

Introduction to HTML and CSS

Lesson 1: **Introduction**

[The CodeRunner Screen](#)

[Working in CodeRunner](#)

[Using the CodeRunner® Editor and Saving Your Page](#)

[Saving and Retrieving Your Page](#)

[Making the Page More Exciting](#)

[You're Ready to Go!](#)

Lesson 2: **The Basics**

[The Head](#)

[The Body](#)

[Lists](#)

[CSS](#)

[Colors](#)

[Attributes](#)

[Void and Empty Elements](#)

[Comments](#)

Lesson 3: **Syntax and Semantics**

[Syntax](#)

[Tell the Browser You're Writing HTML with Doctype](#)

[Use the Meta Tag](#)

[Close Your Elements with Closing Tags and Quote Your Attributes](#)

[Don't Use Deprecated Elements](#)

[Semantics](#)

[Structure Elements](#)

[Flow Elements](#)

[Block and Inline Display](#)

[Using <div> to Structure Your Page](#)

[Add Meaning to <div> with the Id Attribute](#)

[Styling with Id's](#)

[Use the Right Element for the Job](#)

Lesson 4: **Links**

[What is a Link?](#)

[Relative Links](#)

[Absolute Links](#)

[Link Attributes](#)

[Using the ID Attribute to Link within a Document](#)

[Styling Links](#)

Lesson 5: **Images**

[Putting an Image on a Page](#)

[What Kind of Image Should You Use?](#)

[Uploading Images to Your OST Account](#)

[Styling Images](#)

[The alt Attribute](#)

[Using Images as Links](#)

[Putting an Image in the Background](#)

Lesson 6: **Cascading Style Sheets: Introduction**

[CSS Properties](#)

[Styling with Classes](#)

[Styling with ids](#)

[When to Use a Class and When to Use an ID](#)

[Linking to CSS in an External File](#)

[Using the Style Attribute](#)

Lesson 7: **Cascading Style Sheets: The Box Model**

[Padding, Margins, and Border for Your Content](#)

[Having Fun with Padding, Margins and Border](#)

[Padding, Margins, and Borders on Inline Elements](#)

Lesson 8: **Cascading Style Sheets: Inheritance**

[CSS Inheritance](#)

[CSS Descendant Selectors](#)

[Combining Descendant Selectors with IDs and Classes](#)

Lesson 9: **Cascading Style Sheets: Layout and Positioning**

[CSS Positioning](#)

Lesson 10: **HTML and CSS: Forms**

[Basic Input and Attributes](#)

[Placeholder Attribute](#)

[Required Attribute](#)

[Other Kinds of Inputs](#)

[Checkboxes and Radio Buttons](#)

[Textarea](#)

[Dates](#)

[Styling Forms with CSS](#)

[Where Do We Go from Here?](#)

Lesson 11: **HTML and CSS: Tables**

[Creating a Table](#)

[Table Headers](#)

[Captions](#)

[Spanning Multiple Columns](#)

[Styling Tables](#)

[Tables and Forms](#)

Lesson 12: **HTML and CSS: Standards and Validation**

[Validating Your Pages](#)

[All About Standards](#)

[Accessibility standards](#)

[Access Keys](#)

[Create a Great Web Site and Keep It Current](#)

[Just a Few Loose Ends](#)

[Favicons](#)

[More <meta> Tags](#)

Lesson 13: **[HTML and CSS: Final Project](#)**

[Final Project](#)

Copyright © 1998-2014 O'Reilly Media, Inc.



*This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.
See <http://creativecommons.org/licenses/by-sa/3.0/legalcode> for more information.*

Introduction

Welcome to the O'Reilly School of Technology (OST) course on HTML and CSS! In this course, you will learn to create an attractive and organized website using basic and intermediate HTML and CSS.

Course Objectives

When you complete this course, you will be able to:

- create basic HTML elements such as hyperlinks, images, tables, and forms.
- structure a web page effectively.
- control the look and placement of HTML elements using Cascading Style Sheets.
- demonstrate knowledge of box properties and external style sheets.
- create HTML source code that is both readable and upholds HTML5 standards.
- develop a full-fledged website using HTML5 and CSS.

From beginning to end, you'll learn by doing your own HTML and CSS based projects. These projects, as well as the final project, will add to your portfolio and provide needed experience. Besides a browser and internet connection, all software is provided online by the O'Reilly School of Technology.

Learning with O'Reilly School of Technology Courses

As with every O'Reilly School of Technology course, we'll take a *user-active* approach to learning. This means that you (the user) will be active! You'll learn by doing, building live programs, testing them and experimenting with them—hands-on!

To learn a new skill or technology, you have to experiment. The more you experiment, the more you learn. Our system is designed to maximize experimentation and help you *learn to learn* a new skill.

We'll program as much as possible to be sure that the principles sink in and stay with you.

Each time we discuss a new concept, you'll put it into code and see what YOU can do with it. On occasion we'll even give you code that doesn't work, so you can see common mistakes and how to recover from them. Making mistakes is actually another good way to learn.

Above all, we want to help you to *learn to learn*. We give you the tools to take control of your own learning experience.

When you complete an OST course, you know the subject matter, *and* you know how to expand your knowledge, so you can handle changes like software and operating system updates.

Here are some tips for using O'Reilly School of Technology courses effectively:

- **Type the code.** Resist the temptation to cut and paste the example code we give you. Typing the code actually gives you a feel for the programming task. Then play around with the examples to find out what else you can make them do, and to check your understanding. It's highly unlikely you'll break anything by experimentation. If you *do* break something, that's an indication to us that we need to improve our system!
- **Take your time.** Learning takes time. Rushing can have negative effects on your progress. Slow down and let your brain absorb the new information thoroughly. Taking your time helps to maintain a relaxed, positive approach. It also gives you the chance to try new things and learn more than you otherwise would if you blew through all of the coursework too quickly.
- **Experiment.** Wander from the path often and explore the possibilities. We can't anticipate all of your questions and ideas, so it's up to you to experiment and create on your own. Your instructor will help if you go completely off the rails.
- **Accept guidance, but don't depend on it.** Try to solve problems on your own. Going from misunderstanding to understanding is the best way to acquire a new skill. Part of what you're learning is problem solving. Of course, you can always contact your instructor for hints when you need them.
- **Use all available resources!** In real-life problem-solving, you aren't bound by false limitations; in OST courses, you are free to use any resources at your disposal to solve problems you encounter: the Internet, reference books, and online help are all fair game.
- **Have fun!** Relax, keep practicing, and don't be afraid to make mistakes! Your instructor will keep you at it until you've mastered the skill. We want you to get that satisfied, "I'm so cool! I did it!" feeling. And you'll have some projects to show off when you're done.

Lesson Format

We'll try out lots of examples in each lesson. We'll have you write code, look at code, and edit existing code. The code will be presented in boxes that will indicate what needs to be done to the code inside.

Whenever you see white boxes like the one below, you'll *type* the contents into the editor window to try the example yourself. The CODE TO TYPE bar on top of the white box contains directions for you to follow:

CODE TO TYPE:

White boxes like this contain code for you to try out (type into a file to run).
If you have already written some of the code, new code for you to add looks like this.
If we want you to remove existing code, the code to remove ~~will look like this~~.
We may also include instructive comments that you don't need to type.

We may run programs and do some other activities in a terminal session in the operating system or other command-line environment. These will be shown like this:

INTERACTIVE SESSION:

The plain black text that we present in these INTERACTIVE boxes is provided by the system (not for you to type). The commands we want you to type look like this.

Code and information presented in a gray OBSERVE box is for you to *inspect* and *absorb*. This information is often color-coded, and followed by text explaining the code in detail:

OBSERVE:

Gray "Observe" boxes like this contain **information** (usually code specifics) for you to observe.

The paragraph(s) that follow may provide addition details on **information** that was highlighted in the Observe box.

We'll also set especially pertinent information apart in "Note" boxes:

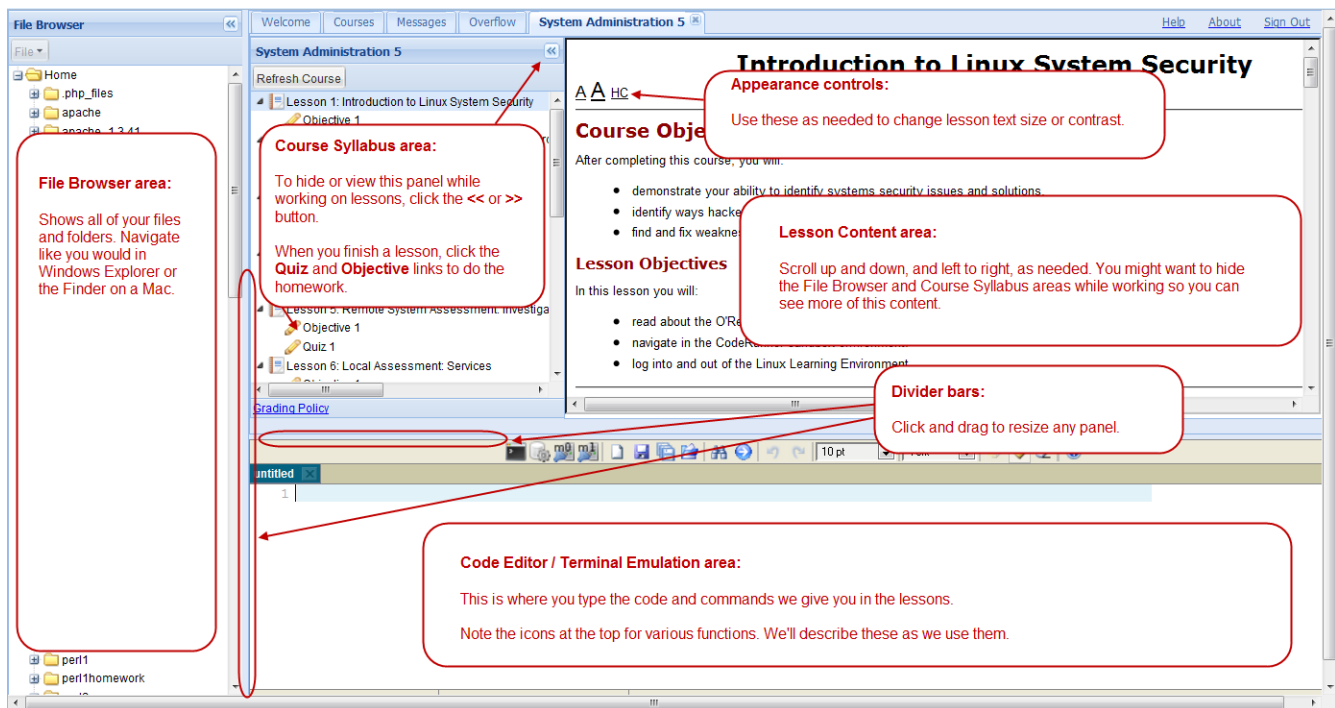
Note Notes provide information that is useful, but not absolutely necessary for performing the tasks at hand.

Tip Tips provide information that might help make the tools easier for you to use, such as shortcut keys.

WARNING Warnings provide information that can help prevent program crashes and data loss.

The CodeRunner Screen

This course is presented in CodeRunner, OST's self-contained environment. We'll discuss the details later, but here's a quick overview of the various areas of the screen:



These videos explain how to use CodeRunner:

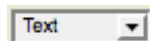
[File Management Demo](#)

[Code Editor Demo](#)

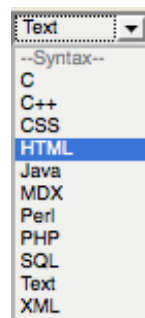
[Coursework Demo](#)

Working in CodeRunner

CodeRunner is a multi-purpose editor; you'll need to use the appropriate **Syntax** when you use it. In this course, you'll usually use **HTML** syntax. The CodeRunner Syntax menu looks like this:



Choose the HTML option:

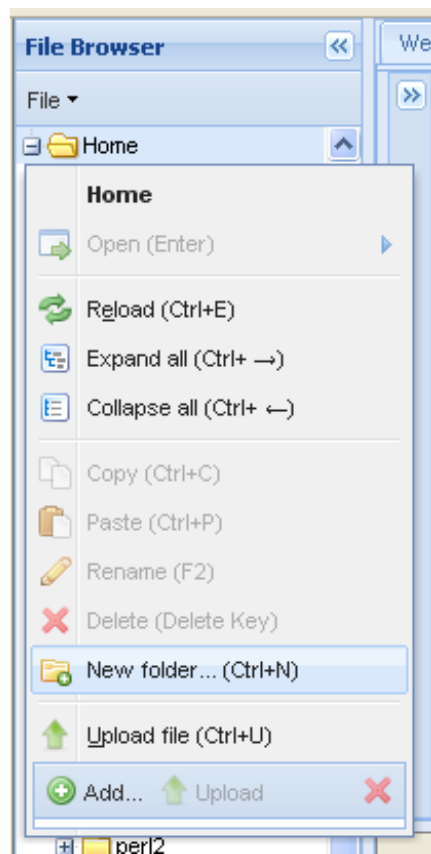


We'll give you additional tips on using the learning environment as needed throughout the course.

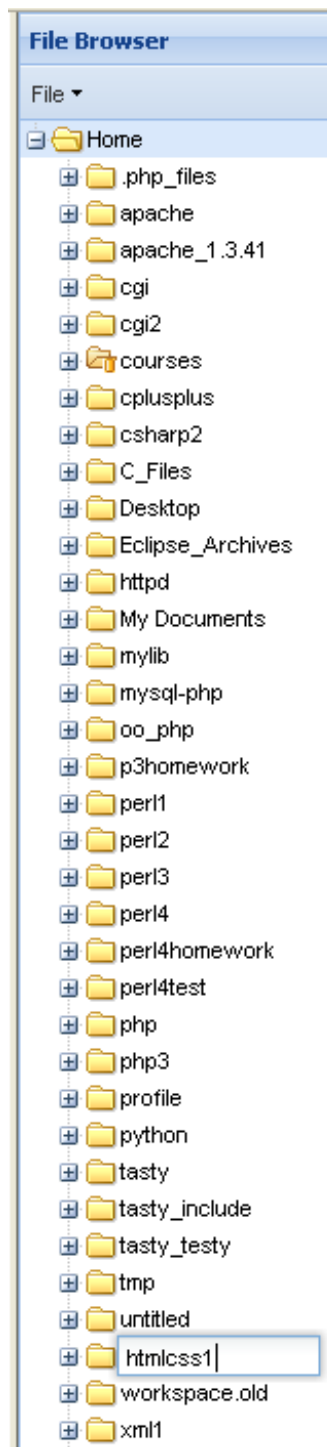
Using the CodeRunner® Editor and Saving Your Page

In this course, you'll learn the language of the web—**HTML** (HyperText Markup Language) and **CSS** (Cascading Style Sheets)—by making your very own web page right here, right now. Let's get started!

First, in CodeRunner, create an **htmlcss1** folder, to hold your work for this course. In the File Browser area to the left, right-click the **Home** folder and select **New folder...**:



Then, enter the folder name **htmlcss1**:




Note You'll see different (and probably fewer) folders in your environment.

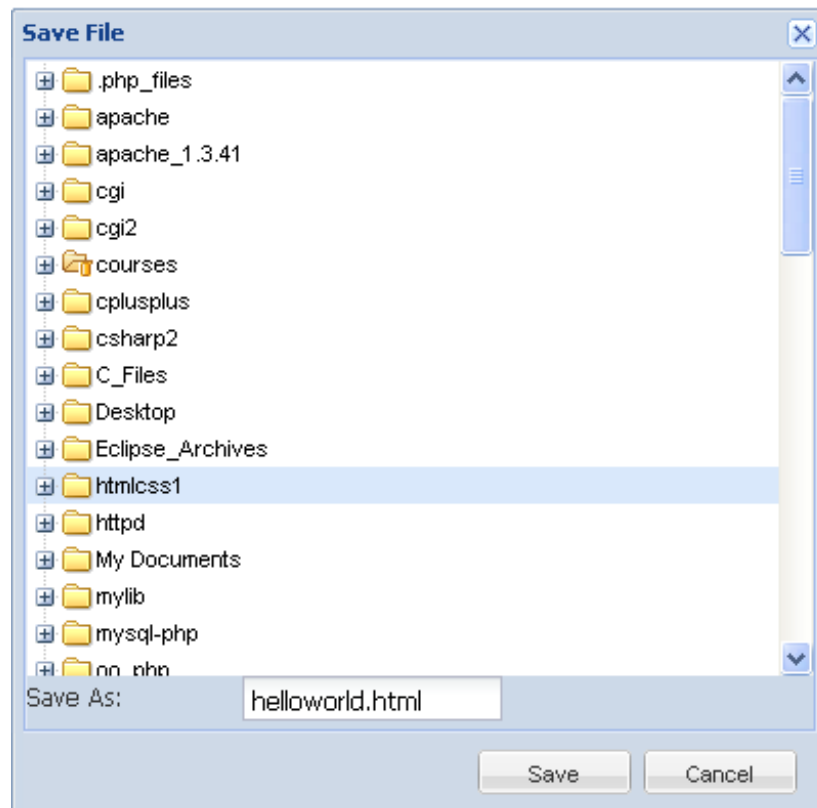
In the bottom half of the window, select **HTML** in the **Syntax** drop-down, and type the code as shown:


CODE TO TYPE:

```
Hello, World!
```

Saving and Retrieving Your Page


Now, save your page on the OST server by clicking the  button. Select the **htmlcss1** folder we just created, and enter the name **helloworld.html** (you need to include the **.html** extension or the browser won't know to treat your code as html):





Also practice using the Save as  button, entering a different name, to save another version of your file.

And there you go, page saved! After saving your file successfully, anybody on the web can type in the URL **<http://yourusername.oreillystudent.com/htmlcss1/myfilename.html>** (with your OST user name in place of *yourusername*) into their browser, and see your page. Pretty cool, right?

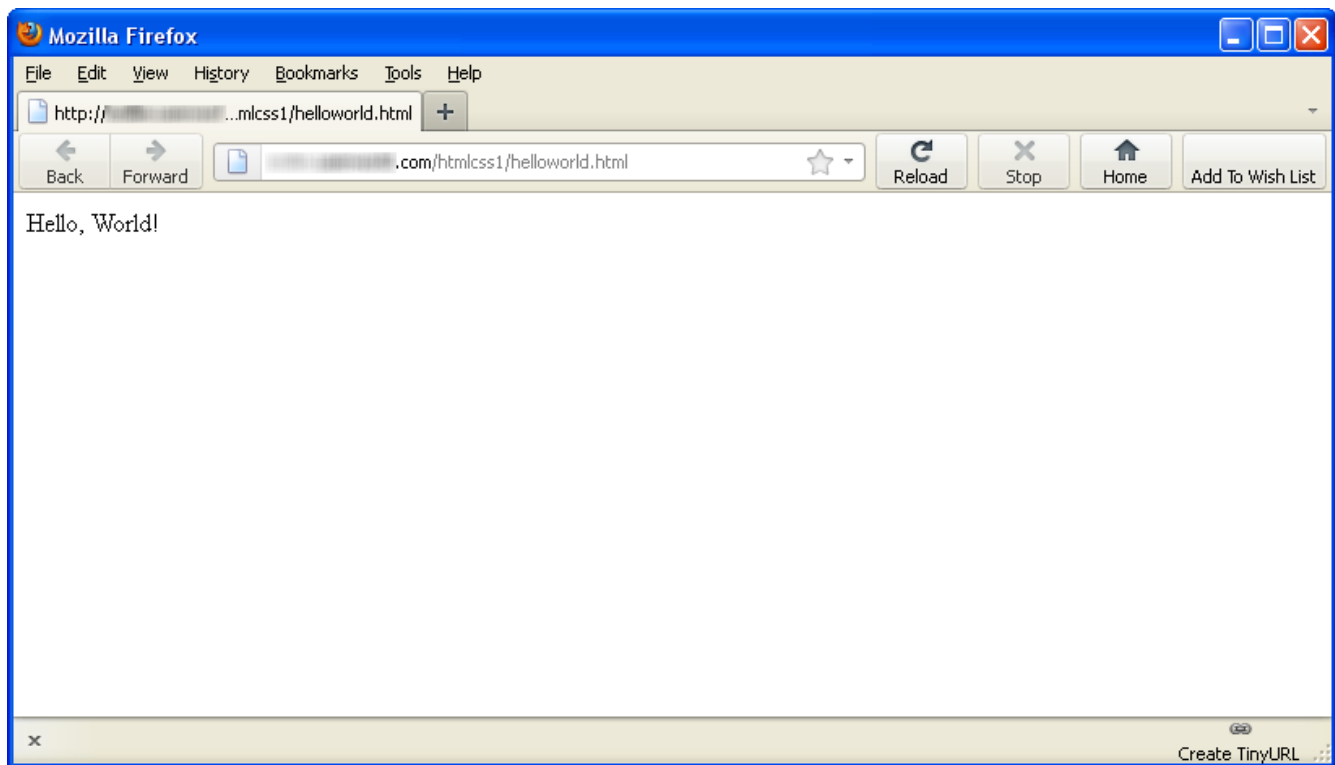
To retrieve your file later, click on the Load File icon , or double-click the file name in your **htmlcss1** folder in the File Browser area to the left.


To preview your file, click **Preview** .

When you preview a file, CodeRunner checks first to see whether this file has been saved previously. If it has, the page will be saved with the same name. If not, the Save File window will open.

Note Whenever you **Preview** an html file, your changes will be saved. If you don't want your previous code overwritten, use  before you preview. Even after your file has been saved, you can use the **Go Back** button  icon to retrieve the previous version.

Once the file has been saved, another browser window or tab should open and show you your first web page:




Anytime you click **Preview** , you'll see your code displayed in a new browser window (or tab). This will enable you to try examples and experiment on your own.

Making the Page More Exciting

Of course, as web pages go, this one is not exactly riveting. Let's make it more interesting by adding some *HTML tags*. Change the file as shown:

CODE TO TYPE:

```
<H2>Hello, World!</H2>
```

Click **Preview** . What happened? The text should have gotten bigger and bolder. We added *tags* to our page. Tags tell the web browser what to do with the text. They are the "**Markup**" part of the HyperText Markup Language. I'll tell you more about web browsers and HTML later. For now, let's have some fun!

