index.html`.
4. Link the needed JavaScript and CSS files in the HTML file.

Here's a simple example that shows how to get started with UIkit.

Demo 13.3: frameworks/Demos/UIkit/index.html

```
1.  <!DOCTYPE html>
2.  <html>
3.      <head>
4.          <title>Title</title>
5.          <meta charset="utf-8">
6.          <meta name="viewport" content="width=device-width, initial-scale=1">
7.          <link rel="stylesheet" href="css/uikit.min.css" />
8.          <script src="js/uikit.min.js"></script>
9.          <script src="js/uikit-icons.min.js"></script>
10.         <style>
11.             h1, h2 {
12.                 text-align: center;
13.             }
14.             .uk-card {
15.                 background:#ececec;
16.             }
17.         </style>
18.     </head>
19.     <body>
20.         <h1>UIkit</h1>
21.         <h2>Basic Grid</h2>
22.         <div class="uk-container">
23.             <div class="uk-grid-medium uk-child-width-expand@s uk-text-center"
            uk-grid>
24.                 <div>
25.                     <div class="uk-card uk-card-default uk-card-body">
26.                       <p>Lorem ipsum dolor sit amet, consectetuer adipiscin.</p>
27.                     </div>
28.                 </div>
29.                 <div>
30.                     <div class="uk-card uk-card-default uk-card-body">
31.                         <p>Lorem ipsum morbi metus nunc eros neque erat dolor
            tempor eget turpis.</p>
32.                     </div>
33.                 </div>
34.                 <div>
35.                     <div class="uk-card uk-card-default uk-card-body">
36.                         <p>Lorem ipsum molestie facilisis aliquam felis mi nam
            ultricies nec libero vestibulum.</p>
37.                     </div>
38.                 </div>
39.             </div>
40.         </div>
41.         <h2>Equal Heights</h2>
42.         <div class="uk-container">
```

```
43.             <div class="uk-grid-match uk-grid-medium uk-child-width-expand@s uk-
                text-center" uk-grid>
44.                 <div>
45.                     <div class="uk-card uk-card-default uk-card-body">
46.                         <p>Lorem ipsum dolor sit amet, consectetuer adipiscin.</p>
47.                     </div>
48.                 </div>
49.                 <div>
50.                     <div class="uk-card uk-card-default uk-card-body">
51.                         <p>Lorem ipsum morbi metus nunc eros neque erat dolor
                    tempor eget turpis.</p>
52.                     </div>
53.                 </div>
54.                 <div>
55.                     <div class="uk-card uk-card-default uk-card-body">
56.                         <p>Lorem ipsum molestie facilisis aliquam felis mi nam
                    ultricies nec libero vestibulum.</p>
57.                     </div>
58.                 </div>
59.             </div>
60.         </div>
61.         <h2>Dividers</h2>
62.         <div class="uk-container">
63.             <div class="uk-grid-divider uk-grid-medium uk-child-width-expand@s
                uk-text-center" uk-grid>
64.                 <div>
65.                     <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
                     Nam vestibulum ultrices est, ut aliquam eros dapibus quis. Ut finibus
                     sem a porta porttitor. Etiam maximus ligula lacinia ipsum eleifend,
                    in tincidunt arcu hendrerit. Cras ante metus, finibus vel leo et, iac ↵↵
                    ulis fermentum lectus. Nam et augue fringilla neque ultricies dictum.
                     In hac habitasse platea dictumst.</p>
66.                 </div>
67.                 <div>
68.                     <p>Suspendisse quis vestibulum tellus, in luctus lacus. Ut
                    nisl tellus, auctor nec odio convallis, auctor consectetur risus. Aenean
                     condimentum dui a imperdiet condimentum.</p>
69.                 </div>
70.                 <div>
71.                     <p>In ut pretium nisi. Cras ut ultrices tortor, at tincidunt
                     risus. Etiam accumsan lectus eu blandit hendrerit. Vestibulum
                    scelerisque tellus a ligula malesuada eleifend. Donec interdum nulla
                    sed vestibulum porttitor.</p>
72.                 </div>
73.             </div>
74.         </div>
75.         <h2>Grid Widths</h2>
76.         <div class="uk-container">
```

```
77.             <div class="uk-grid-medium uk-child-width-expand@s uk-text-center"
            uk-grid>
78.                 <div class="uk-width-1-4">
79.                     <div class="uk-card uk-card-default uk-card-body">
80.                       <p>Lorem ipsum dolor sit amet, consectetuer adipiscin.</p>
81.                     </div>
82.                 </div>
83.                 <div class="uk-width-1-4">
84.                     <div class="uk-card uk-card-default uk-card-body">
85.                         <p>Lorem ipsum morbi metus nunc eros neque erat dolor
            tempor eget turpis.</p>
86.                     </div>
87.                 </div>
88.                 <div class="uk-width-1-2">
89.                     <div class="uk-card uk-card-default uk-card-body">
90.                         <p>Lorem ipsum molestie facilisis aliquam felis mi nam
            ultricies nec libero vestibulum.</p>
91.                     </div>
92.                 </div>
93.             </div>
94.         </div>
95.         <h2>Grid Child Widths</h2>
96.         <div class="uk-container">
97.             <div class="uk-child-width-1-4 uk-grid-medium uk-text-center" uk-
            grid>
98.                 <div>
99.                     <div>
100.                        <ul uk-accordion>
101.                            <li>
102.                              <a class="uk-accordion-title" href="#">Toggle</a>
103.                                <div class="uk-accordion-content">
104.                                    <p>Lorem ipsum morbi metus nunc eros neque
            erat dolor tempor eget turpis.</p>
105.                                </div>
106.                            </li>
107.                        </ul>
108.                    </div>
109.                </div>
110.                <div>
111.                    <div>
112.                        <p uk-margin>
113.                            <button class="uk-button uk-button-default">De ↵↵
            fault</button>
114.                         <br><button class="uk-button uk-button-primary">Pri ↵↵
            mary</button>
115.                        <br><button class="uk-button uk-button-secondary">Sec ↵↵
            ondary</button>
```

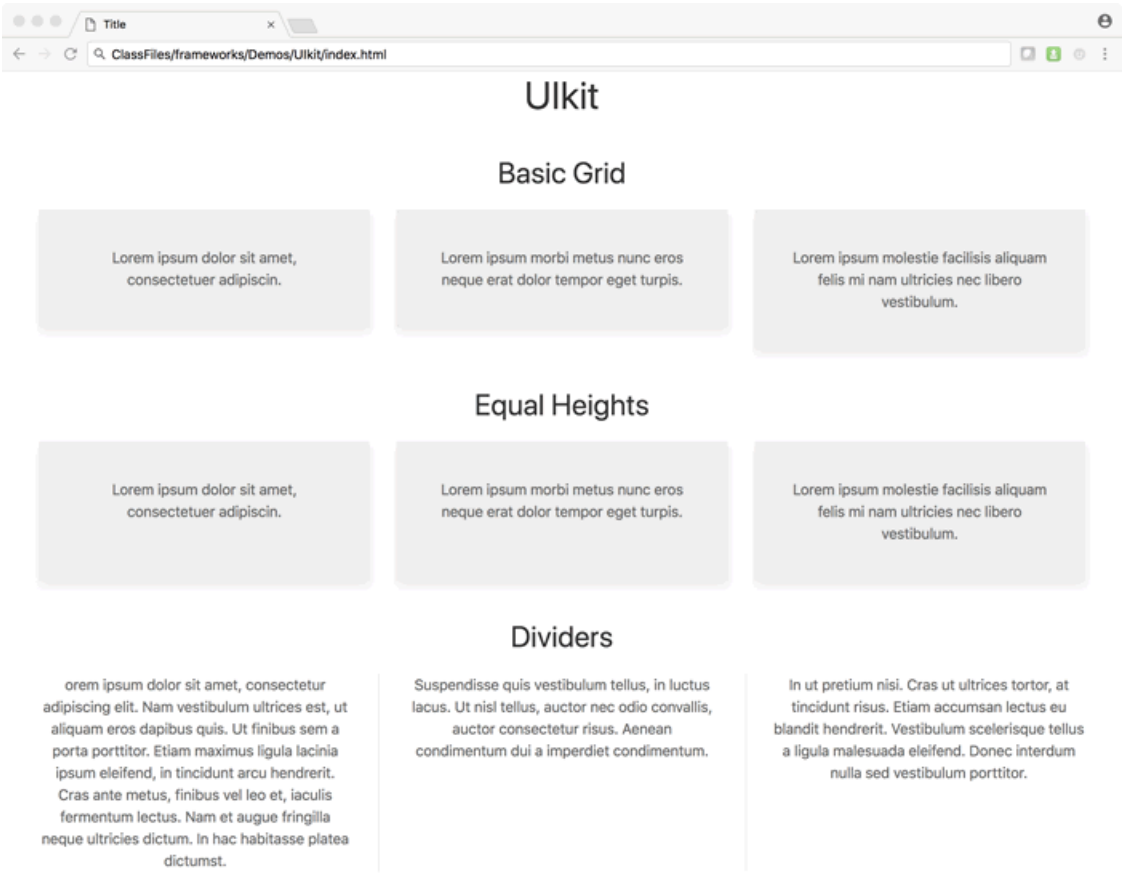
```
116.                     <br><button class="uk-button uk-button-danger">Dan ↵↵
        ger</button>
117.                     <br><button class="uk-button uk-button-
        text">Text</button>
118.                     <br><button class="uk-button uk-button-
        link">Link</button>
119.                 </p>
120.             </div>
121.         </div>
122.         <div>
123.             <div class="uk-card uk-card-default uk-card-body">
124.                 <p>Lorem ipsum molestie facilisis aliquam felis mi nam
        ultricies nec libero vestibulum.</p>
125.             </div>
126.         </div>
127.         <div>
128.             <div class="uk-card uk-card-default uk-card-body">
129.               <p>Lorem ipsum dolor sit amet, consectetuer adipiscin.</p>
130.             </div>
131.         </div>
132.         <div>
133.             <div class="uk-card uk-card-default uk-card-body">
134.                 <p>Lorem ipsum morbi metus nunc eros neque erat dolor
        tempor eget turpis.</p>
135.             </div>
136.         </div>
137.         <div>
138.             <div class="uk-card uk-card-default uk-card-body">
139.                 <p>Lorem ipsum molestie facilisis aliquam felis mi nam
        ultricies nec libero vestibulum.</p>
140.             </div>
141.         </div>
142.         <div>
143.             <div class="uk-card uk-card-default uk-card-body">
144.               <p>Lorem ipsum dolor sit amet, consectetuer adipiscin.</p>
145.             </div>
146.         </div>
147.         <div>
148.             <div class="uk-card uk-card-default uk-card-body">
149.                 <p>Lorem ipsum morbi metus nunc eros neque erat dolor
        tempor eget turpis.</p>
150.             </div>
151.         </div>
152.         <div>
153.             <div class="uk-card uk-card-default uk-card-body">
154.                 <p>Lorem ipsum molestie facilisis aliquam felis mi nam
        ultricies nec libero vestibulum.</p>
```

```
                    </div>
                </div>
            </div>
        </div>
    </body>
</html>
```