Try using some other heading tags. There are six such elements: **H1**, **H2**, **H3**, **H4**, **H5**, and **H6**. (Replace the existing code so you see the effect of the different headings when you preview). Modify your file as shown:

CODE TO TYPE:

```
<H1>Heading 1</H1>
<H2>Heading 2</H2>
<H3>Heading 3</H3>
<H4>Heading 4</H4>
<H5>Heading 5</H5>
<H6>Heading 6</H6>
Hello, World!
```

Click **Preview** . It'll look like this:

Heading 1

Heading 2

Heading 3

Heading 4

Heading 5

Heading 6


Note

Sometimes, a browser may not "register" changes to a page because the page content is *cached* (stored to, and reloaded from, memory). If you have any difficulty seeing changes you've made, try reloading the page or [clearing the cache in your web browser](#).

Each pair of tags tells the web browser what to do with the text in between the **beginning** (`<H2>`) tag and the **ending** (`</H2>`) tag. For example, we can use a set of **center** tags to center some of the text we've written. Did you notice that the highest level heading, `<h1>` is bigger than the lowest level heading, `<h6>`? That's because the browser is applying some default *style* to the headings using Cascading Style Sheets (CSS). But you aren't stuck with the browser's default style for eternity; you can create our own styles, also using CSS. For example, you can add a style to a header to change the font and color. Edit the file as shown:

CODE TO TYPE:

```
<H1 style="font-family: Arial; color: blue;">Heading 1</H1>
<H2>Heading 2</H2>
<H3>Heading 3</H3>
<H4>Heading 4</H4>
<H5>Heading 5</H5>
<H6>Heading 6</H6>
Hello, World!
```

Click **Preview**  again, and you should see a change to your `<h1>` heading:

Heading 1

Heading 2

Heading 3

Heading 4

Heading 5

Heading 6

A few questions to ponder:

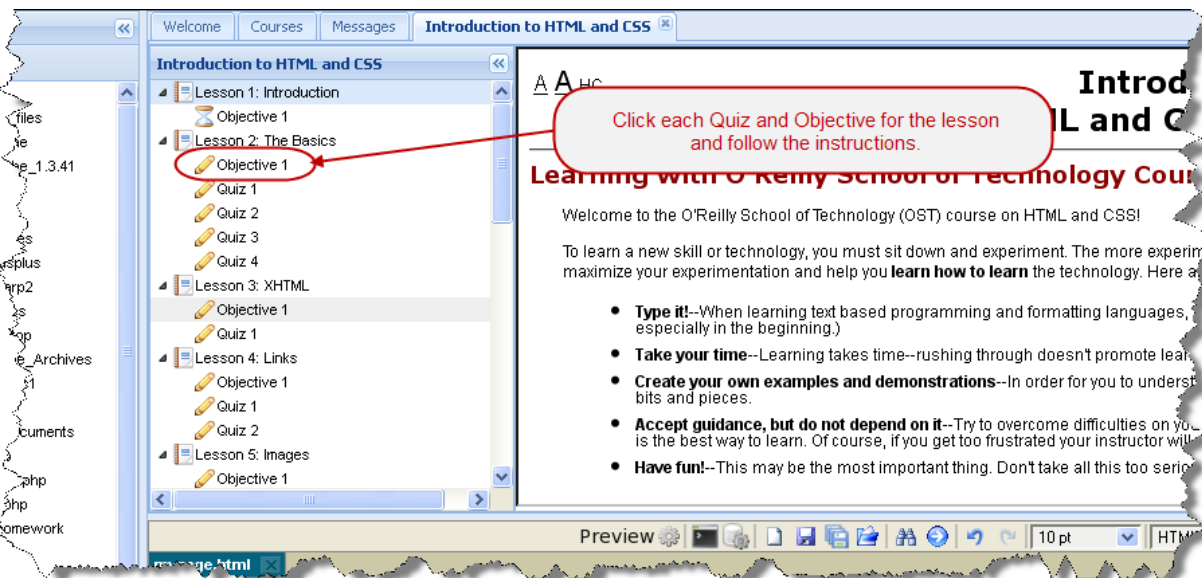
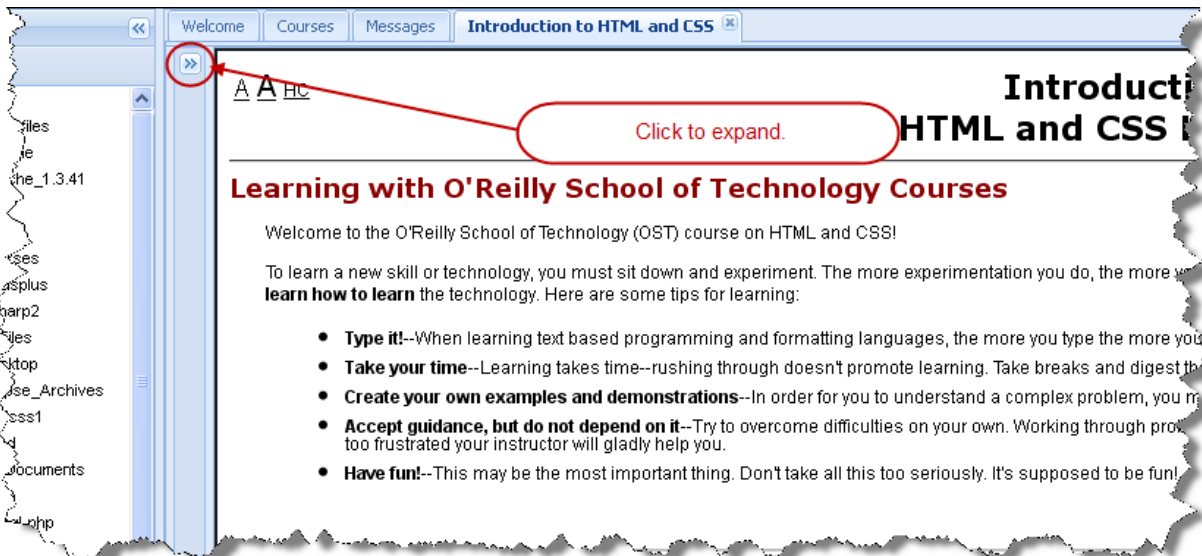
- Do spaces between words matter much?
- What does adding carriage returns (repeatedly hitting the **Enter** key) do?
- What happens if you remove one of the ending tags? (Try removing the ending `</H2>` tag.)
- What are the arrows ("`<`" and "`>`") doing? Try putting any word in between them, and see what happens.
- What happens if you forget to type a semicolon after "Arial" in the style?
- What are some other ways you might like to style headings?

Now experiment and answer these questions for yourself!

You're Ready to Go!

At this point you're probably pretty comfortable using Preview and also with saving and retrieving your files. You're going to use this stuff a lot, so practice using it on your own until you feel confident and ready to move forward.

Once you finish each lesson, go back to the syllabus to complete the homework:



Copyright © 1998-2014 O'Reilly Media, Inc.



This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License. See <http://creativecommons.org/licenses/by-sa/3.0/legalcode> for more information.

The Basics

Lesson Objectives

When you complete this lesson, you will be able to:

- create a head and a body for a web page.
- use HTML tags in your web page.
- include basic CSS in your web page.
- incorporate comments, color, and other attributes into your web page, as well as *void elements*.

When we use programs like DreamWeaver, we can click buttons that insert HTML tags automatically and then see the resulting changes immediately. We call this **WYSIWYG** (**What You See Is What You Get**) fashion. So, if we can use tags that way, why bother to learn HTML at all?



Good question. Those editors are great if you want to create basic, no-frills web pages quickly. However, if you want to have better control over the presentation and functionality of your pages, you'll need to learn web programming techniques like CSS, JavaScript, and CGI. To learn those techniques, you'll need to learn HTML first and learn it well.

The Head

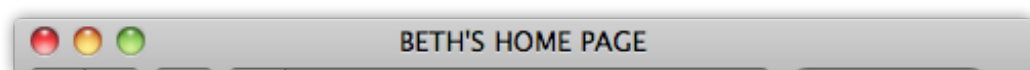
Let's go over a few new tags—the **head** tag and the **title** tag. Create a new HTML file as shown:

CODE TO TYPE:

```
<html>
<head>
<title>My Home Page </title>
</head>
</html>
```

Click  and save this file in your **htmlcss1** folder as **mypage.html**. Then, click  **Preview**. When the new window comes up, examine the title bar at the top of the window. What do you see?

You'll see "My Home Page" displayed in the title bar. You can change the title to anything you want. For example, the title of my web page looks like this:



Before we talk about what these tags do, notice that our tags are in pairs. For instance, **<head>** is an opening tag and **</head>** is a closing tag. In between the tags we have our *content*. The content is *contained* by the tags.

The opening tag, the content, and the closing tag combine to make an *element*. We call **<head>content</head>** the *head element*. So, **<head>** is an opening tag that is closed by the **</head>** tag. **<title>** is an opening tag that is closed by the **</title>** tag. All together **<title>My Home Page</title>** is called the *title element*.

Everything you write in an HTML page goes in the *html element* between the opening **<html>** and closing **</html>** tags.

The opening and closing tags tell the computer how to treat the text *contained* between them. Most elements have both opening and closing tags, but there are a few exceptions to this rule that we'll see later.

The Body

Okay, pay close attention here because I'm about to share one of the most important ideas for you to take from this lesson. Your HTML files will always have two parts: a **head** and a **body**. And, just like you, your page can have only *ONE head and ONE body*.

Every HTML file has this structure:

<head>



<---- Tags that are used within the head.

</head>

<body>



<---- Tags that are used within the body.

</body>




Most of the tags you use will be contained in the body, between **<body>** and **</body>**.

Update your code to include a **body** and an **h1** element:

CODE TO TYPE:

```
<html>
<head>
  <title> My Home Page </title>
</head>
<body >
  <h1>Welcome to My Home Page</h1>
</body>
</html>
```

Click **Preview** . You'll see the content of the **h1** element in the main part of the page, and the title (from the **head**) at the top of the browser window.

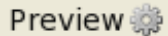
Lists

Let's get some more content into the body of this page. We'll create a list of favorite rock bands to add to the page. There are a few different ways to create a list in HTML; we'll start with an unordered list, ****. To put items in your list, you use the **** (line item) tag.

Try creating your own list—you can use your own favorite bands (if you aren't a fan of the hippie jam band, that's okay), favorite movies, or whatever you like. Place your list elements in an example like this:

CODE TO TYPE:

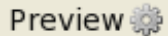
```
<html>
<head>
  <title> My Home Page </title>
</head>
<body >
  <h1>Welcome to My Home Page</h1>
  <p>
    My favorite live rock bands:
  </p>
  <ul>
    <li>Phish</li>
    <li>Blues Traveler</li>
    <li>Widespread Panic</li>
    <li>Pink Floyd</li>
    <li>Rolling Stones</li>
  </ul>
</body>
</html>
```

Click . The **ul** element contains many **li** elements, and is closed with a **** tag that comes after all of the **li** elements. Each of the **li** elements is closed with a **** tag.

So, what happens if you change **ul** to **ol**?:

CODE TO TYPE:

```
<html>
<head>
  <title> My Home Page </title>
</head>
<body >
  <h1>Welcome to My Home Page</h1>
  <p>
    My favorite live rock bands:
  </p>
  <ol>
    <li>Phish</li>
    <li>Blues Traveler</li>
    <li>Widespread Panic</li>
    <li>Pink Floyd</li>
    <li>Rolling Stones</li>
  </ol>
</body>
</html>
```

Click . The **ol** tag specifies an *ordered* list, so instead of bullets in front of the list items, you get numbers.


CSS

So far, so good! You've got a web page with some content. But what if you want to change the way your content *looks*? For instance, you might want to change the font of the text on the page, or add a background color, or even change the way the list items are displayed.

That's when we want to use Cascading Style Sheets (CSS). CSS is how you *style* your pages. Let's try some CSS now. We'll start by changing the background color of the page:

CODE TO TYPE:

```
<html>
<head>
  <title> My Home Page </title>
<style>
  body {
    background-color: #d9d9d9;
  }
</style>
</head>
<body >
  <h1>Welcome to My Home Page</h1>
  <p>
    My favorite live rock bands:
  </p>
  <ol>
    <li>Phish</li>
    <li>Blues Traveler</li>
    <li>Widespread Panic</li>
    <li>Pink Floyd</li>
    <li>Rolling Stones</li>
  </ol>
</body>
</html>
```

Click . Your page should have a light grey background now.

You're probably thinking that CSS looks a whole lot different from HTML. That's because it's a whole different language! But don't worry, I have every confidence that you'll be able to learn it. We'll go into a lot more detail about CSS later, but for now, keep these facts in mind:

Within the **<style>** element, **body** is the *selector*. It "selects" the element in your page to be styled. In this case, we selected the body to style, so the background of the entire page changed color.

background-color is the *property name*. This is the style property you're specifying. There are lots of properties you can use to style your elements.

#d9d9d9 is the *property value*. This value changes the style of your element. In this case, the property value is a color. I know—it doesn't *look* like a color, but it is. (More on that subject in a minute).

The property name and property value are separated by a colon (:). At the end of the line is a semicolon (;). Don't forget the semicolon (;)! It's a common mistake and if you make it, your CSS won't work. If you run into problems, check to make sure your semicolons are in order.

All of the CSS is contained in the **<style>** element. This is typically where you'll put your style, but you can also add style directly to an element, or put all your style into a separate file altogether. (You'll see how to do this later too.)

So, how do you think you'd set the background color of the **ol** element? Here's a hint: the color code for white is: #ffffff.

Colors

The six characters following the **#** are the hexadecimal representation of the colors red, green, and blue. To get other colors, all you have to do is change each character to one of sixteen values: **0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F**. Try changing these characters and see what kind of colors you can make. The first two characters determine how much **red** is used. If you do not want any red in the color, set the value of the first two characters to 00. If you want to incorporate just a little red, then the value to use would be 01. A little more? Use 02! You get the idea.

The second two characters determine how much **green** will be used, and the last two characters determine how much **blue** will be used.

If you want a bright red color, you would want to use the most red possible (and the least amounts of blue and green), so you would use **#ff0000**.

If you want less red, you might use **#770000**.

Now that you know more about colors, let's experiment a bit. Try setting the background color of the page to other

colors. Try setting the background color of the list to a different color.

What if you want to set the background color of an individual list item? Can you think of how you might do this?

Attributes

To set the background color of an individual list item (or to style it in any way) we need a way of selecting that item. If we used `li` as the selector, we'd wind up selecting *all* of the list items, rather than just one of them.

To determine how you'd style an individual list item (for example, if you wanted to set the background color of "Pink Floyd" to pink), you first need to know about attributes.

What's an attribute? It's an extra bit of information added to an element in the element's opening tag. **class** is an attribute you'll use often. Let's try adding a class to a list item and see how you can use that to style the list item.

CODE TO TYPE:

```
<html>
<head>
  <title> My Home Page </title>
  <style>
    body {
      background-color: #d9d9d9;
    }
    .pink {
      background-color: #ffcccc;
    }
  </style>
</head>
<body >
  <h1>Welcome to My Home Page</h1>
  <p>
    My favorite live rock bands:
  </p>
  <ol>
    <li>Phish</li>
    <li>Blues Traveler</li>
    <li>Widespread Panic</li>
    <li class="pink">Pink Floyd</li>
    <li>Rolling Stones</li>
  </ol>
</body>
</html>
```

In this example, you'll notice that we added an attribute named **class** with the value "pink" to the list item with the content "Pink Floyd" by typing:

```
<li class="pink">
```

The value of an attribute is always enclosed in quotation marks. In the CSS, we added a *class selector* by typing:

```
.pink {
background-color: #ffcccc;
}
```

Because we are using a class to select the element to be styled, rather than using the element's name, we need to put a . (period) in front of the class name in the CSS. That's how CSS knows it's a class and not an element.


Click  to see the pink background on the Pink Floyd list item. Why do you think the pink background extends all the way across the screen?

Experiment! What happens if you leave out the period? Try changing the background color of each of the items in the list using a class. What happens if you use the pink class on more than one list item?

Let's try one more style. Change the type of the list items using the CSS property **list-style-type** like this:

CODE TO TYPE:

```
<html>
<head>
  <title> My Home Page </title>
  <style>
    body {
      background-color: #d9d9d9;
    }
    li {
      list-style-type: square;
    }
    .pink {
      background-color: #ffcccc;
    }
  </style>
</head>
<body>
  <h1>Welcome to My Home Page</h1>
  <p>
    My favorite live rock bands:
  </p>
  <ul>
    <li>Phish</li>
    <li>Blues Traveler</li>
    <li>Widespread Panic</li>
    <li class="pink">Pink Floyd</li>
    <li>Rolling Stones</li>
  </ul>
</body>
</html>
```

Click **Preview** . Do you see the change made to the list item bullets? Try using these other types for unordered list items: **disc**, **circle**.

Now try these other types for ordered list items: **lower-roman** and **upper-alpha**. (Make sure you change your list from **ul** to **ol** for these).


There are many other attributes you can add to elements; we'll get to them in a bit.

Void and Empty Elements

There are some elements that have *only* opening tags, and no closing tags. They don't need a closing tag because they don't contain any text. Such elements are called *void elements*. An example of a void element is the **br** tag. **br** stands for **break**. Let's see what **br** does:

CODE TO TYPE:

```
<html>
<head>
  <title> My Home Page </title>
  <style>
    body {
      background-color: #d9d9d9;
    }
    li {
      list-style-type: square;
    }
    .pink {
      background-color: #ffcccc;
    }
  </style>
</head>
<body>
  <h1>Welcome to My Home Page</h1>
  <p>
    My favorite live rock bands:
  </p>
  <ul>
    <li>Phish</li>
    <li>Blues <br> Traveler</li>
    <li>Widespread Panic</li>
    <li class="pink">Pink Floyd</li>
    <li>Rolling Stones</li>
  </ul>
</body>
</html>
```

Click **Preview** . It should look something like this:

An **empty** element, on the other hand, is one that is designed to have content, but doesn't for some reason. For example, if you wrote:

OBSERVE:

```
<p></p>
```

you'd say that the `<p>` element is **empty**.

Comments

When you write HTML (or any other document or program), you should include comments to describe for others (or yourself, later!) what your intentions are. Comments in HTML are structured like this:

OBSERVE:

```
<!-- This is an HTML comment. -->
```

CSS comments look like this:

OBSERVE:

```
/*This is a comment*/
```

Comments do not appear in the rendered web page. Because of space considerations and to save you some typing, we won't always include comments in our examples, but in your real programming, you should comment early and often! Ultimately it saves time and improves the quality of your work. Add HTML and CSS comments to your code as shown:

CODE TO TYPE:

```
<html>
<head>
  <title> My Home Page </title>
  <style>
    body {
      /* Set the background color. */
      background-color: #d9d9d9;
    }
    li {
      list-style-type: square;
    }
    .pink {
      background-color: #ffcccc;
    }
  </style>
</head>
<body>
  <!-- Here's my heading. -->
  <h1>Welcome to My Home Page</h1>
  <p>
    My favorite live rock bands:
  </p>
  <ul>
    <li>Phish</li>
    <li>Blues <br> Traveler</li>
    <li>Widespread Panic</li>
    <li class="pink">Pink Floyd</li>
    <li>Rolling Stones</li>
  </ul>
</body>
</html>
```

Now that you know a little more about HTML, have fun using that new knowledge. Try using some other tags; for a list, check out this [w3schools page](#). See you in the next lesson!

Copyright © 1998-2014 O'Reilly Media, Inc.



This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.
See <http://creativecommons.org/licenses/by-sa/3.0/legalcode> for more information.

Syntax and Semantics

Lesson Objectives

When you complete this lesson, you will be able to:

- tell the browser you're writing HTML with doctype.
 - use the meta tag.
 - close your elements with closing tags and quote your attributes.
 - structure elements.
 - use semantics in HTML.
-

Now that you know the basics of HTML and CSS, you're ready to learn a more about the requirements for creating a *valid* HTML document, and to experiment with more of the elements you'll use to create *structure* and *semantics* in your HTML document.

Syntax

HTML has some rules that a document must follow in order to be valid. Valid according to whom, you might ask? Your HTML document must be valid according to the **World Wide Web Consortium**, or the W3C, the governing body that creates and manages the HTML and CSS standards that browser-makers implement. As you may know, the most recent version of HTML is HTML5, a standard that's currently still in development, but already implemented by most of the major browsers (Chrome, Firefox, Safari, Internet Explorer, and Opera). You can validate an HTML document at the W3C Markup Validation Service (we'll talk more about validation later in this course).

Note

Another version of HTML, XHTML 1.1, is also a current standard that is implemented by some browsers. XHTML, unlike HTML, is based on XML, which has very strict syntax rules. We're going to touch on XHTML at the end of this course, but we'll focus on HTML5 for most of our lessons.

Tell the Browser You're Writing HTML with Doctype

So, the first type of syntax we need to add to an HTML document to make it valid is a *doctype declaration* (*DTD*) to the top of our file. A doctype declaration tells the browser which type of HTML to expect in your web pages and which set of W3C standards you are adhering to in your files. For HTML5, it's really straightforward, because HTML5 is the default version of HTML which modern browsers use, and because HTML is *backward compatible*, older versions of HTML will still work in browsers that support newer versions like HTML5.