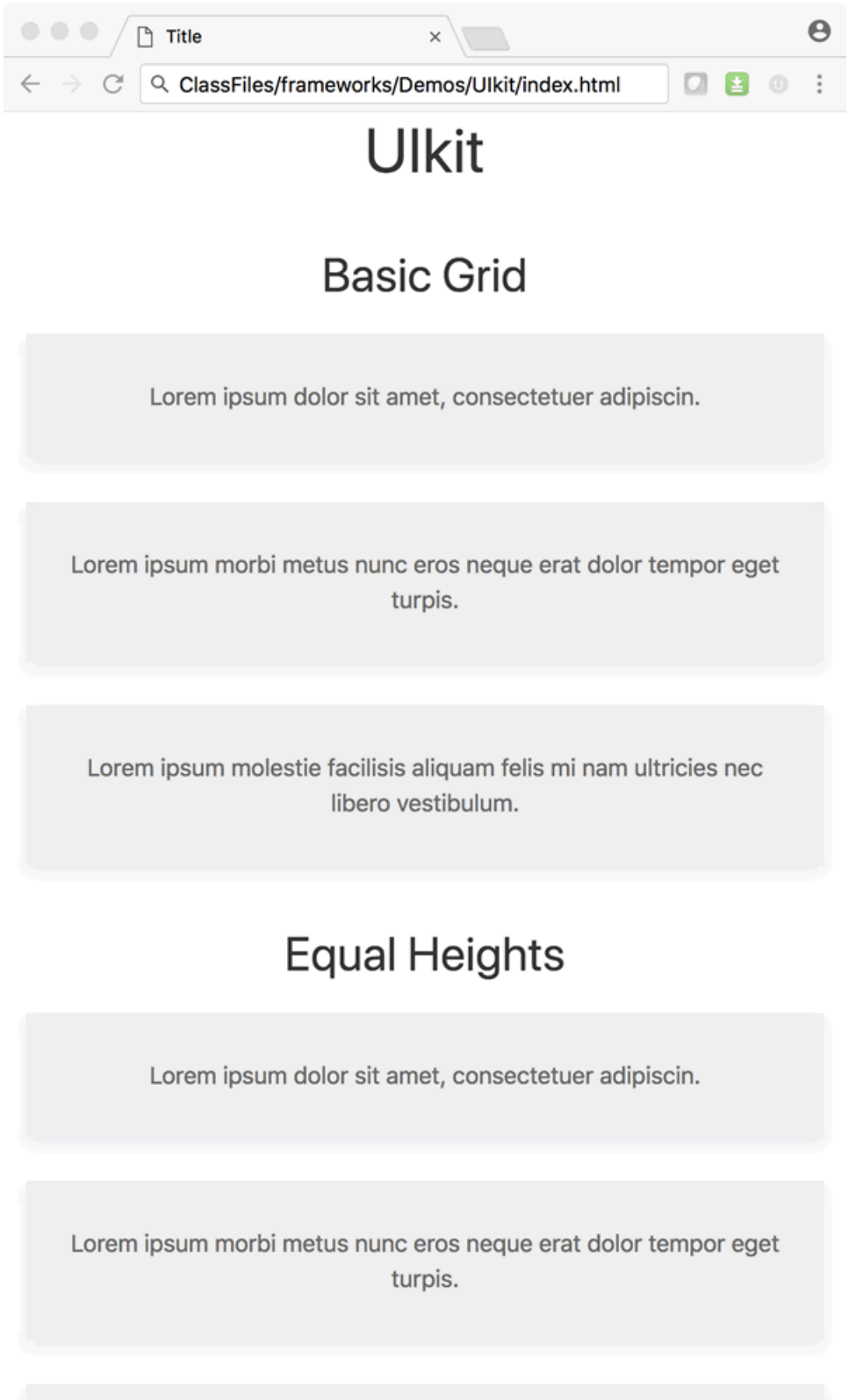
Code Explanation

Open `ClassFiles/frameworks/Demos/UIkit/index.html` in a file editor and in a browser to view the page.

We've created the markup for this page to demonstrate how to use the UIkit grid system and to present a few of the elements. Here's the wide desktop view:



And here's the narrower view:


Title
ClassFiles/frameworks/Demos/Ulkit/index.html
Ulkit
Basic Grid
Lorem ipsum dolor sit amet, consectetuer adipiscin.
Lorem ipsum morbi metus nunc eros neque erat dolor tempor eget turpis.
Lorem ipsum molestie facilisis aliquam felis mi nam ultricies nec libero vestibulum.
Equal Heights
Lorem ipsum dolor sit amet, consectetuer adipiscin.
Lorem ipsum morbi metus nunc eros neque erat dolor tempor eget turpis.

We present a series of examples of the UIkit grid; note that all of the examples contain an outer container element (`<div class="uk-container">`) and an inner element, a `div`, with attribute `uk-grid`; UIkit's JavaScript adds the needed attributes to this `div`. Elements inside the `uk-grid` comprise the grid cells. For most of the examples we use UIkit's `card` classes to more-easily display the cells visually.

The first ("Basic Grid") demonstrates that we need not give cell elements explicit widths; the presence of the `uk-child-width-expand@s` class automatically applies equal widths to the cell items, regardless of how many there are.

The "Equal Heights" example, with class `uk-grid-match`, shows how we can use UIkit to create equal-height cells.

For the "Dividers" example we use class `uk-grid-divider` to separate grid cells with vertical lines.

The "Grid Widths" and "Grid Child Widths" examples show how we can manually set cell widths. We use classes `uk-width-1-4` and `uk-width-1-2` for the first example to set the first two cells to have one-quarter width (`uk-width-1-4`) and the last cell to have one-half width. For the second example, we use class `uk-child-width-1-4` on the parent element to set the widths of each grid cell element to be one-quarter, regardless of how many elements exist.

Also in the "Grid Child Widths" example, we show a couple of the components offered by UIkit: an accordion (which toggles hidden content) and some buttons.

See the `UIKit documentation (https://getuikit.com/docs/introduction)` for more details and a list of available components.

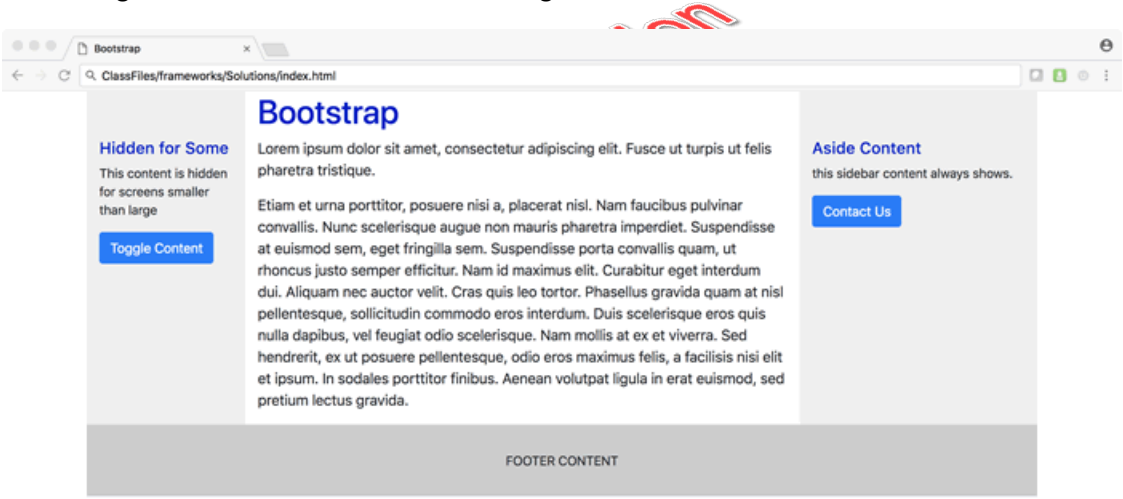
In the next exercise we'll ask you to try out the Bootstrap framework by rending a simple design.