To inform your browser that you're writing HTML, you'll need to add one short line to the top of your document. If it's not already open, go ahead and open your **mypage.html** file, and modify it as shown:

CODE TO TYPE:

```
<!doctype html>
<html>
<head>
  <title> My Home Page </title>
  <style>
    body {
      /* Set the background color. */
      background-color: #d9d9d9;
    }
    li {
      list-style-type: square;
    }
    .pink {
      background-color: #ffcccc;
    }
  </style>
</head>
<body>
  <!-- Here's my heading. -->
  <h1>Welcome to My Home Page</h1>
  <p>
    My favorite live rock bands:
  </p>
  <ul>
    <li>Phish</li>
    <li>Blues <br> Traveler</li>
    <li>Widespread Panic</li>
    <li class="pink">Pink Floyd</li>
    <li>Rolling Stones</li>
  </ul>
</body>
</html>
```

Use the Meta Tag

You also need to add the **<meta>** tag to your HTML file to make it valid. You'll use the **<meta>** tag to help you describe your HTML file in a few different ways now, and in a variety of other ways later. For now, you need to add just one type of **<meta>** tag to your file to define the character set you're using to write your HTML:

CODE TO TYPE:

```
<!doctype html>
<html>
<head>
  <title> My Home Page </title>
  <meta charset="utf-8">
  <style>
    body {
      background-color: #d9d9d9;
    }
    li {
      list-style-type: square;
    }
    .pink {
      background-color: #ffcccc;
    }
  </style>
</head>
<body>
  <h1>Welcome to My Home Page</h1>
  <p>
    My favorite live rock bands:
  </p>
  <ul>
    <li>Phish</li>
    <li>Blues Traveler</li>
    <li>Widespread Panic</li>
    <li class="pink">Pink Floyd</li>
    <li>Rolling Stones</li>
  </ul>
</body>
</html>
```

UTF-8 is a Unicode character set that captures most of the characters we need to display most languages; it has become the standard that web pages use on the internet.

Close Your Elements with Closing Tags and Quote Your Attributes

We've already talked about closing your elements using closing tags (with certain exceptions for void elements), and you've seen that we always put quotation marks around the attributes on elements, like `class="pink"` and `charset="utf-8"`. Browsers can be pretty forgiving if you forget these rules, but for your pages to work exactly like you want them to, it's best to close your elements and quote your attributes.

Don't Use Deprecated Elements

One last syntax rule: don't use elements that are not defined in the current version of HTML. You may have played around with older versions of HTML and used elements like `` or even `<blink>` (yikes!). If you did, you need to forget those elements ever existed! Most deprecated elements were used to control how your page *looked* in the browser. Now that we have CSS, we do all of that with CSS properties, so all of those older elements have been taken out of HTML. Don't worry, if you used those elements in the past, you'll learn how to do everything you used to do with them using CSS instead (except `<blink>`, which was a horrible idea anyway).

Those are really all the syntax rules you need to worry about. You can handle that, right? When other rules pop up, we'll point them out. Let's review:

- Tell the browser that you're writing HTML with doctype.
- Use the meta tag to define the character set ("utf-8").
- Close your elements with closing tags and quote your attributes.
- Don't use deprecated elements.

Semantics

Okay, so you know the syntax rules, and you've experimented with some elements; now it's time to talk more about


the *meaning* of HTML. You understand that the `<h1>` element refers to a top level heading, and that the `` and `` elements refer to an unordered list and an ordered list, respectively. And you've probably guessed that the `<p>` element indicates a paragraph of text.

The meaning associated with different elements in HTML is called *semantics*. By marking up your content with elements, you're telling the browser, "hey, this is a heading," or "this is a paragraph," or even "this is an emphasized word" (the `` element), or "this is an address" (the `<address>` element).

While you type the following code, look at each new element and think about what it means:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title>My Home Page</title>
  <meta charset="utf-8">
  <style>
    body {
      background-color: #d9d9d9;
    }
    li {
      list-style-type: square;
    }
    .pink {
      background-color: #ffeccc;
    }
  </style>
</head>
<body>
  <section>
    <header>
      <h1>Ramblin' Times</h1>
      <h2>My Home Page</h2>
    </header>
    <div>
      <h1>My favorite live rock bands</h1>
      <h2><time datetime="2011-07-01">Thursday, July 1, 2011</time></h2>
      <p>
        I <em>love</em> listening to music. Here are some of my
        <strong>absolute</strong> favorites.
      </p>
      <ul>
        <li>Phish</li>
        <li>Blues Traveler</li>
        <li>Widespread Panic</li>
        <li class="pink">Pink Floyd</li>
        <li>Rolling Stones</li>
      </ul>
    </div>
    <footer>
      This blog is written by <address>Bob Loblaw</address> and
      his dog <address>Rufus</address>.
    </footer>
  </section>
</body>
</html>
```

Click . You've already seen how some of these elements (like `<h1>`, `<h2>` and ``) are displayed in the browser. What about the ones you haven't seen before—can you guess their meanings?

Structure Elements

We added the `lang` attribute to the `html` tag. Language tags are used to indicate the language of text or other items in HTML documents. The `lang` attribute can be applied to the entire document, as we did here, or to individual elements. In both cases, language information is inherited by elements nested inside the element with the declaration. The W3C recommendation is that you declare the primary language for each web page

with the lang attribute inside the html tag. This is meant to assist search engines and browsers.

Many of the elements used in the code you just typed are concerned with creating structure, as well as meaning, in your page. For instance, <section> is present to create a *section* of content. In this case, we have only one major area of content, so the section is also most of the content in the page. <h1> and <h2> are used to create headings, and you may have guessed that the <header> element is used to define the header of a page. You can have a header for your whole page, or if you have more than one section, you can have one header per section. The same applies to <footer>, which is used to present information like the author of the content it's related to, or links to related documents, and so on. Again, you can have a single footer per section or for the entire page.

So, about this <div> element. <div> has been around a long time (much longer than <section>, <header>, and <footer>, which are all new in HTML5) and it's really a generic element that *divides* up your page into logical groups of content. Even though elements like <section>, <header>, and <footer> do a lot of the work that <div> used to do, we still use <div> sometimes, for instance, when we want to group content together for styling and layout purposes.

Note

Unfortunately, as of this writing, the HTML5 structure elements <section>, <header>, <footer> (and a few others) are not well supported in all browsers yet, particularly in older versions of Internet Explorer (for example, IE 6, 7, and 8, which a lot of users are still using). We advise you to use <div> in place of these elements if you know your audience may be using one of those browsers. We'll show you how to do this in just a moment.

Flow Elements

Other elements, like , , <time>, and <address> are less about structure and more about meaning. They are typically found within blocks of text, like a paragraph (<p>). They tell your browser that, for example, Thursday, July 1, 2011 is a time, the word "love" is emphasized, and the word "absolute" is important.

The browser displays content in the element in *italic* style, and the content in the in **bold** style. This is the default style that the browser uses for these elements, but you can change that using CSS if you like. Just don't use the and elements because of the *look* they give the content; use these elements only for the *meaning* they give the content within them. You can make the content look any way you want it to with CSS!

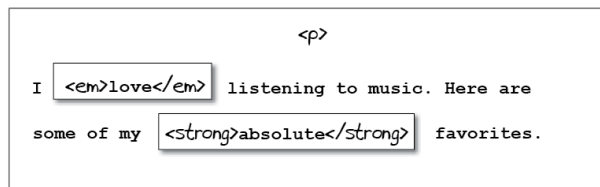
Block and Inline Display

<address> is a little different from the other flow elements. (Take a look at the page again [here](#).) Most browsers add a line break before and after <address> text.

By default, elements are displayed either as a *block*, meaning that they take up the entire width of the page, or *inline*, meaning that they flow in with the rest of the content on the line. Each block element takes up a rectangular space that fills the width of the page, like this:



The inline elements use just the space they need, like this:




So, `<h1>`, `<section>`, and `<div>` fill up the width of the page, while elements like `` and `` fit into the text of the page. So, a block element always has a line break before and after it because it takes up the entire width, forcing the content to stay strictly above and below it, never next to it. Well, "never" is a strong word, because we can change this with CSS!

Since `<address>` is forcing the other text around it to be strictly above and below the content in the address element, `<address>` must be a block element. Let's fix that using CSS. We're modifying just the head of our page this time, so we won't show the entire file here:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title>My Home Page</title>
  <meta charset="utf-8">
  <style>
    address {
      display: inline;
    }
  </style>
</head>
.
```

Click **Preview** . This changes the way the browser displays the content in the address element. By changing the default display from block to inline, you're telling the browser, "Don't put this content in a block, flow it in with the rest of the content in the same line."

Using <div> to Structure Your Page

If elements like <section>, <header>, and <footer> aren't the right elements for structuring your page, or if you need to add extra structure for layout and styling purposes only, then you'll want to use the trusty <div> element. It's a good all-purpose structuring element.

Note

You'll also need to use <div> to structure your page if your audience is still using older versions of browsers.

Let's rewrite the HTML using <div> tags:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title>My Home Page</title>
  <meta charset="utf-8">
  <style>
    address {
      display: inline;
    }
  </style>
</head>
<body>
  <div>
    <div>
      <h1>Ramblin' Times</h1>
      <h2>My Home Page</h2>
    </div>
    <div>
      <h1>My favorite live rock bands</h1>
      <h2><time datetime="2011-07-01">Thursday, July 1, 2011</time></h2>
      <p>
        I <em>love</em> listening to music. Here are some of my
        <strong>absolute</strong> favorites.
      </p>
      <ul>
        <li>Phish</li>
        <li>Blues Traveler</li>
        <li>Widespread Panic</li>
        <li>Pink Floyd</li>
        <li>Rolling Stones</li>
      </ul>
    </div>
    <div>
      This blog is written by <address>Bob Loblaw</address> and
      his dog <address>Rufus</address>.
    </div>
  </div>
</body>
</html>
```

Click



. Does using <div> change the way the browser displays this page by default? It shouldn't, because <div>, <section>, <header>, and <footer> are all structure elements that have block display by default, and no (or very little) default style added to them.

Note

Some styles, like the font size of headings and spacing between elements, will vary if you use `<div>` rather than one of the newer elements. In addition, different browsers may use different rules to apply default style to elements depending on any other elements in which they may be nested. But you have almost total control over how your elements are displayed using CSS. If you aren't happy with the way a browser styles elements by default, you can change it with CSS. And because of the discrepancies in default style rules, a common way to handle this is to use a **"CSS reset"**, which is a file containing CSS rules that set the default style to no style for every element. After a CSS reset, you can add selected CSS rules to build style for every element from scratch and make sure that your elements are displayed consistently on most browsers.


One problem with using `<div>` to structure your page though, is that the `<div>` tags start running together. It's easy to misplace a closing `</div>` tag or get mixed up about where each one starts and stops. Web developers often call this situation *div soup*. It's one of the primary reasons that new structure elements like `<section>`, `<header>`, and `<footer>` were added to HTML5.

Add Meaning to `<div>` with the Id Attribute

You can add more meaning, or semantics, to your elements using the *id attribute*:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title>My Home Page</title>
  <meta charset="utf-8">
  <style>
    address {
      display: inline;
    }
  </style>
</head>
<body>
  <div id="section">
    <div id="header">
      <h1>Ramblin' Times</h1>
      <h2>My Home Page</h2>
    </div>
    <div id="content">
      <h1>My favorite live rock bands</h1>
      <h2><time datetime="2011-07-01">Thursday, July 1, 2011</time></h2>
      <p>
        I <em>love</em> listening to music. Here are some of my
        <strong>absolute</strong> favorites.
      </p>
      <ul>
        <li>Phish</li>
        <li>Blues Traveler</li>
        <li>Widespread Panic</li>
        <li>Pink Floyd</li>
        <li>Rolling Stones</li>
      </ul>
    </div>
    <div id="footer">
      This blog is written by <address>Bob Loblaw</address> and
      his dog <address>Rufus</address>.
    </div>
  </div>
</body>
</html>
```

Click  **Preview**. You won't see any changes in the page, but you'll know that your `<div>` elements have more semantics. We used the names of the elements we used previously as the id's for the `<div>` elements (and added an id of "content" to the `<div>` we had before). To the browser, the id's have no meaning at all (except as a way to distinguish one `<div>` from another), so a `<div id="header">` is just a `<div>` to the browser,

unlike a `<header>`, which *does* have special meaning (another advantage of having these new elements)—it means "this is a header." Even though the id's have no meaning to the browser, they'll make your page a lot easier for *you* to understand. (Also note that the *indentation* we use helps to clarify which `<div>` tags go with which `</div>` tags.)


Styling with Id's

You already know how to select elements for styling with CSS using the element tag name, or with a class name using the class attribute on an element (earlier we used a class name of "pink" to give one of the list items a pink background color).

You can also use the id of an element to style it. Let's do that now. Again, we'll just show the top of the document, as the rest remains unchanged:

CODE TO TYPE:

```
<!doctype html>
<html>
<head>
<title>My Home Page </title>
<meta charset="utf-8">
  <style>
    address {
      display: inline;
    }
    div#section {
      border-width: 2px;
      border-style: solid;
      border-color: black;
    }
    div#header {
      border-width: 2px;
      border-style: dotted;
      border-color: blue;
      background-color: #ccd6f5;
    }
    div#content {
      border-width: 2px;
      border-style: dotted;
      border-color: green;
      background-color: #e6f5eb;
    }
    div#footer {
      border-width: 2px;
      border-style: dotted;
      border-color: red;
      background-color: #ffd6cc;
    }
  </style>
</head>
...
```

Click **Preview** . You see a solid black border around the `<div id="section">`, dotted blue, green, and red borders around the `<div id="header">`, `<div id="content">`, and `<div id="footer">` elements, respectively. What about the background colors on the header, content and footer `<div>` elements? Click [here](#) to see our version of the page and compare it to your own.

To style an element using its id with CSS, you'd type the element name, the "#" character, and then the id name as the selector. So **div#header** is the selector for the `<div id="header">` element.

Keep in mind that *each id in your document must be unique*. You can have only one element with an id of "header," one with an id of "content," one with an id of "footer," and so on. This is different from the class attribute; you can give as many elements as you want the same class. We'll talk a lot more about this in the CSS lesson later in the course, but for now, just remember: *id's MUST be unique*.

Use the Right Element for the Job

You've learned a lot in this lesson about syntax, page structure, and semantics—the meanings—of elements. There are lots of elements in HTML and the designers of the HTML standard have specific uses in mind for each one.

You've also learned that it's important to use elements for their meaning, not for how they make the content look on the page. Using CSS, we can manipulate the look of content however we want. This concept is called the *separation of presentation and content*. HTML is about marking up your content to add *meaning*; CSS is about adding *style* to your content for a particular look and feel.

Over time you'll become familiar with most of the elements in HTML, so you can use the right element for the job. Choose the most appropriate elements so they make sense to the browser and to you. When you go back to edit an old web page, you'll be glad you structured it well and used proper syntax and semantics.

Specs are never fun to read, but ultimately, the HTML specification contains the best information about the language. You can read the **HTML5 Specification: Edition for Web Authors** at <http://dev.w3.org/html5/spec-author-view/Overview.html>.

So far, so good. See you in the next lesson!

Copyright © 1998-2014 O'Reilly Media, Inc.



*This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.
See <http://creativecommons.org/licenses/by-sa/3.0/legalcode> for more information.*

Links

Lesson Objectives

When you complete this lesson, you will be able to:

- use relative links in HTML.
- use absolute links in HTML.
- incorporate some link attributes.
- use styling links in your web page.

What is a Link?

In this lesson, we'll learn one of the most important features of HTML—how to link documents together. The links in web pages allow us to surf the internet. When you're reading a document, and you click on text (often displayed in a different color and/or underlined) and it takes you to another document, that's a link.

Links are the "HT" in HTML. HTML is **HyperText** Markup Language. Hypertext is text that can be read non-linearly. You use the links in the text to jump around between pages which lets you read web content in a non-linear way.

Relative Links

Let's create some links. In order to do that, you'll need a file to link from and a file to link to, so we'll create two files. The first file will be called **pets.html** and the second, **tilla.html**.

To create the first file, click the New File icon as usual:



Enter the code as shown below:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
  <head>
    <title>My Pets</title>
    <meta charset="utf-8">
  </head>
  <body>
    <h1>My Pets</h1>
    <p>
      I have a dog and three cats:
    </p>
    <ul>
      <li><a href="tilla.html">Tilla Mighty Lupa</a></li>
      <li>Pickles</li>
      <li>Jack</li>
      <li>Annie</li>
    </ul>
  </body>
</html>
```

Save this in your **/htmlcss1** folder as **pets.html**. To create the second file, click the **New File** icon again. You'll see a second tab in the CodeRunner editor. Type the HTML code as shown:

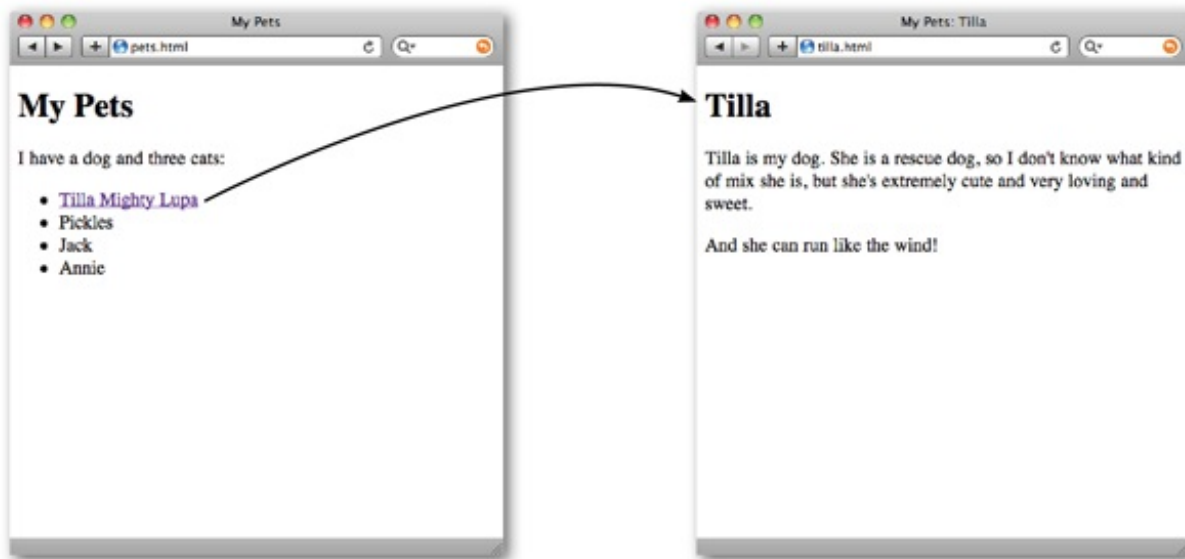
CODE TO TYPE:

```
<!doctype html>
<html lang="en">
  <head>
    <title>My Pets: Tilla</title>
    <meta charset="utf-8">
  </head>
  <body>
    <h1>Tilla</h1>
    <p>
      Tilla is my dog. She is a rescue dog, so I don't know what kind
      of mix she is, but she's extremely cute and very loving and sweet.
    </p>
    <p>
      And she can run like the wind!
    </p>
  </body>
</html>
```

Save this in your **/htmlcss1** folder as **tilla.html**. You can switch between the two files by clicking on the tabs. Just make sure you know which file you're in when you're making changes to the HTML code!

To make a link, you use the **<a>** (anchor) element. We'll create a link from **pets.html** to **tilla.html**.

Open **pets.html** and click **Preview** . When the new window opens, click on **Tilla Mighty Lupa**. When you do, you should see the content that you typed into the file **tilla.html**. You just created a link that links one document, **pets.html**, to another document, **tilla.html**. Like this:




How would you link *back* from **tilla.html** to **pets.html**? Before you peek at the code below, try to figure out how to do it yourself.

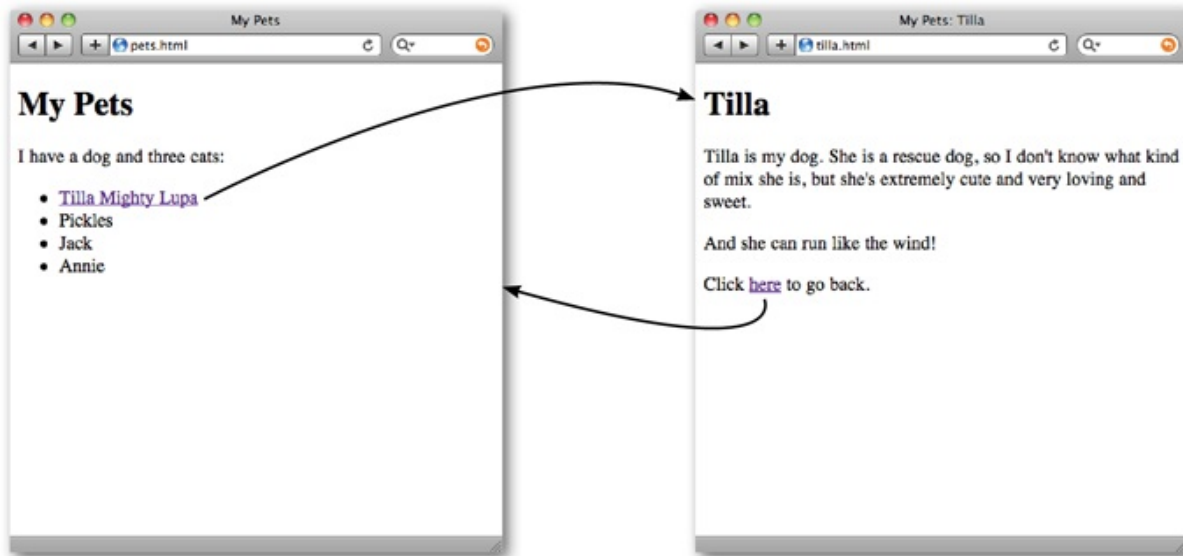
Okay, edit **tilla.html** to add the link. Modify your code as shown here:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
  <head>
    <title>My Pets: Tilla</title>
    <meta charset="utf-8">
  </head>
  <body>
    <h1>Tilla</h1>
    <p>
      Tilla is my dog. She is a rescue dog, so I don't know what kind
      of mix she is, but she's extremely cute and very loving and sweet.
    </p>
    <p>
      And she can run like the wind!
    </p>
    <p>
      Click <a href="pets.html">here</a> to go back.
    </p>
  </body>
</html>
```

Make sure you've saved your new code in the file **tilla.html**, then click **Preview** . You'll be able to click on the word "here" to get back to the first page, **pets.html**. Did it work?