Exercise 24: Bootstrap

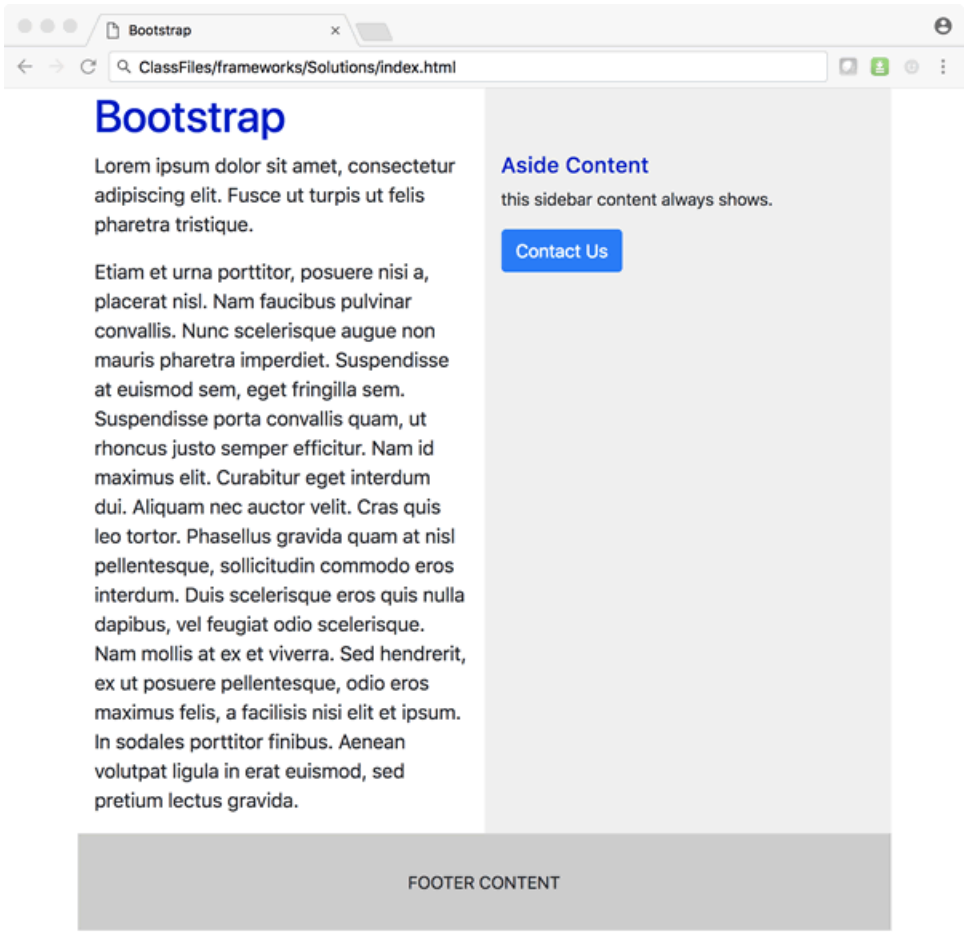
15 to 25 minutes

In this exercise, you will render a design using Bootstrap's grid system and a few Bootstrap components.

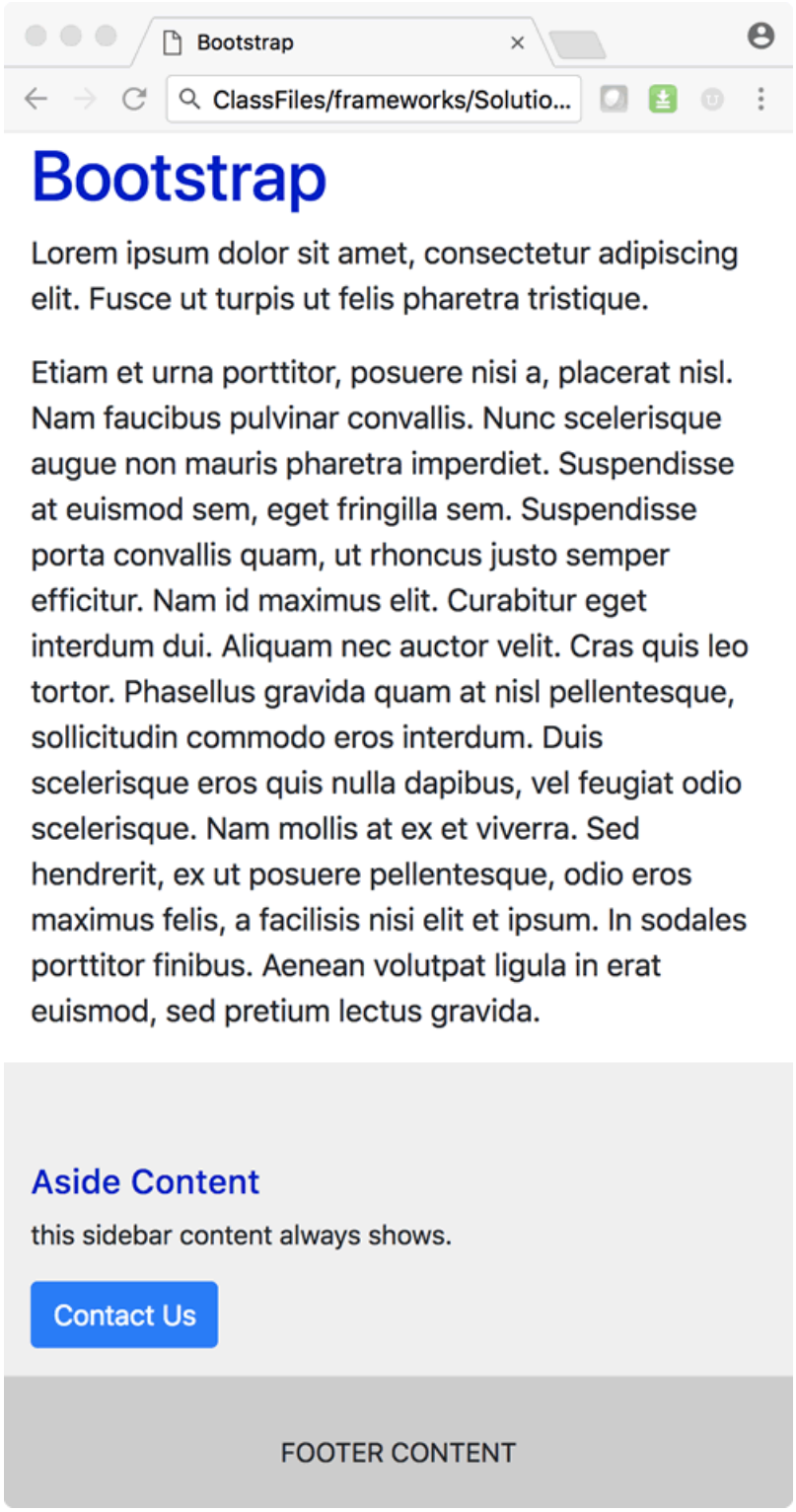
1. Open `ClassFiles/frameworks/Exercises/index.html` in a file editor and in a browser to view the page.
2. The goal here is to render the design to show like this on wide screens:



   like this on medium screens:



and like this on small screens:


Bootstrap
ClassFiles/frameworks/Solutio...
Bootstrap
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce ut turpis ut felis pharetra tristique.
Etiam et urna porttitor, posuere nisi a, placerat nisl. Nam faucibus pulvinar convallis. Nunc scelerisque augue non mauris pharetra imperdiet. Suspendisse at euismod sem, eget fringilla sem. Suspendisse porta convallis quam, ut rhoncus justo semper efficitur. Nam id maximus elit. Curabitur eget interdum dui. Aliquam nec auctor velit. Cras quis leo tortor. Phasellus gravida quam at nisl pellentesque, sollicitudin commodo eros interdum. Duis scelerisque eros quis nulla dapibus, vel feugiat odio scelerisque. Nam mollis at ex et viverra. Sed hendrerit, ex ut posuere pellentesque, odio eros maximus felis, a facilisis nisi elit et ipsum. In sodales porttitor finibus. Aenean volutpat ligula in erat euismod, sed pretium lectus gravida.
Aside Content
this sidebar content always shows.
Contact Us
FOOTER CONTENT

3. The left sidebar should be hidden for widths smaller than large. The sidebar/main content/sidebar elements should display in a ratio of 2/7/3 on wide screens, 6/6 on medium screens (with the left sidebar hidden), and stacked for small screens.
4. Use a custom .css file to render your own styles - try adding some text color, background, padding, etc.
5. Try adding a few Bootstrap elements - we added a `collapse element (https://getbootstrap.com/docs/4.0/components/collapse/)` in the left sidebar and a `button (https://getbootstrap.com/docs/4.0/components/buttons/)` in the right sidebar, but feel free to add whatever components you like.

Solution: frameworks/Solutions/index.html

```
1.   <!doctype html>
2.   <html lang="en">
3.   <head>
4.    <meta charset="utf-8">
5.    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-
            fit=no">
6.    <meta name="description" content="">
7.    <meta name="author" content="">
8.
9.    <title>Bootstrap</title>
10.
11.   <!-- Bootstrap core CSS -->
12.   <link href="css/bootstrap.min.css" rel="stylesheet">
13.   <link href="css/custom.css" rel="stylesheet">
14.  </head>
15.
16.  <body>
17.   <div class="container">
18.    <div class="row">
19.     <aside class="col-lg-2 d-none d-lg-block">
20.      <h2>Hidden for Some</h2>
21.      <p>This content is hidden for screens smaller than large</p>
22.      <p><a class="btn btn-primary" data-toggle="collapse" href="#collapseContent"
              role="button" aria-expanded="false" aria-controls="collapseContent">Tog ↵↵
            gle Content</a></p>
23.      <div class="collapse" id="collapseContent">
24.       <div class="card card-body">
25.        Vivamus vestibulum diam non bibendum egestas. Mauris vel ipsum mi.
26.       </div>
27.      </div>
28.     </aside>
29.     <article class="col-lg-7 col-md-6">
30.      <h1>Bootstrap</h1>
31.      <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce ut turpis
              ut felis pharetra tristique.</p>
32.      <p>Etiam et urna porttitor, posuere nisi a, placerat nisl. Nam faucibus
            pulvinar convallis. Nunc scelerisque augue non mauris pharetra imperdi ↵↵
            et. Suspendisse at euismod sem, eget fringilla sem. Suspendisse porta
             convallis quam, ut rhoncus justo semper efficitur. Nam id maximus
            elit. Curabitur eget interdum dui. Aliquam nec auctor velit. Cras quis
             leo tortor. Phasellus gravida quam at nisl pellentesque, sollicitudin
             commodo eros interdum. Duis scelerisque eros quis nulla dapibus, vel
             feugiat odio scelerisque. Nam mollis at ex et viverra. Sed hendrerit,
             ex ut posuere pellentesque, odio eros maximus felis, a facilisis nisi
             elit et ipsum. In sodales porttitor finibus. Aenean volutpat ligula
            in erat euismod, sed pretium lectus gravida.</p>
```

```
33.      </article>
34.      <aside class="col-lg-3 col-md-6">
35.       <h2>Aside Content</h2>
36.       <p>this sidebar content always shows.</p>
37.       <button type="button" class="btn btn-primary">Contact Us</button>
38.      </aside>
39.     </div>
40.     <footer class="row">
41.      <div class="col">
42.       <p class="text-center">Footer content</p>
43.      </div>
44.     </footer>
45.    </div>
46.    <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-
              KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN"
              crossorigin="anonymous"></script>
47.   <script src="js/bootstrap.min.js"></script>
48.   </body>
49.   </html>
```

Code Explanation

We wrap the content with a `div` of class `container`.

We use a `div` with class `row` to wrap the main (three columns) content.

The first `aside` element gets a class of `col-lg-2 d-none d-lg-block`, which hides the element for widths smaller than large and displays it with width 2/12 when visible.

The `article` element has class `col-lg-7 col-md-6`, giving it a width of 7/12 on large screens and 6/12 for medium screens.

The second `aside` displays with width 3/12 on wide screens and 6/12 on medium screens; we accomplish this with a class of `col-lg-3 col-md-6`.

Bootstrap gives the visibile elements a width of 12/12 (100%) when viewed on screens smaller than medium.

Lastly, we use a “collapse” element in the left sidebar - allowing users to toggle content by clicking the button - and a button element in the right sidebar.

Conclusion

In this lesson, you have learned

- How CSS frameworks help to build responsive websites.
- How to use the Bootstrap, Foundation, and UIKit frameworks.

LESSON 14
Grid Layout

LESSON 15
CSS Level 4 Selectors

LESSON 16

Going Forward/Additional Resources

Topics Covered

☑ How to keep current with future CSS developments.

☑ Where to find good resources on CSS.

☑ How to get involved with W3C CSS development and testing.

Introduction

Specifications for CSS and browser support for it change quickly - we discuss some ideas for keeping up to date and present some resources for improving your skills.

16.1. Going Forward/Additional Resources

❖ 16.1.1. What's Next?

CSS is changing rapidly. Work progresses independently, at different rates, on the various modules. As modules and their specific properties make their way through the W3C's candidate process, acceptance for new CSS capabilities will grow over time. The state of support from each browser (with or without vendor-specific prefixes) changes with each new browser version released. And, of course, the market share from each browser/version affects greatly how much we can use newer CSS techniques in our production sites.

So what's a poor designer/developer to do? A few thoughts, below, on resources to help keep up.

❖ 16.1.2. Online Resources

W3C

The World Wide Web Consortium's CSS Current Work page lists updates about and the current status of the various CSS modules. You can track progress of CSS specifications

as they wend their way from “Working Draft” to “Candidate Recommendation” to full “Recommendation”. A useful - and the most authoritative - place to find what’s new and what’s up with recent CSS developments is the `W3C’s CSS homepage (http://www.w3.org/Style/CSS/Overview.en.html)`.

fantasai’s (sic) article `Understanding the CSS Specifications (http://www.w3.org/Style/CSS/read)` on the W3C website is a wonderfully helpful guide to understanding the process by which the community pushes forward on CSS: how to read the specs, understanding errata, and where to find more info. This isn’t a guide to CSS; rather, it is a guide to understanding the W3C CSS specifications.

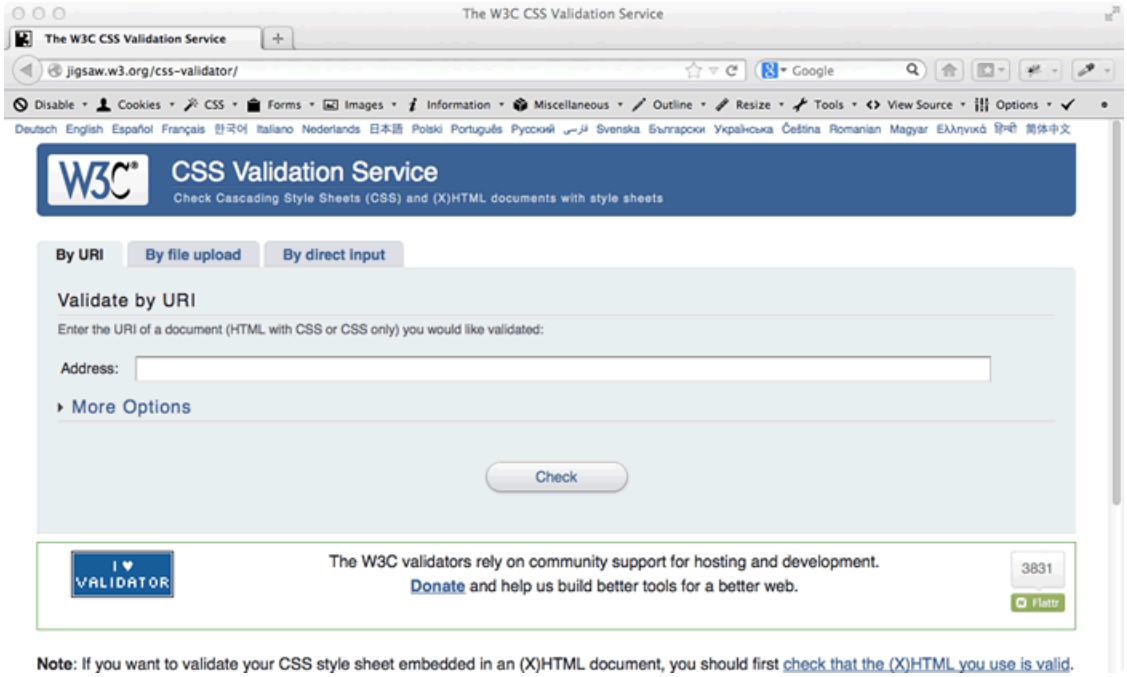
The `W3C’s Cascading Style Sheets: articles and tutorials (http://www.w3.org/Style/CSS/learning.en.html)` site lists upcoming conferences and events, articles and tutorials (in English and other languages), and other useful information.