You just created two links, one from **pets.html** to **tilla.html**, and one from **tilla.html** back to **pets.html**, like this:

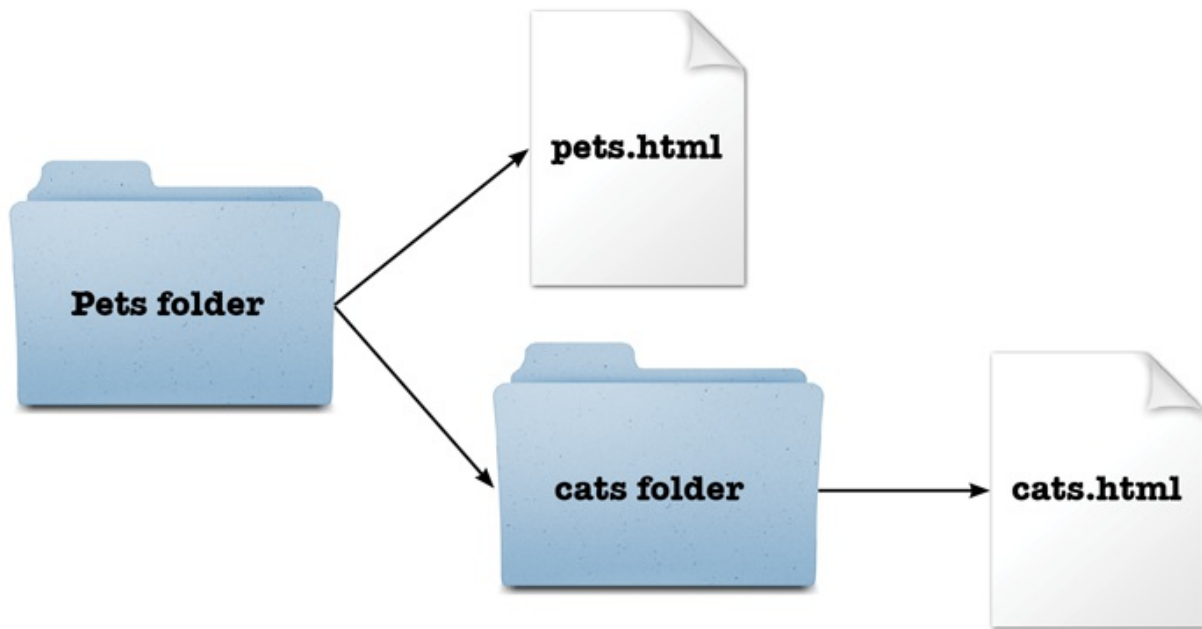


These are called *relative links* because the location specified for the linked file is *relative* to the location of the linking file—in this case, the files are in the same folder. You can also use a relative link to link to a file in a subfolder. Let's say you have another file called "cats.html" and you put it into the "/cats" subfolder. In that case, you'd write the link like this:

OBSERVE:

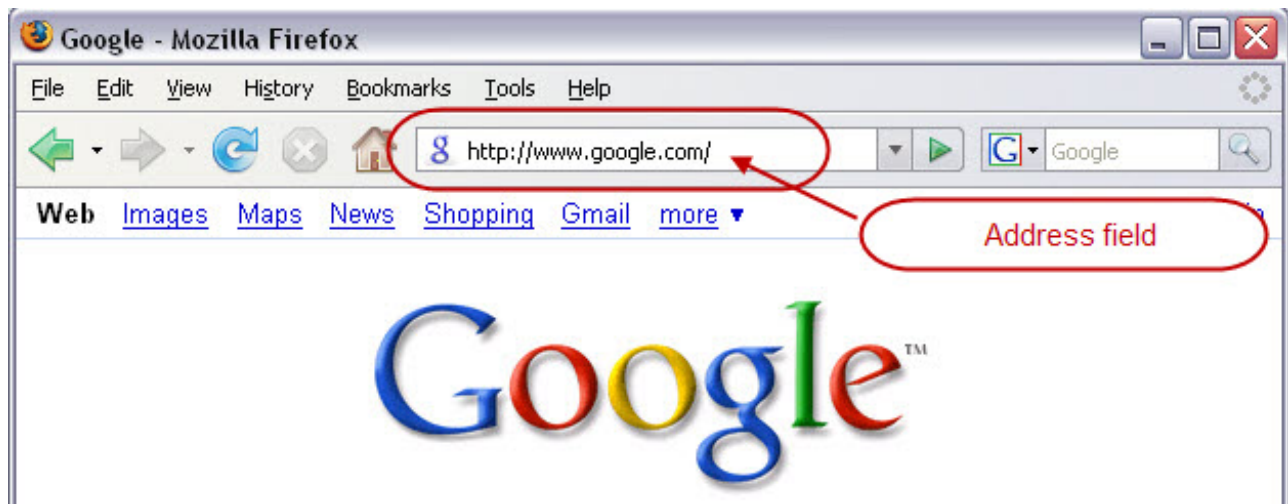
```
<a href="cats/cats.html">My cats</a>
```

You type the **directory name**, then a **slash ("/")** and then the **file name** to link to a file in a subdirectory. This is also a relative link. We specified only as much path information as is needed to find the linked file from where the linking file is located; the path "cats/cats.html" is *relative* to the directory where "pets.html" is located.




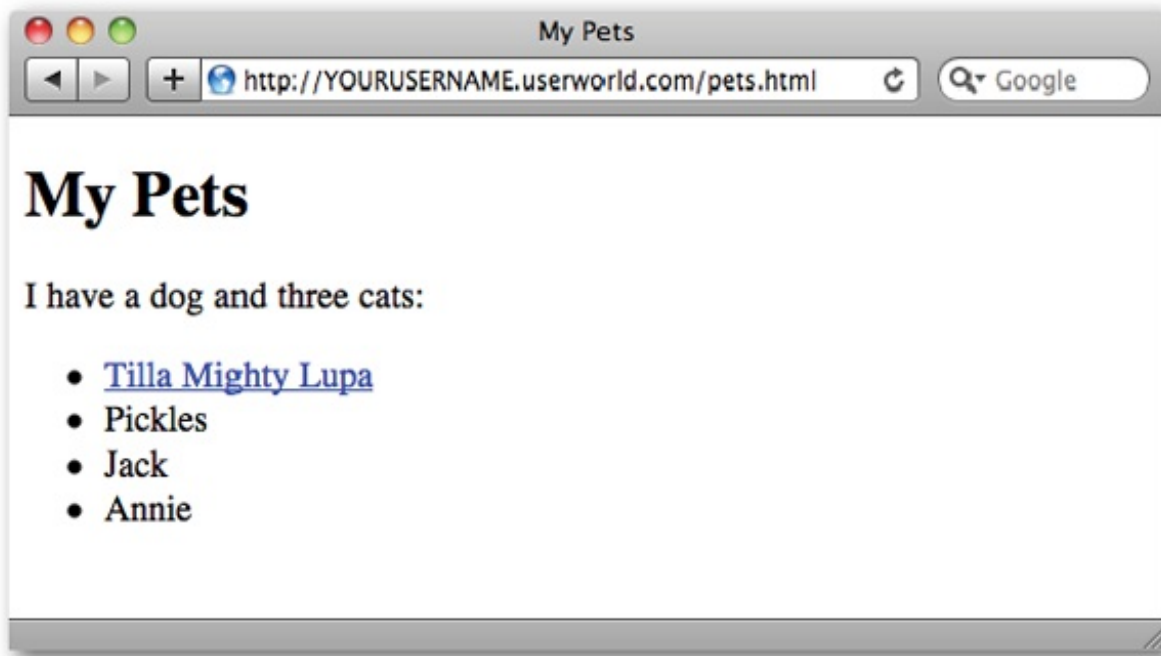
Absolute Links

You can usually use relative links when creating links between files on the same website, but if you want to link to a file that is outside of your domain (we'll talk more about domains in a bit), you need to use an *absolute* link, with a full web address. The address of a document that's on the web is located at the top of your browser window:



At the top of most browser windows, there's a text field that we refer to as the **Location** or **Address** field. This field shows the **URL** (Uniform Resource Locator) of the current web page.

Let's get the URL of your **pets.html** file. Open **pets.html** again, and click **Preview** . At the top of the browser window in the address field, you'll see the URL for your page. It should look something like this (except your username will be shown where it says "YOURUSERNAME" in the screen shot):



That URL is an *absolute web address* for your **pets.html** document. You can use this address to link to it from any other page, anywhere on the web!

Let's add an absolute link to a page on the web now:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title>My Pets</title>
  <meta charset="utf-8">
</head>
<body>
  <h1>My Pets</h1>
  <p>
    I have a dog and three cats:
  </p>
  <ul>
    <li><a href="tilla.html">Tilla Mighty Lupa</a></li>
    <li>Pickles</li>
    <li>Jack</li>
    <li>Annie</li>
  </ul>
  <p>
    Search for pets on <a href="http://www.google.com">Google</a>.
  </p>
</body>
</html>
```

Click **Preview** . Click on the link to Google. This will take you to the Google home page.

Now let's take a look at the parts that make up a URL:

OBSERVE:

```
http://www.google.com
http://www.oreillyschool.com/cert/htmlcss/tutorial/edit3.html
```

Almost every URL begins with **http://**. HTTP means **HyperText Transfer Protocol**. A protocol is a set of language rules that everybody agrees upon so that different computers can talk to each other.

www.oreillyschool.com/ and www.google.com are examples of *domain names* of web pages. A domain name is like your personal address: it indicates where on the internet your web page is located. Domain names are registered and stored on domain name servers all over the world, and these servers know how to look up every domain name on the internet. The suffix **.com** at the end of the domain name means the site is most likely a commercial one. There are many other suffixes—.edu usually indicates an education site; .org usually indicates a non-profit organization. There are also suffixes for countries; for example, .uk means a web site is registered in the United Kingdom, and .it means it's registered in Italy.

In the URL for the O'Reilly School, **cert/htmlcss/tutorial/** is the series of folders on the web site where the specified files are stored, and **edit3.html** is name of the HTML file to display in the browser window.

Note

You probably noticed that the Google URL didn't specify a file name at the end of the URL. When no file name is specified, it usually means one of two things: either a default file name is being used (such as `index.html`) or the web "page" is actually an application (for instance, a PHP script) rather than an HTML page.

Link Attributes

You've been using the **href** attribute to specify the URL of the file you're linking to in your `<a>` elements. **href** means **hypertext reference**, which means it contains a reference to another document available on the web.

Another attribute you might see sometimes in the `<a>` element is **title** (if you're looking at other people's HTML). This attribute was designed to contain text that helps describe the link. In theory, this could be helpful for people (and computers) reading your HTML code, as well as the visually impaired. However, in practice, it's actually better *not* to use the **title** attribute, because screen readers for the visually impaired do not handle this attribute well.

An attribute that *is* very useful for the visually impaired, however, is the **accesskey** attribute, which you'll learn about in a later lesson. For now, you'll use the **href** attribute, and, when you need to link to a specific part of your page, the **id** attribute, which you'll learn about next.


Using the ID Attribute to Link within a Document

You've used the **id attribute** to add meaning to an element and as a way to select elements for styling with CSS. You can also use the **id** attribute to create links within a document.

To see how that works, let's change the link in **pets.html** so that it links to another section of the document:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
  <head>
    <title>My Pets</title>
    <meta charset="utf-8">
  </head>
  <body>
    <h1>My Pets</h1>
    <p>
      I have a dog and three cats:
    </p>
    <ul>
      <li><a href="#tilla">Tilla Mighty Lupa</a></li>
      <li>Pickles</li>
      <li>Jack</li>
      <li>Annie</li>
    </ul>
    <p>
      Search for pets on <a href="http://www.google.com" title="Google search engine">G
oogle</a>.
    </p>
    <br><br><br><br><br><br><br>
    <br><br><br><br><br><br><br>
    <br><br><br><br><br><br><br>
    <br><br><br><br><br><br><br>
    <div id="tilla">
      Tilla is my awesome dog!
    </div>
  </body>
</html>
```

We changed the URL that links to the file `tilla.html` to `"#tilla"`. `#tilla` is the id of another element in the page, the `<div>` near the bottom of the page under all those `
` elements. We added all those `
`s (line breaks) to add space in the page (and make it a really long page) to help you to see how this works. Click [Preview](#) , and then click on the link for Tilla. Your page should scroll down to the `<div>` with the id of "tilla."

So using an id instead of a URL in the href attribute of the `<a>` element allows you to link to a particular location on the page. Just make sure you add id's to each of the elements to which you want to link. (Keep in mind that every id must be unique!) This is also a good way to create a table of contents in your page.


So, what would happen if you changed the id of the `<div>` element to `TILLA` instead of `tilla`? Do you think the link would still work? Give it a try and see for yourself!

Styling Links

Let's say you decide that you don't like the color of your links. Perhaps the default blue color clashes with your web page color scheme. In such cases, you can style your links using CSS. Go ahead and add some snazzy style to the links in `pets.html` now. Modify your code like this:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title>My Pets</title>
  <meta charset="utf-8">
  <style>
    a {
      color: #990000;
    }
  </style>
</head>
<body>
  <h1>My Pets</h1>
  <p>
    I have a dog and three cats:
  </p>
  <ul>
    <li><a href="#tilla">Tilla Mighty Lupa</a></li>
    <li>Pickles</li>
    <li>Jack</li>
    <li>Annie</li>
  </ul>
  <p>
    Search for pets on <a href="http://www.google.com">Google</a>.
  </p>
  <br><br><br><br><br><br><br>
  <br><br><br><br><br><br><br>
  <br><br><br><br><br><br><br>
  <br><br><br><br><br><br><br>
  <div id="tilla">
    Tilla is my awesome dog!
  </div>
</body>
</html>
```


Click **Preview** . Did your links change color? Here, we set the color of the text for all the `<a>` elements in the page to a darkish-red.

In the default color scheme, a link changes color from blue to purple after you click on it. When we set the color for the `<a>` elements above, we set the color for links you haven't clicked on yet, as well as those you have. We can change the color of links you've clicked on using CSS as well. Change your code as shown:

CODE TO TYPE:

```
<style>
  a {
    color: #990000;
  }

  a:visited {
    color: #000099;
  }
</style>
```

Here we just showed the style block; nothing else changes. Click **Preview**  again. Since you've clicked on all the links in the page recently, both of the links in our example should turn darkish-blue (the color identified by the hexadecimal 000099).


Now, you might be wondering about the **a:visited** selector we used in the CSS above. This is called a *pseudo-class*. A pseudo-class gives you access to part of an element based on a characteristic of that element—in this case, it's based on whether the `<a>` element has been visited recently.

There is another pseudo-class you can use with the `<a>` element that's kind of fun; the **a:hover** pseudo class. Give it a try. Add the code below to your style and see if you can guess what it means:

CODE TO TYPE:

```
<style>
a {
  color: #990000;
}
a:visited {
  color: #000099;
}

a:hover {
  background-color: #ffff66;
}
</style>
```

Click **Preview**  again. Try moving your cursor over the links in the page. Do you see a yellow background when your mouse pointer is **hovering** over the link? The **a:hover** psudeo-class sets the style of the link when your hover over it.

You have lots of different styling properties at your disposal in the pseudo-classes for the <a> element. Experiment with some of your own ideas too! If you need a few suggestions, see [the w3schools page on pseudo-classes](#).

You're doing really well so far. Keep it up and I'll see you in the next lesson!

Copyright © 1998-2014 O'Reilly Media, Inc.



This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.
See <http://creativecommons.org/licenses/by-sa/3.0/legalcode> for more information.

Images

Lesson Objectives

When you complete this lesson, you will be able to:

- upload images to your OST account.
- style your images.
- use images as links.

The internet is a vibrant and diverse place where you can find random funny cats, compromising celebrity photos, stunning views from space, and billions of other captivating visuals, thanks to our ability to display images on web pages.

Putting an Image on a Page

So, how do we get images onto our web pages? We use the **** element. Let's take a look at how **** works. Type the code as shown below:

CODE TO TYPE:

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Images</title>
  </head>
  <body>
    
  </body>
</html>
```

Save the file in your **/htmlcss1** folder as **imagetest.html** and click **Preview** . You'll see the OST logo:

OBSERVE: The tag

```

```

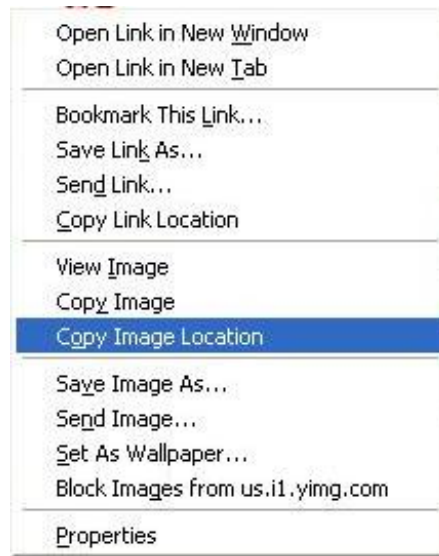
The **src** attribute specifies the URL of the image to display. Like an html file, an image file stored on a server has an address that you can use to access it. Notice that we used an *absolute* URL to access the OST logo; we can also use a *relative* URL to access an image in the same directory (or a subdirectory) as our HTML page. For example, if you had the OST logo in the same directory as your HTML, you could write **** instead.

Notice that the **** element has no closing tag. There is no text content in an **** element, so this element is **void**, and you only ever use an opening **** tag to add an image to your page.

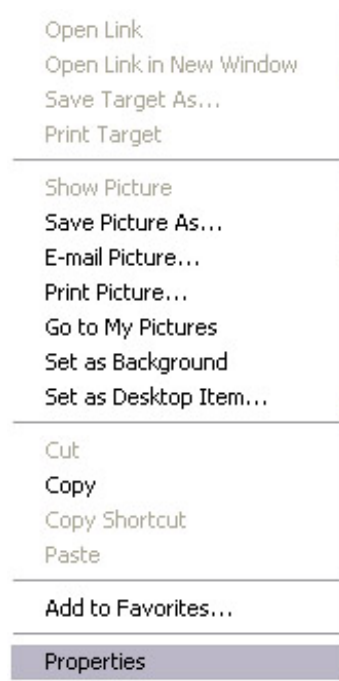
Let's try replacing the OST logo with a different image. Surf around the web until you find one you like (performing a search with Google images is a great way to find all kinds of cool images). Once you've found one, copy the URL of the image (select the URL and use **Ctrl+C** (PC) or **-C** (Mac)).

Right-click the image you want to use.

If you're using **Firefox**, you'll see a menu that looks like this:



If you're using **Internet Explorer**, you'll see a menu that looks like this:



Select **Properties** and then copy the address of the image from the window that pops up. When you're finished, come back to the CodeRunner® Editor and replace **`https://oreillyschool.com/images/OST_Logo.gif`** with the image address you just copied. (Paste it using **Ctrl+V** (PC) or **Command+V** (Mac).)

You can practice using this image:



Keep in mind that .gif, .jpg, and .png images are the only types that can be displayed on all browsers. If you want to display an image of another type (like .bmp, for instance), you'll need to convert the image to a valid type using software like Adobe Photoshop or Paint Shop Pro.

What Kind of Image Should You Use?

GIF works best for images with just a few solid colors and line drawings, like logos, clip art, and small text in images. One really nice feature of GIF images is that they support transparency, so the background color will show through and you can place the image over another element in your page for interesting effects. GIF images can be compressed, but

they are a *lossless* format, meaning no information is thrown away in compression.

JPEG (or JPG) images work best for photos—images with a lot of different colors and no text. JPEG images can represent a lot more colors than GIF (16 million instead of 256), so you'll get richer color. JPEG is a *lossy* format, meaning that when you compress images, you lose some information. However, for the web, it's unlikely anyone will notice unless they're viewing your images at very high resolution, which isn't advised anyway because the images would be so large. JPEG does not support transparency, so you aren't able to use them like GIFs to create effects where you see the background behind the image.

PNG is an image format that was designed to be a better version of GIF. The format is best used where you'd use a GIF—it supports transparency and represents lines and text well—but also can represent a lot more colors than GIF. Be careful when using PNG though, because PNG images tend to be large as a result of their greater depth of color (you may want to stick with JPEG for photos).

When you save an image for the web, it's a good idea to save the image in the size you plan to use on the page. So, for example, don't save an image at 800 pixels wide by 600 pixels high if you know you'll only need a 200x150 image on the web page. You can reduce the file size of an image by reducing its width and height. This reduction in size also means your images will load faster!

Uploading Images to Your OST Account

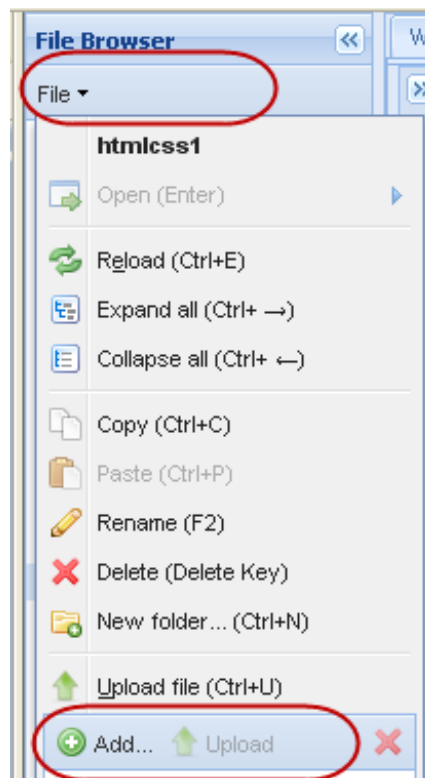
Note As you work through the next section, if you decide to use an image other than the one we've provided, be sure to choose an image that does not violate copyright laws. A quick search for **public domain images** or **creative commons images** should do the trick!

You can display images from other locations on your own page by linking to them using absolute URLs. You can also upload images (or any other file that is saved on your computer) to your OST account, and link to them using relative URLs instead.

Let's save an image that we can then upload to your account. Right-click on the image of the dog (Tilla) below and you'll see the same menu from before. Select **Save Image As/Save Picture As** and save the image to your computer (make a note of the location where you save it).



Next, select your `/htmlcss1` folder and click **File | Add** in the CodeRunner® **File Browser** window on the left:



Find **Tilla.JPG** and double-click it. **Tilla.JPG** appears in the **Add** field. Click the **Upload** icon:




(To clear the file from the Add menu, click the red "X").

Now that you have an image uploaded to your account, let's use it in a page. Open your **pets.html** from the last lesson, and modify it as shown:

CODE TO TYPE:

```
<!doctype html>
<html>
<head>
  <title>My Pets</title>
  <meta charset="utf-8">
  <style>
    a {
      color: #990000;
    }
  </style>
</head>
<body>
  <h1>My Pets</h1>
  <p>
    I have a dog and three cats:
  </p>
  <ul>
    <li><a href="#tilla" title="More info about Tilla">Tilla Mighty Lupa</a></li>
    <li>Pickles</li>
    <li>Jack</li>
    <li>Annie</li>
  </ul>
  <h1>Tilla</h1>
  <p>
    
    Tilla is my dog. She is a rescue dog, so I don't know what kind
    of mix she is, but she's extremely cute and very loving and sweet.
  </p>
  <p>
    And she can run like the wind!
  </p>
  <p>
    Search for pets on <a href="http://www.google.com" title="Google search engine">Google</a>.
  </p>
  <br><br><br><br><br><br><br>
  <br><br><br><br><br><br><br>
  <br><br><br><br><br><br><br>
  <br><br><br><br><br><br><br>
  <div id="tilla">
    Tilla is my awesome dog!
  </div>
</body>
</html>
```

Click **Preview** . You'll see your page with the image of Tilla in it.

Styling Images


You may have noticed that the image of Tilla is to the left of the paragraph that it's in, and that the text of the paragraph is aligned with the bottom of the image. Also, the paragraph after the one containing the image appears below the image. Can you think of why our image and text are formatted like this?

Remember that we have *inline* display and *block* display elements. The `` element is an inline display element, so it flows the image content in with the text of the paragraph in which it is contained. And of course, since `<p>` is a block display element, the content of the second paragraph must go below the first, including the image.

We can use CSS to have some control over where our images appear, as well as how they appear in the page. Let's begin by aligning the text of the paragraph it's in with the top of the image instead of the bottom. Modify **pets.html** as shown:

CODE TO TYPE:


```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>My Pets</title>
  <style>
    img {
      vertical-align: top;
    }
  </style>
</head>
<body>
  <h1>My Pets</h1>
  <p>
    I have a dog and three cats:
  </p>
  <ul>
    <li>Tilla Mighty Lupa</li>
    <li>Pickles</li>
    <li>Jack</li>
    <li>Annie</li>
  </ul>
  <h1>Tilla</h1>
  <p>
    
    Tilla is my dog. She is a rescue dog, so I don't know what kind
    of mix she is, but she's extremely cute and very loving and sweet.
  </p>
  <p>
    And she can run like the wind!
  </p>
</body>
</html>
```

Click **Preview** . The text in the paragraph next to the image is now aligned with the top of the image instead of the bottom. Try **middle** too.

So, what if you want the image over to the right instead of on the left? Let's try to make that happen. Modify **pets.html** as shown:

CODE TO TYPE:


```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>My Pets</title>
  <style>
    img {
      float: right;
      vertical-align: top;
    }
  </style>
</head>
<body>
  <h1>My Pets</h1>
  <p>
    I have a dog and three cats:
  </p>
  <ul>
    <li>Tilla Mighty Lupa</li>
    <li>Pickles</li>
    <li>Jack</li>
    <li>Annie</li>
  </ul>
  <h1>Tilla</h1>
  <p>
    
    Tilla is my dog. She is a rescue dog, so I don't know what kind
    of mix she is, but she's extremely cute and very loving and sweet.
  </p>
  <p>
    And she can run like the wind!
  </p>
</body>
</html>
```

Click **Preview** . Now the image should be over on the right. But you should also see that *all* of the text, not just the text of the one paragraph, is to the left of the image. That's because **float** takes the image out of the normal inline flow and "floats" the image element beside the other content of the element in which the image is contained.

Now, let's add a border to the image. Modify **pets.html** again, as shown:

CODE TO TYPE:

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>My Pets</title>
  <style>
    img {
      float: right;
      vertical-align: top;
      border: 5px solid black;
    }
  </style>
</head>
<body>
  <h1>My Pets</h1>
  <p>
    I have a dog and three cats:
  </p>
  <ul>
    <li>Tilla Mighty Lupa</li>
    <li>Pickles</li>
    <li>Jack</li>
    <li>Annie</li>
  </ul>
  <h1>Tilla</h1>
  <p>
    
    Tilla is my dog. She is a rescue dog, so I don't know what kind
    of mix she is, but she's extremely cute and very loving and sweet.
  </p>
  <p>
    And she can run like the wind!
  </p>
</body>
</html>
```

Click **Preview** . Do you see a border? We used a CSS shortcut to combine three properties into one:

OBSERVE:

```
border-width: 5px;
border-style: solid;
border-color: black;
```

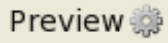
When you write it using the shortcut, it's important that you get the order right—width first, then style, then color.

The width is specified in *px*, or pixels. Other styles you can use include dotted, dashed, ridge, inset, and groove. The color is either a named color or a hexadecimal code for the red, green, and blue elements of a color. Go ahead and experiment with your image, changing sizes, styles, and colors.

We'll start with size here. Even though this image is 400px by 300px, you might want to display it at, say, 150px wide. We can change the display size of an image like this:

CODE TO TYPE:

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>My Pets</title>
  <style>
    img {
      float: right;
      vertical-align: top;
      border: 5px solid black;
      width: 150px;
    }
  </style>
</head>
<body>
  <h1>My Pets</h1>
  <p>
    I have a dog and three cats:
  </p>
  <ul>
    <li>Tilla Mighty Lupa</li>
    <li>Pickles</li>
    <li>Jack</li>
    <li>Annie</li>
  </ul>
  <h1>Tilla</h1>
  <p>
    
    Tilla is my dog. She is a rescue dog, so I don't know what kind
    of mix she is, but she's extremely cute and very loving and sweet.
  </p>
  <p>
    And she can run like the wind!
  </p>
</body>
</html>
```

Click . Did you see the image reduce in size a bit on the page? Keep in mind that even though the image *looks* like it's smaller, it's still from the same image file, just resized by the browser when it displays the page. This means the browser has to do more work (not much, but a little), and it's still *loading* the same-sized image file. If you make a big change to image size, then you'll want to resize it using an image manipulation program like Adobe Photoshop or Paint Shop Pro. But tweaking the size a little using CSS is fine.


Did you notice that we set only the width of the image in CSS? We can let CSS figure out the right height for the image to keep the image aspect ratio correct (which saves you a bit of time trying to figure out what height goes with a width of 150px). Try a few other widths. Also try changing both **width** and **height**.

The alt Attribute

So, what if the image doesn't load, or a user has images turned off or is visually impaired and is using a screen reader? If the image doesn't load, you'll see a broken image icon; many browsers also display the text in the **alt** attribute, which can help the user to understand what your image was meant to convey. Let's add an alt attribute to our image of Tilla. Modify your code as shown:

CODE TO TYPE:

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>My Pets</title>
    <style>
      img {
        float: right;
        vertical-align: top;
        border: 5px solid black;
        width: 150px;
      }
    </style>
  </head>
  <body>
    <h1>My Pets</h1>
    <p>
      I have a dog and three cats:
    </p>
    <ul>
      <li>Tilla Mighty Lupa</li>
      <li>Pickles</li>
      <li>Jack</li>
      <li>Annie</li>
    </ul>
    <h1>Tilla</h1>
    <p>
      
      Tilla is my dog. She is a rescue dog, so I don't know what kind
      of mix she is, but she's extremely cute and very loving and sweet.
    </p>
    <p>
      And she can run like the wind!
    </p>
  </body>
</html>
```


Click [Preview](#) . Because we changed the JPG extension to be lowercase, the browser can't find the image of Tilla and displays the alt text.

Using Images as Links

On occasion, you may want to use an image in a link. To do that, you'll use the `<a>` element and then use the image as the content of the element (that is, put the image between the opening `<a>` tag and closing `` tag). Let's add a link to a Google search for "dogs," so that when you click on the image of Tilla, you get taken to Google. Modify your code as shown:

CODE TO TYPE:

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>My Pets</title>
  <style>
    img {
      float: right;
      vertical-align: top;
      border: 5px solid black;
    }
  </style>
</head>
<body>
  <h1>My Pets</h1>
  <p>
    I have a dog and three cats:
  </p>
  <ul>
    <li>Tilla Mighty Lupa</li>
    <li>Pickles</li>
    <li>Jack</li>
    <li>Annie</li>
  </ul>
  <h1>Tilla</h1>
  <p>
    <a href="http://google.com/?q=dogs"></a>
    Tilla is my dog. She is a rescue dog, so I don't know what kind
    of mix she is, but she's extremely cute and very loving and sweet.
  </p>
  <p>
    And she can run like the wind!
  </p>
</body>
</html>
```

Click  and click the picture of Tilla. You could also use this kind of linking to provide a link to a larger version of the image.

Putting an Image in the Background

Now let's try to use an image for the background of our web page. As you might've guessed, you do this using a CSS property.


Save the image below (right-click the image and save to your computer), and upload it to your OST account (select your **/htmlcss1** folder in the File Browser's, select **File | Add**, select your image, click **Upload**, and then click the red X to dismiss):



Once you've got the image **plaster.jpg** uploaded, update your CSS style as shown:

CODE TO TYPE:

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>My Pets</title>
  <style>
    body {
      background: url(plaster.jpg) repeat;
    }
    img {
      float: right;
      vertical-align: top;
      border: 5px solid black;
    }
  </style>
</head>
<body>
  <h1>My Pets</h1>
  <p>
    I have a dog and three cats:
  </p>
  <ul>
    <li>Tilla Mighty Lupa</li>
    <li>Pickles</li>
    <li>Jack</li>
    <li>Annie</li>
  </ul>
  <h1>Tilla</h1>
  <p>
    <a href="http://google.com/?q=dogs"></a>
    Tilla is my dog. She is a rescue dog, so I don't know what kind
    of mix she is, but she's extremely cute and very loving and sweet.
  </p>
  <p>
    And she can run like the wind!
  </p>
</body>
</html>
```

Click **Preview** . Do you see the image in the background? Even though the image is small, it fills the background. We used the repeat property value when we set the background image on the body in the CSS rule, but the background image repeats by default anyway. Try changing the property so the image repeats only across the top or down the right side of the page. You'll learn more about the **background** CSS property later in the course.

Tip Go easy when using background images—a busy image can obscure other content on the page.

Experiment some more with images changing sizes, styles, and colors. When you're feeling comfortable working with images, move on to the next lesson...

Copyright © 1998-2014 O'Reilly Media, Inc.



This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.
See <http://creativecommons.org/licenses/by-sa/3.0/legalcode> for more information.

Cascading Style Sheets: Introduction

Lesson Objectives

When you complete this lesson, you will be able to:

- use CSS to style your web page.
- style your page using classes when appropriate.
- style your page using ids when appropriate.
- link to CSS in an external file.
- use the style attribute.

We've already used a bit of CSS in this course, but we haven't explored exactly what it is and how it works. We're ready to dig deeper now to find out what CSS is really all about!


CSS stands for "**C**ascading **S**tyle **S**heets." We use CSS language to describe the *style* of our pages. So, content is described using HTML elements to "mark up" your pages, while style is described by CSS properties that tell the browser how to display the content within those elements. Using HTML and CSS, we *separate content from presentation*.

CSS Properties

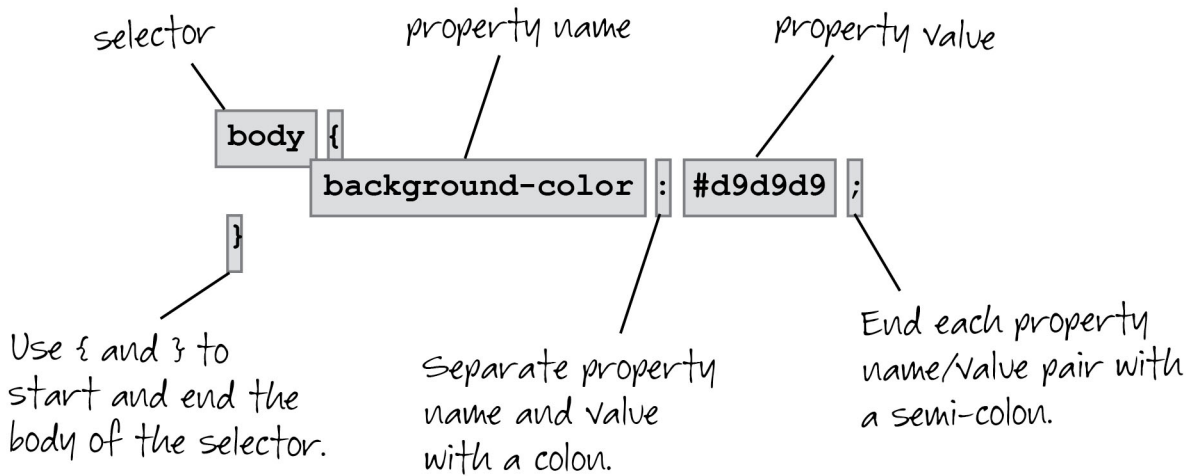
Let's take a closer look at CSS using one of our existing examples. (Just this once, I'll look the other way if you cut and paste the code):

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title> My Home Page </title>
  <meta charset="utf-8">
  <style>
    body {
      background-color: #d9d9d9;
    }
  </style>
</head>
<body >
  <h1>Welcome to My Home Page</h1>
  <p>
    My favorite live rock bands:
  </p>
  <ul>
    <li>Phish</li>
    <li>Blues Traveler</li>
    <li>Widespread Panic</li>
    <li>Pink Floyd</li>
    <li>Rolling Stones</li>
  </ul>
</body>
</html>
```

Save it in your `/htmlcss1` folder as `mybands.html`, and then click [Preview](#) . You may recall that we used CSS here to set the background color of the page to a light grey.

Now let's check out a CSS *rule*. A rule consists of a *selector* and one or more *properties*:



To select the element you want to style, specify a selector. In our case, we want to style the entire body of the page, so we use the name of the `<body>` element to select it. We don't use `<` and `>` to select the body, we just use the tag name, **body**.

All of the properties of `<body>` that we want to style go inside of the opening and closing brackets `{ }`.

Next, we specify which property of the body we want to style—the **background-color**.

Then we give that property a value by specifying a color for the background, for our example we'll use **#d9d9d9**, which is a light grey color. (Remember to separate the property name and value with a colon.)

Finally, we end the line with a semi-colon. We use semi-colons to separate properties within a single **rule**. Let's add another property now. Modify your code as shown:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title> My Home Page </title>
  <meta charset="utf-8">
  <style>
    body {
      background-color: #d9d9d9;
      font-family: Arial, Helvetica, sans-serif;
    }
  </style>
</head>
<body >
  <h1>Welcome to My Home Page</h1>
  <p>
    My favorite live rock bands:
  </p>
  <ul>
    <li>Phish</li>
    <li>Blues Traveler</li>
    <li>Widespread Panic</li>
    <li>Pink Floyd</li>
    <li>Rolling Stones</li>
  </ul>
</body>
</html>
```

Click **Preview** . Did the font change?

We specified the font for the page using the **font-family** property. The **font-family** property relies on fonts being available on the computer where the web page is viewed. We specify multiple font names, separated by commas. This

gives the browser more than one font option, that way if a particular font is not supported, it can choose another. The browser will try each font in turn, from left to right, and use the first one that is supported on the viewing computer.

Most computers can use these fonts: Times, Arial, Helvetica, and Courier. If none of those fonts are available, you can specify a generic name for a font. In the property above, we're telling our computer that, "if Arial or Helvetica are not available, then pick the default sans-serif font to display the page." A sans-serif font is a font without the serifs (curls or tips) on the letters.


Go ahead and change the first font in the list above. For example, change Arial to Times. Try Courier. Try taking the font names off completely and just specifying one of the generic fonts available on most computers:

- **serif** e.g. *Times, New York, Garamond*
- **sans-serif** e.g. *Arial, Helvetica, Geneva*
- **cursive** e.g. *Zapf Chancery, Ribbon*
- **fantasy** *Decorative fonts such as Western*
- **monospace** e.g. *Courier, or Monaco*

We can also change the look of our page completely using CSS. Modify your code as shown:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title> My Home Page </title>
  <meta charset="utf-8">
  <style>
    body {
      background-color: black;
      font-family: monospace;
      color: white;
    }
  </style>
</head>
<body >
  <h1>Welcome to My Home Page</h1>
  <p>
    My favorite live rock bands:
  </p>
  <ul>
    <li>Phish</li>
    <li>Blues Traveler</li>
    <li>Widespread Panic</li>
    <li>Pink Floyd</li>
    <li>Rolling Stones</li>
  </ul>
</body>
</html>
```

Click  and see the changes that we made to our page.

We set the background-color property to black, set the foreground **color**—the color of the text—to white, and changed the font-family to monospace. This completely changes the look of our page. Don't forget to use semi-colons to separate the properties in your CSS rule for body; this is easy to forget, and could cause all of the CSS that falls below the forgotten semi-colon to fail, which can be confusing! If you find yourself running into problems, that's the first thing to check.


Styling with Classes

Suppose you want to change the background color of every other element. How would you do that?

First we'll try using the element as a selector. Modify your code as shown:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title> My Home Page </title>
  <meta charset="utf-8">
  <style>
    body {
      background-color: white;
      font-family: Arial, Helvetica, sans-serif;
    }
    li {
      background-color: #a9a9a9;
    }
  </style>
</head>
<body >
  <h1>Welcome to My Home Page</h1>
  <p>
    My favorite live rock bands:
  </p>
  <ul>
    <li>Phish</li>
    <li>Blues Traveler</li>
    <li>Widespread Panic</li>
    <li>Pink Floyd</li>
    <li>Rolling Stones</li>
  </ul>
</body>
</html>
```

Click **Preview** . Wait a minute—this is *not* what we wanted. We've set the background color of every list item. We used the selector **li**, which selects every list item in the page!

You may recall that we can use a **class** to select specific elements with a class attribute that matches a class in the CSS. Let's use a class to select only every other list item element to have a particular background color.

To use classes, we'll need to add the class attribute to the HTML first. Modify your code again, as shown:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title> My Home Page </title>
  <meta charset="utf-8">
  <style>
    body {
      background-color: white;
      font-family: Arial, Helvetica, sans-serif;
    }
    li.grey {
      background-color: #a9a9a9;
    }
  </style>
</head>
<body>
  <h1>Welcome to My Home Page</h1>
  <p>
    My favorite live rock bands:
  </p>
  <ul>
    <li class="grey">Phish</li>
    <li>Blues Traveler</li>
    <li class="grey">Widespread Panic</li>
    <li>Pink Floyd</li>
    <li class="grey">Rolling Stones</li>
  </ul>
</body>
</html>
```

Click **Preview** .

We added **.grey** to the **li** selector, and added the class "grey" to every other list item. By doing this, we are telling our program to, "select every element in the page that has a class of grey." Compare this to what we had before, when we were using just the **li**, which prompted our program to, "select every element in the page." We've made our selector more *specific* by adding the class.


Classes give you more flexibility in selecting elements than using the element tag name alone.

You can also use classes without the element tag name in the selector. Modify your code, like this:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title> My Home Page </title>
  <meta charset="utf-8">
  <style>
    body {
      background-color: white;
      font-family: Arial, Helvetica, sans-serif;
    }
    .grey {
      background-color: #a9a9a9;
    }
  </style>
</head>
<body>
  <h1>Welcome to My Home Page</h1>
  <p>
    My favorite live rock bands:
  </p>
  <ul>
    <li class="grey">Phish</li>
    <li>Blues Traveler</li>
    <li class="grey">Widespread Panic</li>
    <li>Pink Floyd</li>
    <li class="grey">Rolling Stones</li>
  </ul>
</body>
</html>
```


You still need the "." in front of "grey" to specify that we're using a *class* as a selector, not a tag name.

Click **Preview** . The page should look exactly the same as before, because now we're indicating that we want to, "select every element with a class of grey," which just so happens to be the same set of elements we were selecting before.

But what happens if we use the grey class on another element, say our heading element? Update your HTML and add the class to the heading too, as shown:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title> My Home Page </title>
  <meta charset="utf-8">
  <style>
    body {
      background-color: white;
      font-family: Arial, Helvetica, sans-serif;
    }
    .grey {
      background-color: #a9a9a9;
    }
  </style>
</head>
<body >
  <h1 class="grey">Welcome to My Home Page</h1>
  <p>
    My favorite live rock bands:
  </p>
  <ul>
    <li class="grey">Phish</li>
    <li>Blues Traveler</li>
    <li class="grey">Widespread Panic</li>
    <li>Pink Floyd</li>
    <li class="grey">Rolling Stones</li>
  </ul>
</body>
</html>
```

Click **Preview** . The heading at the top of your page now has a grey background. You can mix and match using class selectors with element selectors to be as general or specific as you want to be when selecting which elements to style. You'll see lots more examples of how we do this throughout the course.

So, how would you add a grey background to the paragraph?

Styling with ids

Let's add a little more structure to our HTML by adding a couple of <div> elements, and update our header. We'll also use the id attribute and give each <div> element a unique id. Go ahead and remove all of the class attributes from the HTML elements while you're at it. Modify your code as shown:

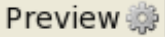
CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title> My Home Page </title>
  <meta charset="utf-8">
  <style>
    body {
      background-color: white;
      font-family: Arial, Helvetica, sans-serif;
    }
    .grey {
      background-color: #a9a9a9;
    }
  </style>
</head>
<body>
  <div id="header">
    <h1 class="grey">Ramblin' Times</h1>
    <h2>My Home Page</h2>
  </div>
  <div id="content">
    <p>
      My favorite live rock bands:
    </p>
    <ul>
      <li class="grey">Phish</li>
      <li>Blues Traveler</li>
      <li class="grey">Widespread Panic</li>
      <li>Pink Floyd</li>
      <li class="grey">Rolling Stones</li>
    </ul>
  </div>
</body>
</html>
```

Now suppose we wanted to give the header (everything in the <div> element with the id of "header") a background color of grey, and the main content of the page (in the <div> element with the id of "content") a background color of light purple. We could use the **.grey** class with the first <div> element, and create another class to use with the second. Or we could use the ids that we've given these <div> elements. Let's try that now. Modify your code again, as shown:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title> My Home Page </title>
  <meta charset="utf-8">
  <style>
    body {
      background-color: white;
      font-family: Arial, Helvetica, sans-serif;
    }
    .grey {
      background-color: #d9d9d9;
    }
    div#header {
      background-color: #d9d9d9;
      font-family: Times, serif;
    }
    div#content {
      background-color: #ccccff;
    }
  </style>
</head>
<body>
  <div id="header">
    <h1>Ramblin' Times</h1>
    <h2>My Home Page</h2>
  </div>
  <div id="content">
    <p>
      My favorite live rock bands:
    </p>
    <ul>
      <li>Phish</li>
      <li>Blues Traveler</li>
      <li>Widespread Panic</li>
      <li>Pink Floyd</li>
      <li>Rolling Stones</li>
    </ul>
  </div>
</body>
</html>
```

Click . Now your header has grey for its background color, and your main content has purple for its background color.