W3C Mailing List

The W3C maintains an email list for technical discussions on CSS and its specifications; see the `archives page (http://lists.w3.org/Archives/Public/www-style/)` for details, including how to sign up. Note that this isn’t a list for questions like “how to I get CSS drop shadowing to work in Chrome?” but rather for discussion of the ongoing specifications.

W3C CSS Validator

The W3C offers a `validation service (http://jigsaw.w3.org/css-validator/)` where you can test your CSS code for conformity against the CSS specifications. You can test by entering the URL of your style sheet, by pasting in CSS code, uploading a file:



caniuse.com

We mentioned `caniuse.com (http://caniuse.com/)` previously in this course - it's a great place to keep track of browser version support for CSS (as well as HTML5 and other technologies) and the need for vendor prefixes. Browser support for CSS changes with each new version; caniuse.com can help you keep current.

Other Sites & Authors

`CSS3.info (http://www.css3.info/)` has news on recent CSS Level 3 developments, updates on module statuses, and a useful selector test, showing whether your browser supports any of 41 different selectors and pseudo-classes. `Eric Meyer (http://meyerweb.com/eric/css/)` is a well-respected online writer, speaker, and author about CSS topics - his thoughts are always well written and useful.

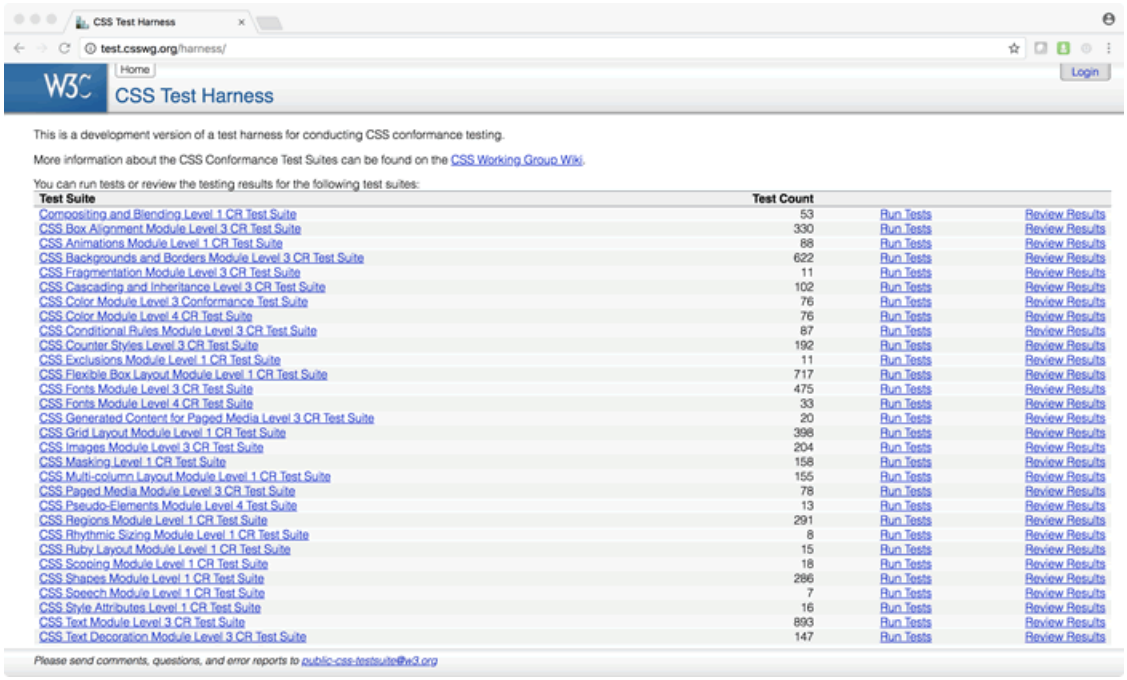
❖ 16.1.3. Get Involved

We learn best by doing - and how better to learn CSS than by being part of the team that develops it? You can contribute to the ongoing development of the various CSS modules by volunteering to help write test suites. The W3C's `Cascading Style Sheets Official`

`W3C Test Suites (http://www.w3.org/Style/CSS/Test/)` page has a nice summary of why test suites matter; it reads, in part:

> Interoperability is important to web designers. Better interoperability among CSS implementations means designers can write their CSS for one browser and see that it works predictably well on the other browsers. It means reducing the incompatibilities in the way CSS implementations interpret CSS.

You might, for instance, run through a series of tests like those found on the `W3C's Test Harness (http://test.csswg.org/harness/)` page:



Let's try out the media query test suite.

Exercise 25: Testing CSS

5 to 10 minutes

In this exercise, you will test the W3C's test suite for media queries.

1. Open `http://test.csswg.org/harness/` in your browser - choose the test suite "Media Queries Level 3 Conformance Test Suite" and click the "Run Tests" (`http://test.csswg.org/harness/suite/mediaqueries-3_dev/`) link.
2. Run the full test suite.
3. Compare the "Test Case" with the "Reference Page"; choose the appropriate button ("Pass", "Fail", etc.) at bottom.

Conclusion

In this lesson, you have learned:

- How to keep current with future CSS developments via useful online resources.
- How to get involved with W3C CSS test-suite development and testing.