We used the id of each div to select that div. To use an id to select an element, we use the "#" symbol in front of the id name in the CSS. We are using a combination of the element name and the id name as the selector. We aren't required to do that, but it's good practice because it makes the CSS easier to read. Try your CSS without the element selector here—that is, try using just **#header** and **#content**—it should work just the same.

When to Use a Class and When to Use an ID

The id name you give an element using the id attribute must be *unique*. So, when you style your elements using the id, that style can only ever apply to the element with that id. However, many elements can share the same class. In fact, elements can even have more than one class.

When you're deciding whether to use a class or an id, keep these general rules in mind:

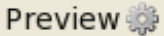
- When you are creating style that might be useful in many different situations, use a class.
- When you are creating style that is unique to a specific element in your page, use an id.

Also, keep in mind that an element can have both an id attribute *and* a class attribute, so you're not limited to one or the other—you can create CSS for both! Let's give that a try.

Update your HTML to remove the background-color from your "div#header" rule, and add the "grey" class to your header div as shown:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title> My Home Page </title>
  <meta charset="utf-8">
  <style>
    body {
      background-color: white;
      font-family: Arial, Helvetica, sans-serif;
    }
    .grey {
      background-color: #d9d9d9;
    }
    div#header {
      background-color: #d9d9d9;
      font-family: Times, serif;
    }
    div#content {
      background-color: #ccccff;
    }
  </style>
</head>
<body>
  <div class="grey" id="header">
    <h1>Ramblin' Times</h1>
    <h2>My Home Page</h2>
  </div>
  <div id="content">
    <p>
      My favorite live rock bands:
    </p>
    <ul>
      <li>Phish</li>
      <li>Blues Traveler</li>
      <li>Widespread Panic</li>
      <li>Pink Floyd</li>
      <li>Rolling Stones</li>
    </ul>
  </div>
</body>
</html>
```

Click . Your page should look exactly the same, even though now, we're using a combination of two different CSS rules to style the header div: the **.grey** rule that selects all elements with the class "grey", and the **div#header** rule that selects the <div> element with the id **header**. Both of these rules are applied to the header div, so that header gets both a grey background and a Times font.

We're specifying the background-color and font-family in the body rule, and then *overriding* those properties in rules for specific elements. We'll come back to this again later. For now, just consider the reasons that may be important.

Linking to CSS in an External File

Once your HTML and CSS starts getting larger, you'll probably want to keep the CSS in a separate file.

Another good reason to move your CSS into a separate file is so you can use that CSS with more than one HTML file. For example, if your website has many HTML files that use the same styles, you can keep the CSS in one file and link to it from many HTML files.

Let's see how to link to an external CSS file now. Copy all the CSS between the opening <style> tag and the closing </style> tag (**do not include the style tags themselves!**) and create a new file using the New File icon:




Paste in the CSS and save it in your **/htmlcss1** folder as **homepage.css**.

Now we need to link to that file from your HTML so it knows to use that style. Switch back to mybands.html, update your `<head>` element to add a link to your new CSS file, then remove the `<style>` element, like this:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title> My Home Page </title>
  <meta charset="utf-8">
  <style>
</style>
  <link rel="stylesheet" href="homepage.css">
</head>
<body >
  <div class="grey" id="header">
    <h1>Ramblin' Times</h1>
    <h2>My Home Page</h2>
  </div>
  <div id="content">
    <p>
      My favorite live rock bands:
    </p>
    <ul>
      <li>Phish</li>
      <li>Blues Traveler</li>
      <li>Widespread Panic</li>
      <li>Pink Floyd</li>
      <li>Rolling Stones</li>
    </ul>
  </div>
</body>
</html>
```

Click **Preview** . Your page should look exactly the same, even though now your style is being loaded from the homepage.css file rather than from directly within your HTML.

You can use both of these methods together; that is, you can link to an external CSS file *and* define style in your page. Typically, you'll do this to link to a file containing style that is used for multiple pages, and then define a few rules in your HTML file that apply to that file only.

Using the Style Attribute

There is one other way to define style for an HTML element; you might remember this from the beginning of the course. You can use the **style** attribute directly in an HTML element to define style. Let's add a CSS rule to the `<p>` element. Modify your code as shown:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title> My Home Page </title>
  <meta charset="utf-8">
  <link rel="stylesheet" href="homepage.css">
</head>
<body>
  <div class="grey" id="header">
    <h1>Ramblin' Times</h1>
    <h2>My Home Page</h2>
  </div>
  <div id="content">
    <p style="font-style: italic;">
      My favorite live rock bands:
    </p>
    <ul>
      <li>Phish</li>
      <li>Blues Traveler</li>
      <li>Widespread Panic</li>
      <li>Pink Floyd</li>
      <li>Rolling Stones</li>
    </ul>
  </div>
</body>
</html>
```

In general, you'll want to avoid using the **style** attribute and defining style rules directly in elements. It's easier to read and keep track of your CSS if you put all your rules in a `<style>` element at the top of your page, or put all your CSS in a separate file and link to that file using the `<link>` element. Feel free to experiment with the style attribute now.

After all of that practice, I'm pretty confident that you're comfortable with the syntax of CSS. (If you're not, keep experimenting, and play around with CSS until you are!) In the next lesson, we'll learn more about block and inline elements and how you can modify their default styles using CSS.

Copyright © 1998-2014 O'Reilly Media, Inc.



This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.
See <http://creativecommons.org/licenses/by-sa/3.0/legalcode> for more information.

Cascading Style Sheets: The Box Model

Lesson Objectives

When you complete this lesson, you will be able to:

- Use padding, margins, and border for your content.


Padding, Margins, and Border for Your Content

Every HTML element has three CSS properties that determine the spacing around it: padding, margins, and border. These properties can be set to 0 (the default) so you can't see them, but you can always change that with CSS style.

Before we look at exactly how padding, margin, and border work, let's create an example. Create a new HTML file and add this HTML:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title>My Home Page</title>
  <meta charset="utf-8">
  <link rel="stylesheet" href="cssbox.css">
</head>
<body>
  <div id="content">
    <h1>My favorite activities</h1>
    <h2><time datetime="2011-09-09">Friday, September 9, 2011</time></h2>
    <p>
      I <em>love</em> getting out into the great outdoors. Here are
      some of my <strong>absolute</strong> favorite things to do.
    </p>
    <ul>
      <li>Hiking</li>
      <li>Kayaking</li>
      <li>Cycling</li>
      <li>Beach walks</li>
    </ul>
  </div>
</body>
</html>
```

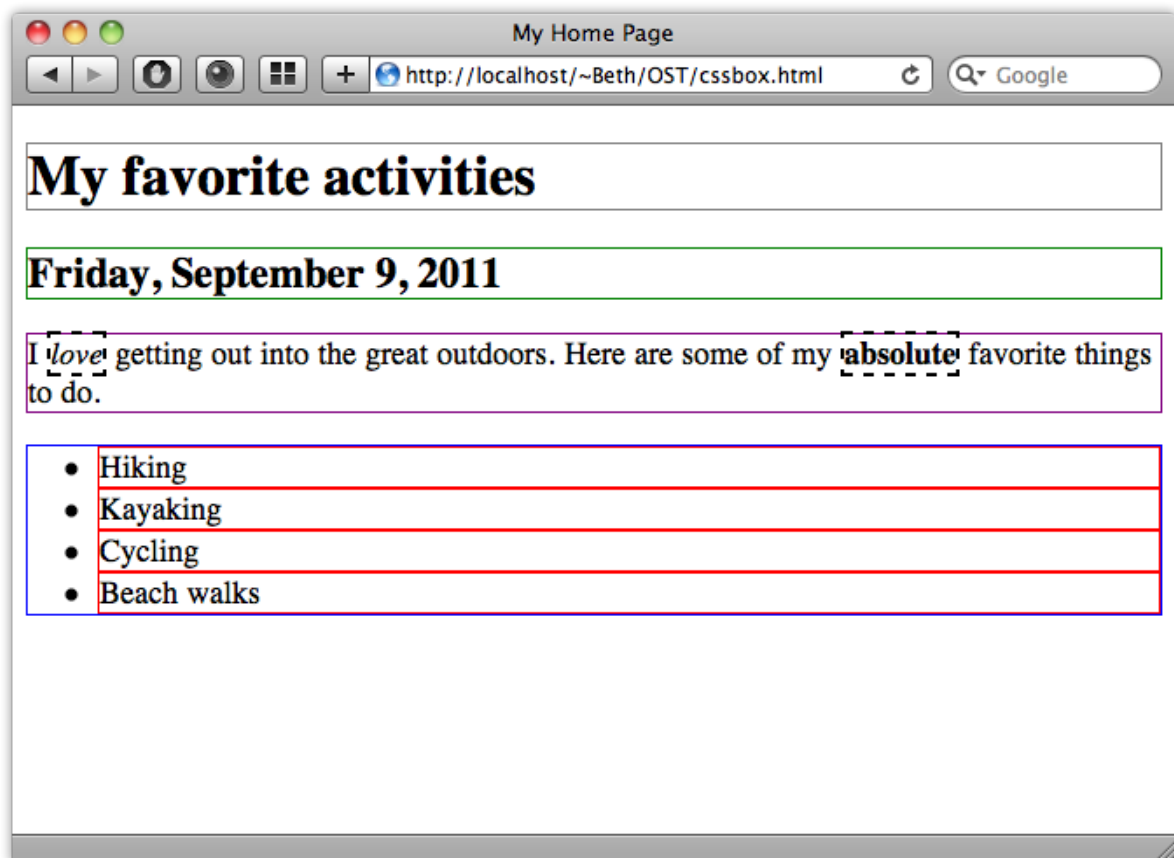
Save it in your **/htmlcss1** folder and click **Preview**  to see how it looks without any added style (although you're linking to a stylesheet file with **<link rel="stylesheet" href="cssbox.css">**, you'll see the default style for the page because the file doesn't yet exist).

Next, create a new file and select the CSS syntax, and type the code shown:

CODE TO TYPE:

```
h1 {  
  border: 1px solid grey;  
}  
  
h2 {  
  border: 1px solid green;  
}  
  
p {  
  border: 1px solid purple;  
}  
  
em, strong {  
  border: 2px dashed black;  
}  
  
ul {  
  border: 1px solid blue;  
}  
  
li {  
  border: 1px solid red;  
}
```

Save it in your `/htmlcss1` folder as `cssbox.css`. Now, preview your HTML again and you'll see boxes around each element:



The borders of the block elements, like `<h1>`, `<h2>` and `<p>` go all the way to the right side of the page, because the block elements take up the full width of the page. Conversely, the borders around the inline elements `` and `` don't go all the way to the far right side of the page; those borders hug the elements tightly because inline elements flow "in line" with the rest of the content.

Note

We combined the rules for two elements, `` and `` into one rule by separating the selectors (tag names) with a comma. If you're using the same rule for two or more elements, you can combine them like that.

Now take a look at the spacing inside and outside of the borders. The `` element (in blue) has some space between the bullets and the left side of the element where you see the border, but the `<h1>` and `<h2>` elements don't.

Okay, now you're ready to take on padding and margins. Update your CSS to add some padding to the `<h1>` element:

CODE TO TYPE:

```
h1 {
  border: 1px solid grey;
  padding: 10px;
}


h2 {
  border: 1px solid green;
}

p {
  border: 1px solid purple;
}

em, strong {
  border: 2px dashed black;
}

ul {
  border: 1px solid blue;
}

li {
  border: 1px solid red;
}
```

Click **Preview** . You should now see extra space on the left, top, and bottom of the element between the element content and the border. Try resizing your browser window so it's smaller than the width of the heading. The 10 pixels of padding is applied all around the heading; you'll see it on the right side too. Try adding padding to the `<p>` element too:

CODE TO TYPE:

```
h1 {
  border: 1px solid grey;
  padding: 10px;
}


h2 {
  border: 1px solid green;
}

p {
  border: 1px solid purple;
  padding: 10px;
}

em, strong {
  border: 2px dashed black;
}

ul {
  border: 1px solid blue;
}

li {
  border: 1px solid red;
}
```

Click **Preview** . You'll see space added to the `<p>` element.

We've used "px" a few times so far. "px" stands for pixels and is one of several different ways you can specify sizes in


CSS. Computer monitors can range from 800 pixels wide by 600 pixels high, up to thousands of pixels in height and width. A pixel is one small dot of color on the monitor; they allow you have fine-grained control over the appearance of your elements.

Before we take a closer look at how padding, margins, and border work more precisely, let's add a margin to your list item elements:

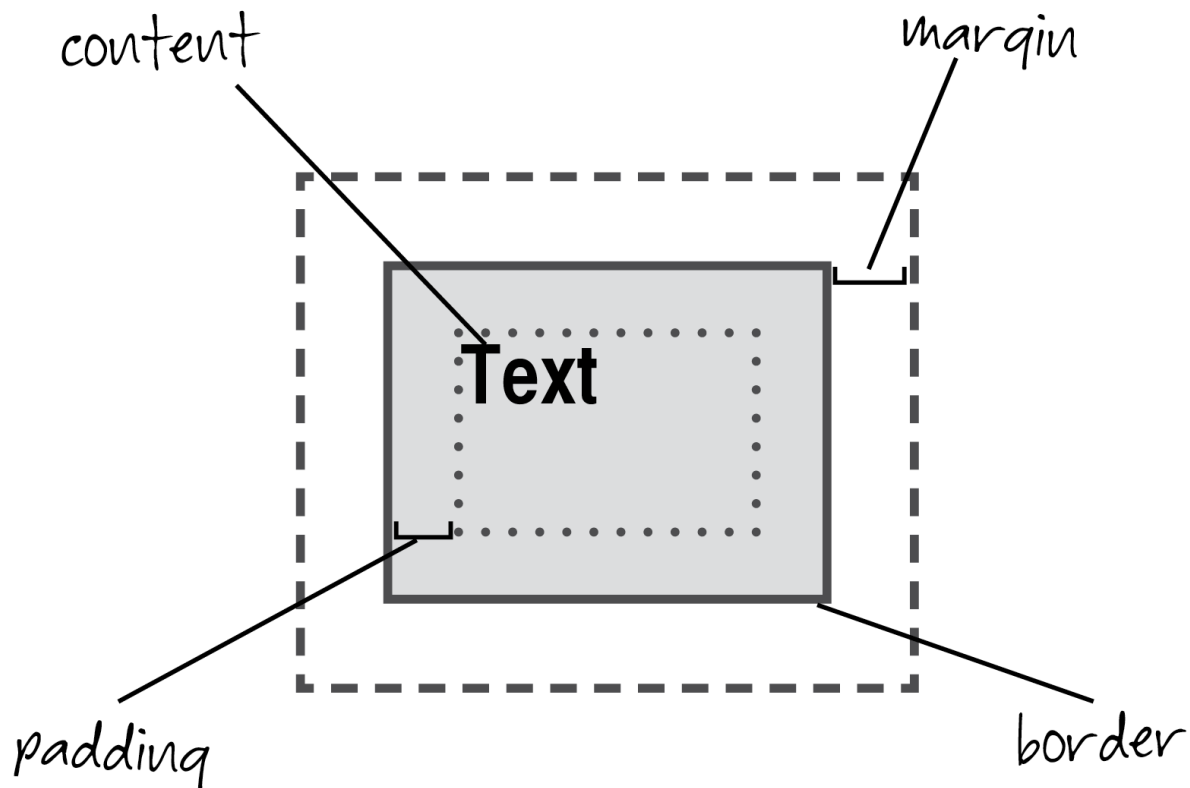
CODE TO TYPE:

```
h1 {
  border: 1px solid grey;
  padding: 10px;
}
h2 {
  border: 1px solid green;
}
p {
  border: 1px solid purple;
  padding: 10px;
}
em, strong {
  border: 2px dashed black;
}
ul {
  border: 1px solid blue;
}

li {
  border: 1px solid red;
  margin-top: 20px;
}
```

Click **Preview** . Do you see additional space above each list item? Notice that the space is *outside* the border, not inside, like it was when we added padding?

Okay, enough playing. What are padding, margins, and border really? The CSS *Box Model* determines the spacing around elements, and it is defined by padding, margins, and border, like this:



The content (the text or image in your element) sits in the middle of the box. The padding is the spacing between the content and the border of the element. For many elements, the default padding is 0 or a small amount. Then comes the border, again usually 0 by default. The margin, which is the spacing outside the border, between the elements. Typically block elements like `<h1>` and `<p>` have a small default margin to add a little space between them.


Having Fun with Padding, Margins and Border

Now that you know about padding, margins, let's have some more fun with them!

First, let's try coloring the background of an element again so you can see more clearly what's going on. Update your CSS to add a background color of light purple to your `<h1>` element (and remove the padding).


CODE TO TYPE:

```
h1 {  
  border: 1px solid grey;  
  padding: 10px;  
  background-color: #ccccff;  
}  
h2 {  
  border: 1px solid green;  
}  
p {  
  border: 1px solid purple;  
  padding: 10px;  
}  
em, strong {  
  border: 2px dashed black;  
}  
ul {  
  border: 1px solid blue;  
}  
li {  
  border: 1px solid red;  
  margin-top: 20px;  
}
```

Click **Preview** . Notice that the background color goes all the way to the border of the element. Try adding some padding back in to your `<h1>`:

CODE TO TYPE:


```
h1 {
  border: 1px solid grey;
  background-color: #ccccff;
  padding: 10px;
}
h2 {
  border: 1px solid green;
}
p {
  border: 1px solid purple;
  padding: 10px;
}
em, strong {
  border: 2px dashed black;
}
ul {
  border: 1px solid blue;
}
li {
  border: 1px solid red;
  margin-top: 20px;
}
```

Click **Preview**  again. The background color of an element includes the space the content is in, as well as the padding space.

Now try this:

CODE TO TYPE:

```
h1 {
  border: 1px solid grey;
  background-color: #ccccff;
  padding: 10px;
}
h2 {
  border-bottom: 3px solid green;
}
p {
  border: 1px solid purple;
  padding: 10px;
}
em, strong {
  border: 2px dashed black;
}
ul {
  border: 1px solid blue;
}
li {
  border: 1px solid red;
  margin-top: 20px;
}
```

Click **Preview** . The `<h2>` element now has a green line under it rather than a border all the way around it, and it is thicker (3 pixels). The **border-bottom** property sets the bottom border only, rather than the entire border, like when we used **border**. You'll probably see that this gives you a lot of control over what your borders look like—you can control each side separately, or all sides together. The properties you use to do this are: **border-top**, **border-right**, **border-bottom**, and **border-left** (and the same applies to margins and padding).

We specified border by using shortcuts for three separate properties:


OBSERVE:

```
border-width: 3px;
border-style: solid;
border-color: green;
```

You can also specify different sizes for different sides of an element property by breaking out the one number into four, like this:

CODE TO TYPE:

```
h1 {
  border: 1px solid grey;
  background-color: #ccccff;
  padding: 10px;
}
h2 {
  border-width: 1px 3px 3px 1px;
  border-style: solid;
  border-color: green;
}
p {
  border: 1px solid purple;
  padding: 10px;
}
em, strong {
  border: 2px dashed black;
}
ul {
  border: 1px solid blue;
}
li {
  border: 1px solid red;
  margin-top: 20px;
}
```

Make sure you separate each size with a space (not a comma!) and end the line with a semi-colon like you normally do. Also, always specify "px" on each number. Click **Preview** . You'll see thin green lines at the top and left, and thicker green lines at the bottom and right of your <h2> element. We specified the top, right, bottom, and left border widths separately. Think of it like a clock:



The top is at 12, and the right, bottom, and left sides are 3, 6, and 9 respectively.

There are a variety of border styles to choose from, including:

- solid
- dashed
- dotted
- groove
- ridge
- inset
- outset


Experiment with the different kinds of borders on different sides of an element. For some of them you'll notice you need a size greater than 1px to really see what it looks like. Which one do you like best? Also try experimenting with different sizes of padding and margin.

Padding, Margins, and Borders on Inline Elements

We've already added dashed borders to the inline elements in the page, `` and ``. Now, try increasing the padding and see what happens:

CODE TO TYPE:

```
h1 {
  border: 1px solid grey;
  background-color: #ccccff;
  padding: 10px;
}
h2 {
  border-width: 0px 0px 3px 0px;
  border-style: solid;
  border-color: green;
}
p {
  border: 1px solid purple;
  padding: 10px;
}
em, strong {
  border: 2px dashed black;
  padding: 50px;
}
ul {
  border: 1px solid blue;
}
li {
  border: 1px solid red;
  margin-top: 20px;
}
```

Click [Preview](#) . Think for a minute about what you're seeing....

We do indeed have 50 pixels of space between the content and the border, and as you might expect, the words next to "love" and "absolute" are shifted over, but the spacing has no effect on the spacing of the paragraph as a whole (that is, the lines still take up the same amount of space in a vertical direction) and the spacing also no effect on the overall size of the paragraph (the paragraph as a whole still takes up the same vertical space). Make the browser width smaller so you can see how the padding on these words affects how the paragraph is laid out in a variety of sizes of the paragraph.

Now try adding a margin to these elements. What happens?

You'll see that the padding and margin have an effect on the words in the same line, but not on the block in which that line is placed. The general rule is that padding, margins, and borders on inline elements have some effect on other inline elements, but not on block elements (there are, of course, exceptions to every rule, but that's a good place to start). Because inline elements are flowing inline with the rest of the content and mix with other rules in effect (such as the line height of each line in a paragraph), the browser uses a complex algorithm to determine how various CSS properties affect layout. In general, unless you're doing something really complicated, the rules and ideas that we've covered address most circumstances:

- Padding is the space between the content and the border.
- Border is the space (usually transparent, or 0px, by default) between the padding and the margin. You can style the border and give it a color and size.
- Margin is the space outside the border, and provides space between an element and other elements.
- For block elements, padding, margin, and border add space between elements in all directions.
- For inline elements, padding, margin, and border usually only add space only between elements on the same line.
- To remember the order of sizes when you specify the size of a property of each side of an element, think of a clock.

Before you go on to the exercises or the next lesson, spend some time experimenting with your CSS. Try different styles and sizes for borders; try different sizes for padding and margin; try adding background-colors to see what kinds of graphic effects you can create.

Copyright © 1998-2014 O'Reilly Media, Inc.



This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.

See <http://creativecommons.org/licenses/by-sa/3.0/legalcode> for more information.

Cascading Style Sheets: Inheritance

Lesson Objectives

When you complete this lesson, you will be able to:

- incorporate the concept of inheritance into your web page.
- combine descendant selectors with IDs and classes.

CSS Inheritance

When you set the color of the body of your HTML, that color is used as the text color for every element in your page. Every element nested in the `<body>` element *inherits* that color. The same holds true when you set the color of a `<div>` element; the elements nested inside of that `<div>` inherit the color. Let's look at how this works with an example:

CODE TO TYPE:

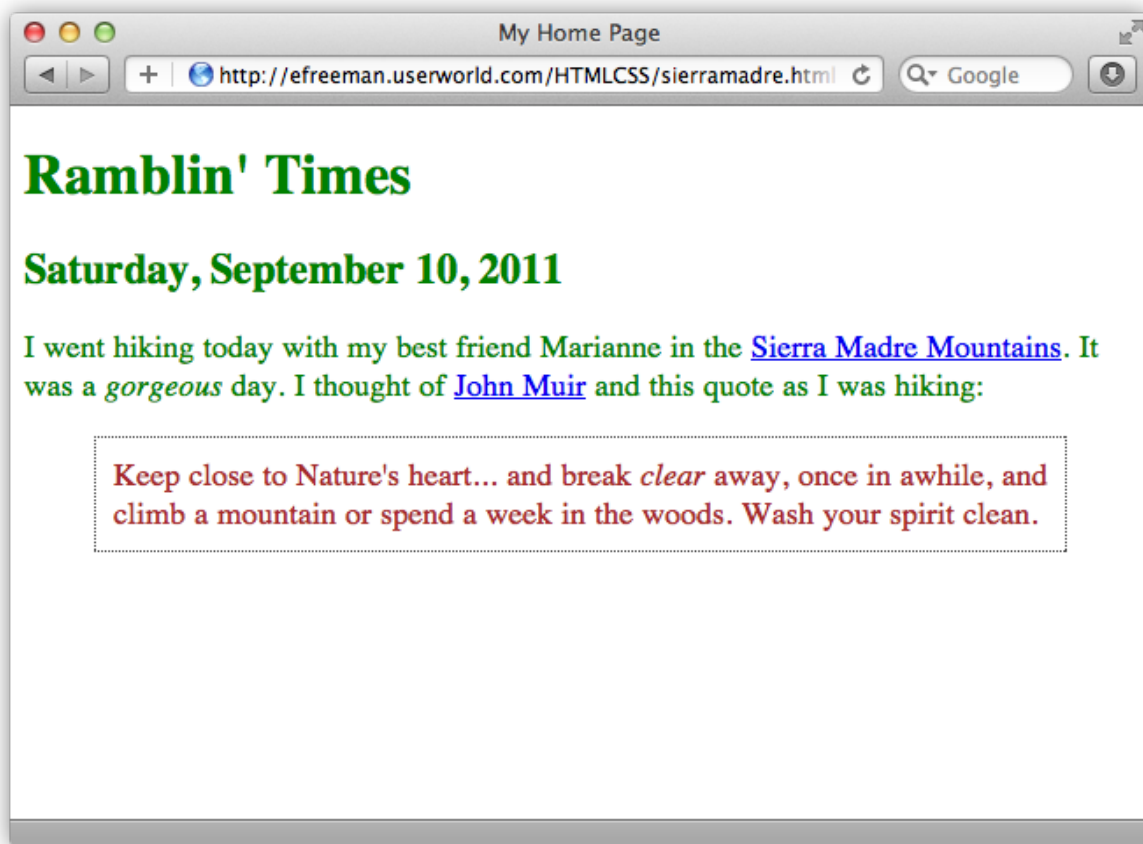
```
<!doctype html>
<html lang="en">
<head>
  <title>My Home Page</title>
  <meta charset="utf-8">
</head>
<body>
  <div id="content">
    <h1>Ramblin' Times</h1>
    <div class="article">
      <h2>
        <time datetime="2011-09-10">Saturday, September 10, 2011</time>
      </h2>
      <p>
        I went hiking today with my best friend Marianne in the
        <a href="http://en.wikipedia.org/wiki/Sierra_Madre_Mountains_(California)"
        >Sierra Madre Mountains</a>. It was a <em>gorgeous</em>
        day. I thought of
        <a href="http://en.wikipedia.org/wiki/John_Muir">John Muir</a>
        and this quote as I was hiking:
      </p>
      <blockquote>
        Keep close to Nature's heart... and break <em>clear</em>
        away, once in awhile, and climb a mountain or spend a week in the woods.
        Wash your spirit clean.
      </blockquote>
    </div>
  </div>
</body>
</html>
```

Save it in your `/htmlcss1` folder as `sierramadre.html`. Now add some style, either in your HTML file in a `<style>` element in the `<head>` element, or in a linked external CSS style sheet like we used in the previous lesson. Either way is fine, but to save space here, we'll show it as if it is in a separate file.

CODE TO TYPE:

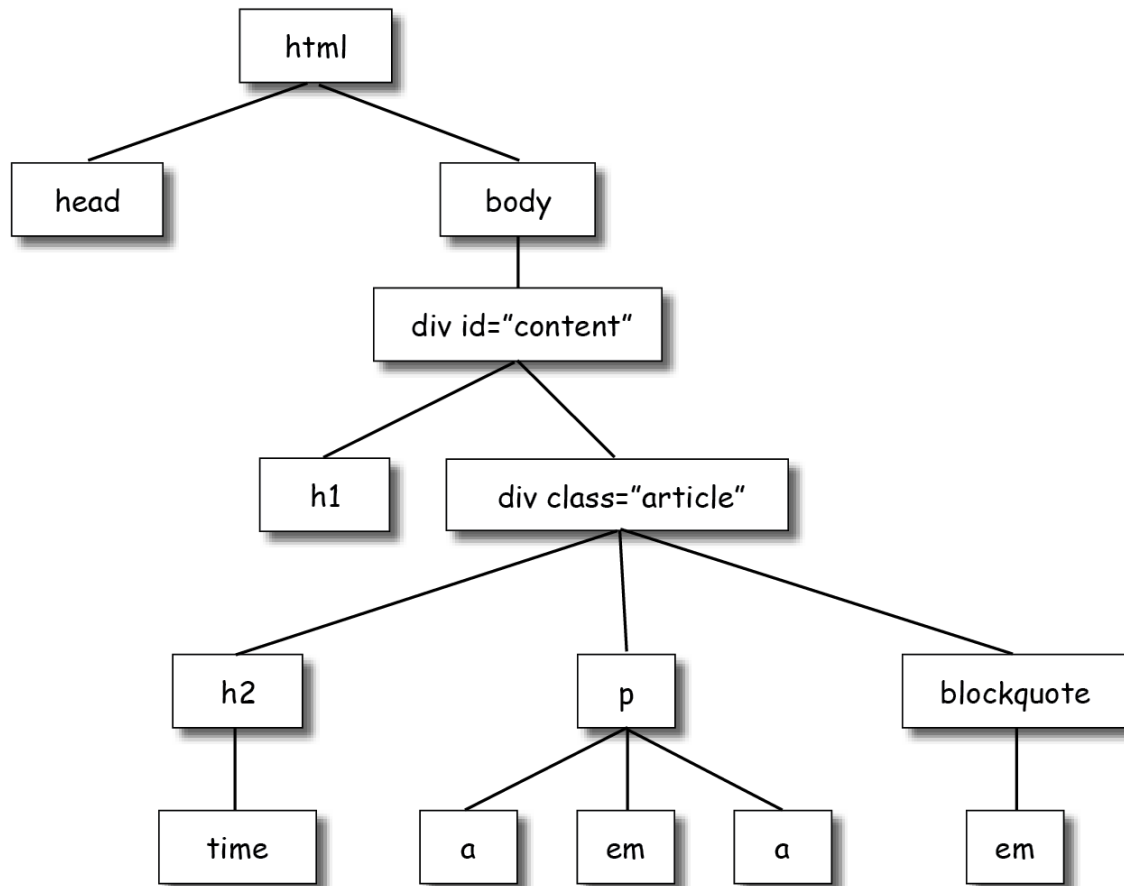
```
body {
  color: green;
}
blockquote {
  color: brown;
  padding: 10px;
  border: 1px dotted #373737;
}
```

Preview your HTML. The text should be green. All the elements nested inside the **<body>** element have a green text color (including the `<h1>`, `<h2>`, and `<p>` elements), except for the text in the `<blockquote>` element (and the `<a>` elements used for the links, which are blue).

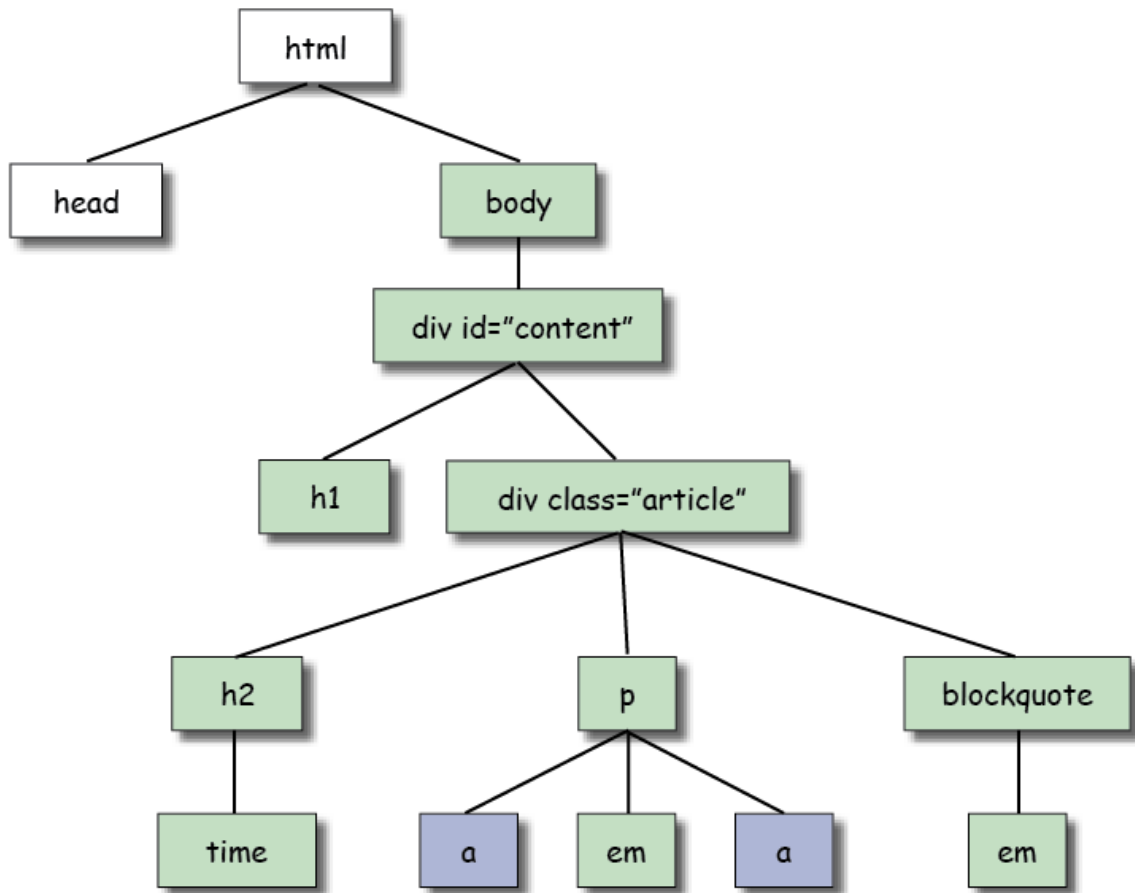


Note The `<blockquote>` element is used to mark up a long quotation.

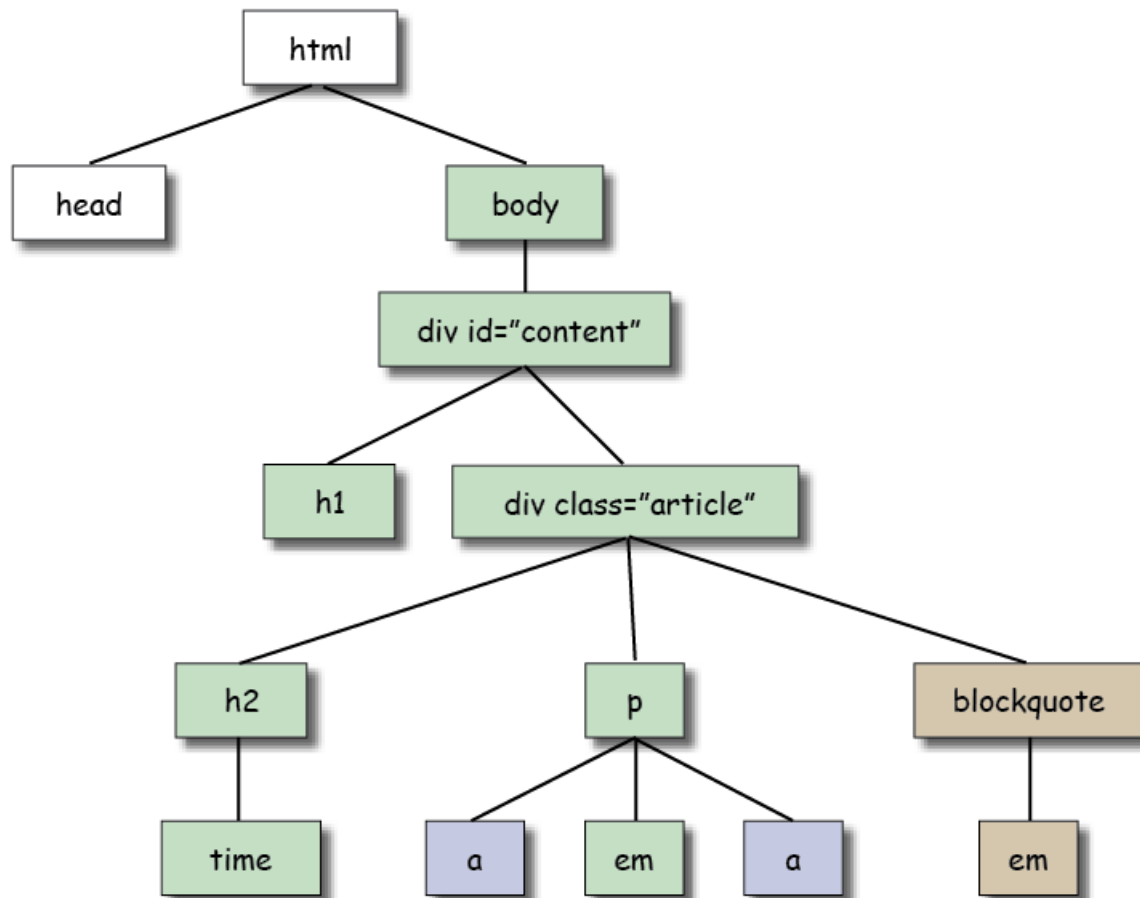
Think of the structure of an HTML document like a tree:



When we set the color of the `<body>` element, all of the elements on the `<body>` "branch" (the *children* of `<body>`) **inherit** that color (except for `<a>` elements, which the browser sets to blue so you can easily identify them as links).



By specifying a color specifically for the `<blockquote>` element, we *override* the color the `<blockquote>` inherited from its *parent*, `<body>`. (The child of `blockquote`, `em`, inherits `blockquote`'s color, too.)



Because nested elements inherit some properties from their parents, it's good practice to set property values that apply to an entire document or section of a document, like the font, on a parent element, rather than setting it in all the children elements. That way if you decide to change the value of a property, you only need to make that change in one place rather than several. It's common practice to set inherited properties like color and font on the `<body>` element, and then override them on children elements only when you specifically want a different font.

Note

Only certain properties are inherited. For details, read over the [CSS specification](#). In general, you'll be able to recognize whether a property will be inherited or not. For example, font is an inherited property, but border style is not. That means, if you set the font for a `<div>` element, all elements nested inside that `<div>` will also have that font. However, if you set a border style for that `<div>`, none of the nested elements will inherit that border style.

Experiment with some properties to see which ones are inherited or not. Here are a few you can try:

- Background color for the page and for elements nested in the page. This one is tricky because the default background-color for elements is **transparent**, so if you set the background-color on the body, it will *look* like all the elements have inherited it, but they haven't; it's just that the color is shining through their transparent backgrounds.
- Margins and padding on different elements (like the `<body>`, the `<div id="content">` element, and so on).
- Text alignment on different elements. For example, if you set this property to center in a rule for the `<body>` element, do the elements nested inside the `<body>` element also have center-aligned text? While you're trying this new property, also try the value "right."
- Font weight on different elements. Find out what happens if you add this property to your rule for `<body>`, or what happens if you put it in a rule for the `<p>` element instead.

CSS Descendant Selectors

Now you understand how your HTML document is like a family tree, with parent elements and child elements nested inside their parent elements. We can use this parent-child relationship of elements in our CSS to create *descendant selectors*. To see how descendant selectors work, let's make a few changes to the CSS in our example.

Let's say you want the text inside the `<blockquote>` element to be italic. Modify your code as shown:

CODE TO TYPE:

```
body {
    color: green;
}
blockquote {
    color: brown;
    padding: 10px;
    border: 1px dotted #373737;
    font-style: italic;
}
```

Preview your HTML file. The John Muir quote appears in italic font-style, but the word "clear" that was emphasized in italic style no longer appears to be emphasized because the entire quote is italicized, so that word doesn't stand out.

So, let's say you want to change the text in *all* the `` elements in your page to be displayed as bold, non-italicized text; that is, the **font-style** as **normal** and the **font-weight** as **bold**. You can add a new rule to your CSS that applies only to `` elements:

CODE TO TYPE:

```
body {
    background-color: #d6f5d6;
}
blockquote {
    background-color: #d9d9d9;
    padding: 10px;
    border: 1px dotted #373737;
    font-style: italic;
}
em {
    font-style: normal;
    font-weight: bold;
}
```

Preview your HTML file. Notice that the font-style and font-weight of all the `` elements in your page are now bold and non-italicized. (Also, notice that the font-style (italics or normal) and the font-weight (bold or normal) are two different CSS properties!)



Hmmm...that's interesting, but you'd still really like the content of the `` element in the blockquote to be in italics because you want *all* the text in the blockquote to be in italics, even if it's emphasized.

So think for a moment: what options do we have to override the general rule for **em** that you just added, but only for `` elements nested inside the `<blockquote>` element?

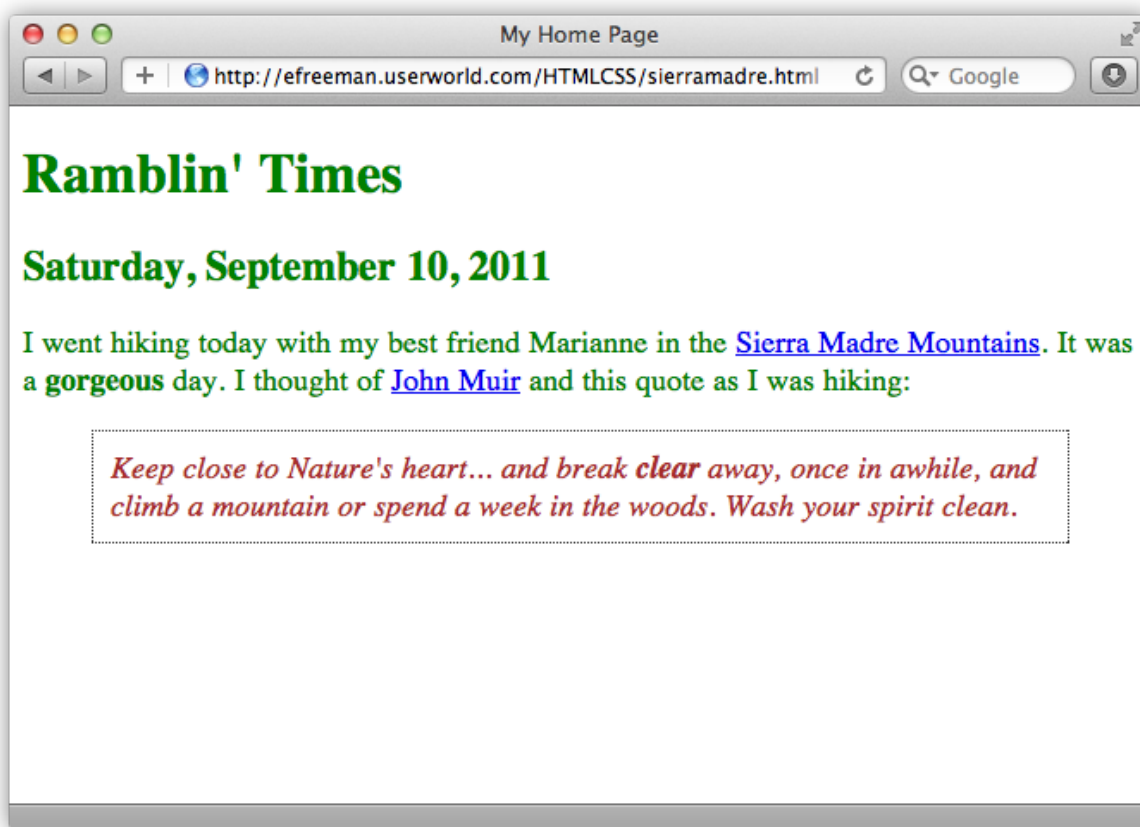
You might think that you could add an id specifically to the `` element in the blockquote, which would certainly work, but that would get a bit tedious if you have lots of blockquotes with `` elements nested inside them; you'd end up having to give each `` an id, and have a separate rule for every one of them within your CSS! so that's definitely not the best way to go.

Okay then, so instead you might consider adding a class specifically to the `` element in the blockquote. This is an option, and a better one than using an id because you only have to define the rule for the class once in your CSS and you can use it for all `` elements nested in `<blockquote>` elements. But, there's an even better way we can do this. Modify your code as shown:

CODE TO TYPE:

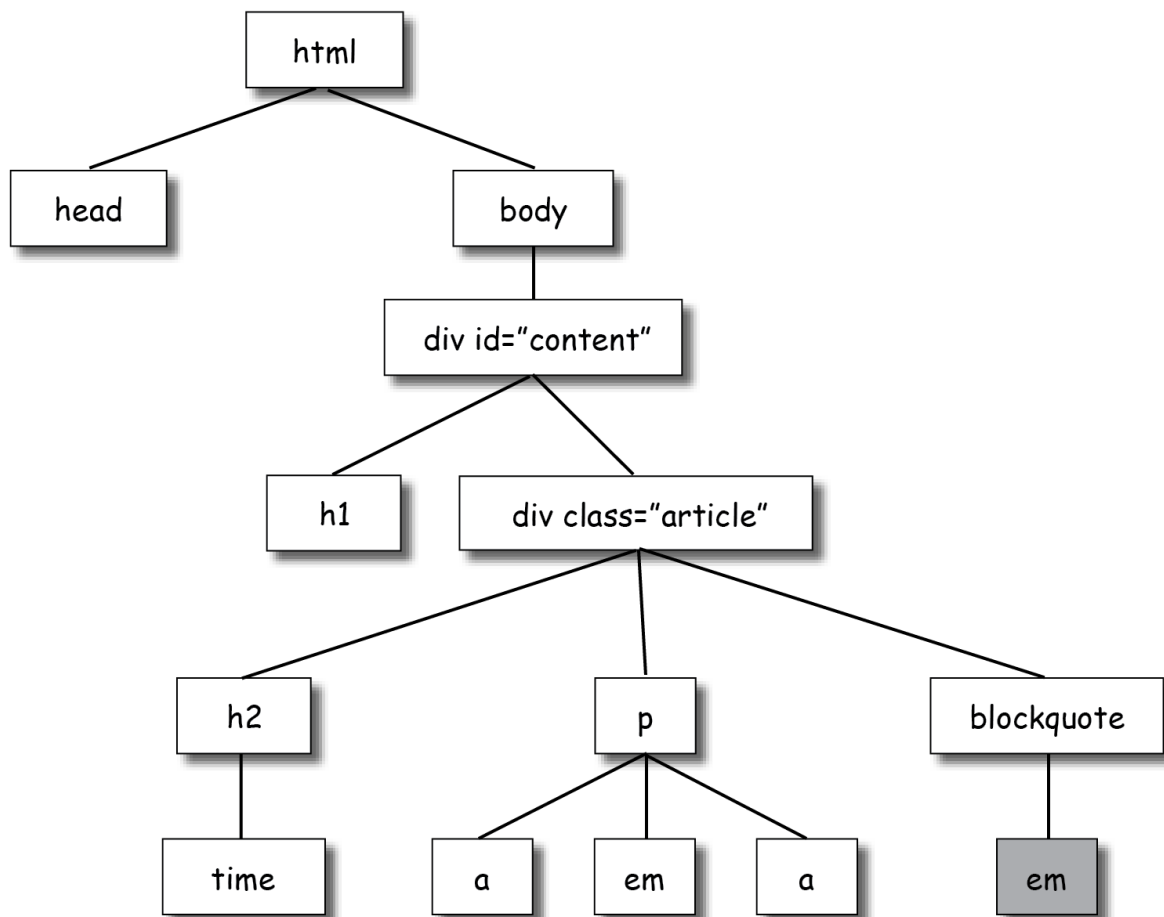
```
body {
    color: green;
}
blockquote {
    color: brown;
    padding: 10px;
    border: 1px dotted #373737;
    font-style: italic;
}
em {
    font-style: normal;
    font-weight: bold;
}
blockquote em {
    font-style: italic;
}
```

Type this in carefully. Notice there is a space between "blockquote" and "em" in the selector of this rule. Once you've updated your CSS, preview the HTML. You'll see that the content of the element that is nested within the <blockquote> element is now bold weight and in italics, but the other element is still bold weight and normal style.



With these two rules we are saying: "For all elements in the page, make the font-style of the text normal, and the font-weight of the text bold. But if an element is nested inside a <blockquote> element, change the font style to **italic**."

This is a bit tricky, so be careful. Because font-weight and font-style are inherited, *three* rules are affecting the in the <blockquote>! First, the inherits the font-style italic from the rule for blockquote. Then we apply the font-style normal and font-weight bold to all elements in the page using the general rule for em. And finally, we *override* this general rule with a more *specific* rule for nested within <blockquote> elements, using the descendant selector **blockquote em**, to set the font-style to italic. Because we are not overriding the font-weight property, however, elements nested within <blockquote> elements are displayed using font-weight bold (from the general rule for em) and font-style italic (from the more specific rule for elements nested within <blockquote> elements).



Try adding another `<blockquote>` element with an `` element nested within it. You should see that the rule you wrote applies to it as well.

Now, try using a descendant selector to select the `<a>` elements in the paragraph. Modify your code as shown:

CODE TO TYPE:

```
body {
  color: green;
}
blockquote {
  color: brown;
  padding: 10px;
  border: 1px dotted #373737;
  font-style: italic;
}
em {
  font-style: normal;
  font-weight: bold;
}
blockquote em {
  font-style: italic;
}
p a {
  background-color: yellow;
}
```

Save your CSS and preview the HTML page. With this rule, you're selecting all the `<a>` elements that are nested within a `<p>` element, and setting the background color of those elements to yellow. Try adding an `<a>` element somewhere else in the page, not nested in the `<p>` element (for example, in the `blockquote`). Is its background yellow? Make sure you understand why it's not!

Now try changing the text color of `<a>` using the `color` property. You can easily change both the background and text

colors of the links in your page using these two properties.

Combining Descendant Selectors with IDs and Classes

You can also use descendant selectors with selectors using ids and classes. Let's say you want to be even more specific about your rule for `` elements nested within `<blockquote>` elements; in fact, you want to select only those `` elements that are in `<blockquote>` elements that are in `<div>` elements with a class of "article." Modify your code as shown:

CODE TO TYPE:

```
body {
    color: green;
}
blockquote {
    color: brown;
    padding: 10px;
    border: 1px dotted #373737;
    font-style: italic;
}
em {
    font-style: normal;
    font-weight: bold;
}
div.article blockquote em {
    font-style: italic;
}
p a {
    background-color: yellow;
}
```

Save your CSS and preview the html page. The word "clear" in the blockquote should still be italicized and bold. Now, what happens if you replace "article" in **div.article** in the rule above with something else? Try changing "article" to "other," so you have **div.other** instead. What happens?

What if you wanted to make your selector *even more specific* and select only `` elements that are nested in `<blockquote>` elements, that are nested in `<div>` elements with a class of "article," that are nested in the `<div>` element with an id of "content"? Can you see how to do that?

There are many other kinds of selectors too, which we'll cover in a later course, but for now you have enough information at your disposal to make lots of interesting selections. Have fun, play around a bit with different selectors, inheritance, and some of the new properties you've learned about.

Copyright © 1998-2014 O'Reilly Media, Inc.



This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.
See <http://creativecommons.org/licenses/by-sa/3.0/legalcode> for more information.

Cascading Style Sheets: Layout and Positioning

Lesson Objectives

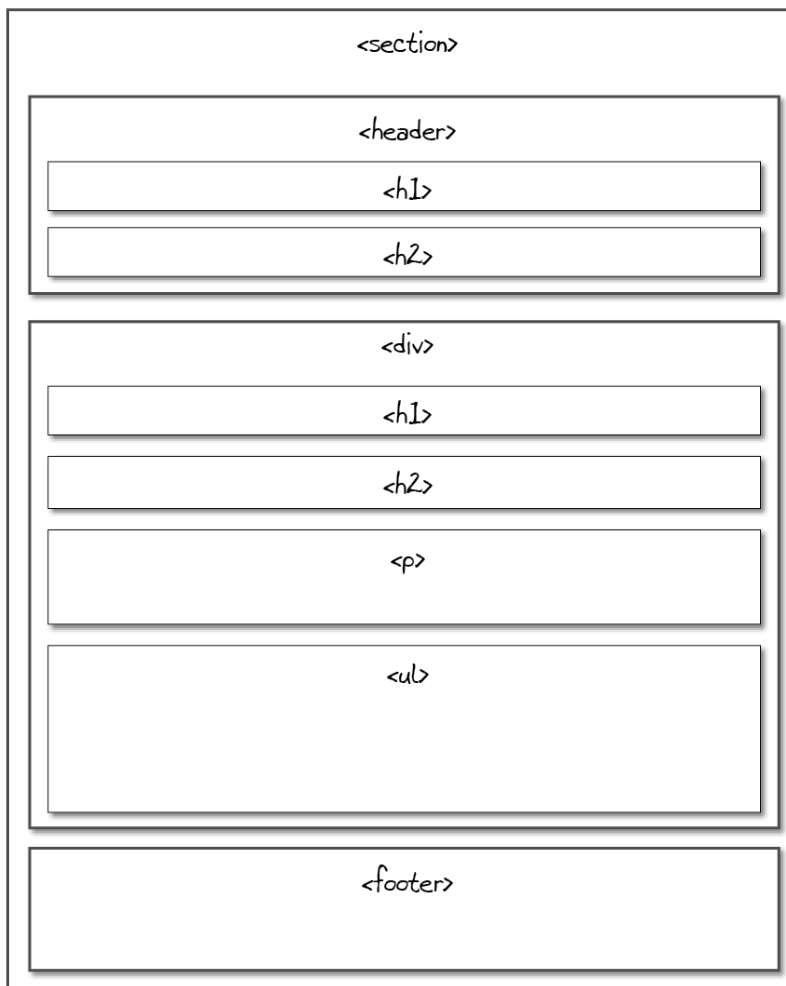
When you complete this lesson, you will be able to:

- use CSS to change the default layout of your page.
- use CSS to position your elements.

We've used CSS to set various properties such as color and font, as well as changed the box properties of elements with margins, padding, and border. You can also use CSS to change the default layout of your page.

The browser lays out a page (by default) by flowing elements on to the page, top down, and left to right. Each block element (like `<div>`, `<p>`, or ``) gets its own block so these elements end up stacked on top of one another, taking up the full width of the browser. Each inline element (like text, ``, ``, and ``) is placed into the page on the same line as the previous inline element up to the edge of the browser, and then a new line of inline elements is started.

Block Layout:



Inline Layout:

`<p>`

I `love` listening to music. Here are
some of my `absolute` favorites.

By default, the browser starts flowing elements into the page at the upper-left hand corner.

In one common layout on the web, the page content is placed in the center of the browser window. As you resize the browser window, the content stays in the middle. Let's build that layout now.

Note

Before you start the HTML, get the links to the two images you'll need, [coffee1.jpg](#) and [coffee2.jpg](#). You can link to them using their URLs, or if you want to have your own copies of the images, click the links, save the images to your computer, and upload them to your **/htmlcss1/images** folder. In the example below we assume the images are in an /images folder, but you can replace that relative URL with an absolute URL to the images if you'd rather do that.

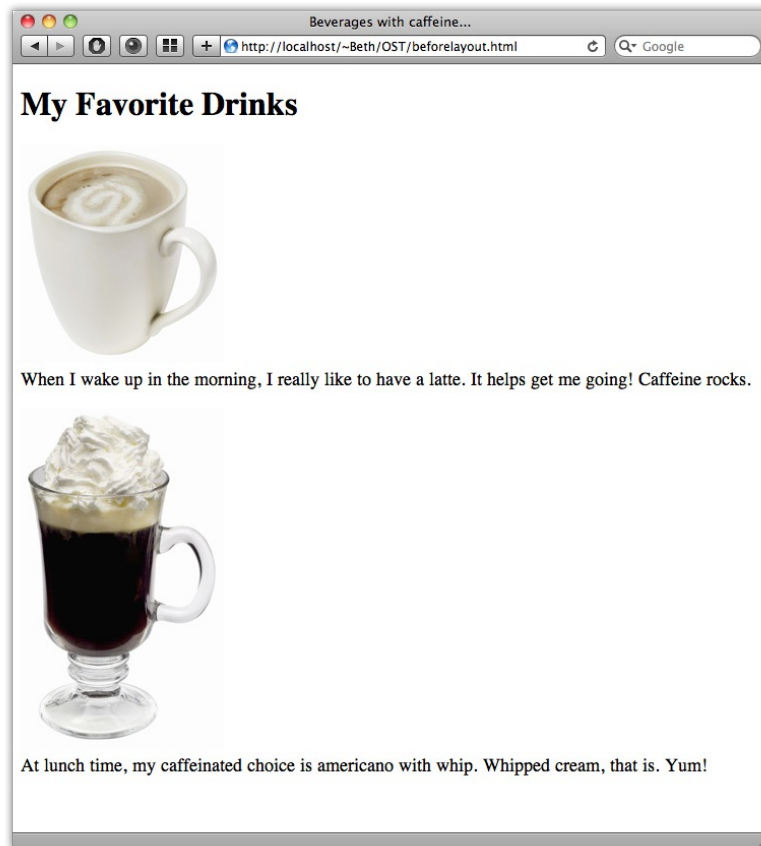
Create a new HTML file as shown:

CODE TO TYPE:

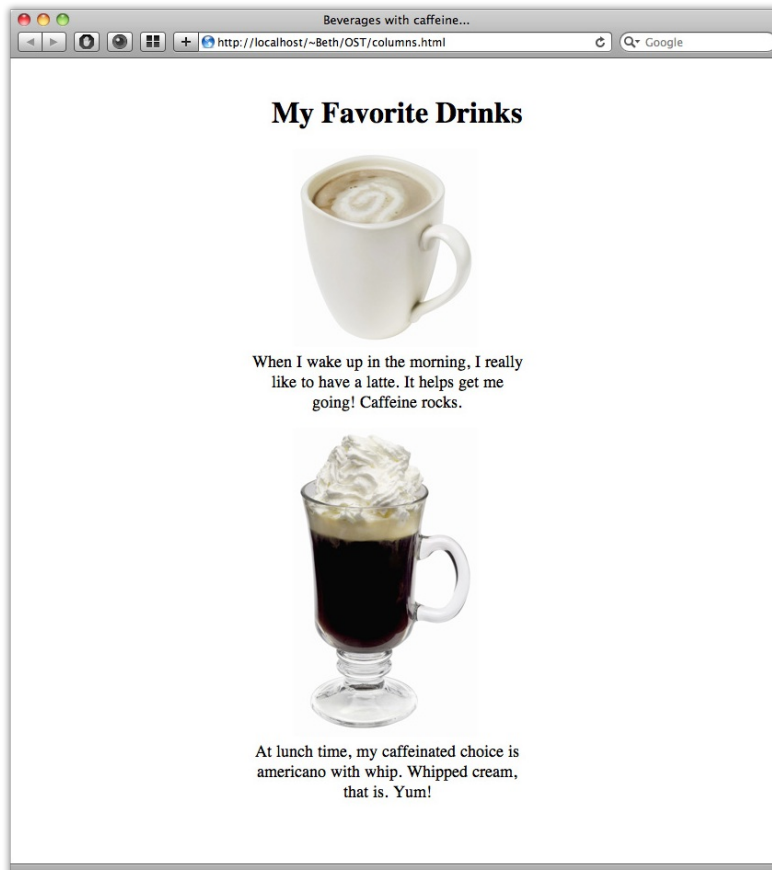
```
<!doctype html>
<html lang="en">
<head>
  <title>My Home Page</title>
  <meta charset="utf-8">
</head>
<body>
  <div id="content">
    <div id="header">
      <h1>My Favorite Drinks</h1>
    </div>

    <div id="drinks">
      <p>
        
        <br>
        When I wake up in the morning, I really like to have
        a latte. It helps get me going! Caffeine rocks.
      </p>
      <p>
        
        <br>
        At lunch time, my caffeinated choice is americano with whip.
        Whipped cream, that is. Yum!
      </p>
    </div>
  </div>
</body>
</html>
```

Save it in your **/htmlcss1** folder as **layout.html**. Before we add any style, preview the page. You'll see this:



Now we'll use CSS to make the page look like this instead:



Now let's add some style. You can either add this to your HTML file in a `<style>` element in the `<head>` element, or you can link to an external CSS style sheet like we did earlier. Either way is fine. Again, we'll present it as though it's in a

separate CSS file. Modify your code as shown:

CODE TO TYPE:

```
p {
  text-align: center;
  width: 400px;
}
```

First, we style the `<p>` element so that the text is aligned to be centered and the **width** of the element is 400px.

The **width** property sets the width of *block* elements to a certain size. Even if you decrease the width of the element relative to the current size of the browser window, the elements that follow that block element will still be located on the next line.

The **text-align** property affects the `` element as well as the text because the `` element is an inline element; text-align (despite its name) actually affects the alignment of any inline element, as well as text that is nested in the element to which it is applied. (This comes in pretty handy sometimes.)

Preview your HTML page. It's looking more like we want it to, but all of the content is still over on the left side of the page. Let's move it to where we want it, in the center of the page. Modify your code as shown:

CODE TO TYPE:

```
p {
  text-align: center;
  width: 400px;
}
div#content {
  width: 400px;
  margin: auto;
  padding-top: 20px;
}
```

Preview the HTML page. Everything has magically moved to the center of the browser! Try resizing your browser window. You'll see that whether you make your browser window narrow or wide, the content stays in the center. Let's trace how this works.

We style the `<div>` element with the id "content." To get a centered layout like this, we need a `<div>` element that contains all of our content, because we want to center all of our content by setting a margin between that `<div>` and the `<body>`. Remember that a margin is the space between elements, so by setting a margin on this `<div>` we're adding space between the edge of the body (which contains the `<div>`) and the edge of the `<div>`.

We could have set the left margin to a fixed size, like 200px, and if the browser is open to 800px wide, then the content would be located exactly in the center (200px margin + 400px wide content + 200px space on the right places your content in the middle). We don't do that though because we have no idea how wide the user's browser is going to be.

Using **margin: auto**; we instruct the browser to set the margin itself automatically. And it turns out that almost all major browsers set it automatically so that the left and right margins are equal in size. This is a useful trick you can use to get content into the middle of your page.

Note You might see different results with some browsers; most will center the content page as described.

We set the width of the `<div>` to the same as that of the `<p>` so that the content will be centered exactly on the page (if we made the `<div>` wider than the `<p>` we'd have to add some margin to the `<p>` to make sure it remains centered).

We also added some padding to the top to give the content a little breathing room at the top of the page.

Did you notice that the content of the heading is not centered? You can fix that. Just add the text-align property to a rule for the `<h1>` element too. Modify your code as shown:

CODE TO TYPE:

```
p {
  text-align: center;
  width: 400px;
}
div#content {
  width: 400px;
  margin: auto;
  padding-top: 20px;
}
h1 {
  text-align: center;
}
```

Preview your HTML page. Ta-da! We have created the page layout we wanted, using CSS.

Now, knowing what you know about CSS inheritance, could we have achieved the same effect if we'd set **text-align: center** in the rule for the "content" <div> element instead? Try it! Add **text-align: center** to the **div#content** rule and remove it from the h1 and p rules as shown:

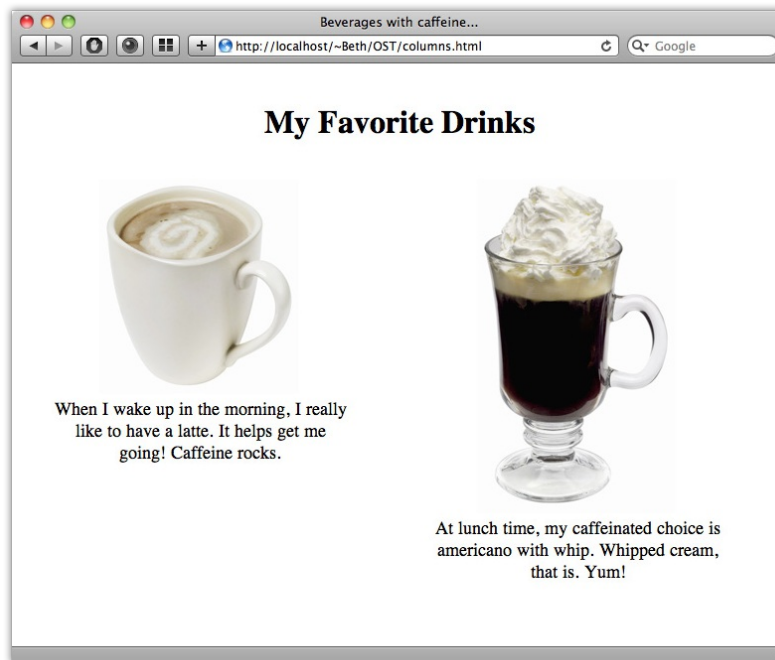
CODE TO TYPE:

```
p {
  text-align: center;
  width: 400px;
}
div#content {
  width: 400px;
  margin: auto;
  padding-top: 20px;
  text-align: center;
}
h1 {
  text-align: center;
}
```

Does it work?

CSS Positioning

Let's make things a tad more complicated and create a layout where the two drinks are side-by-side instead of one above the other, like this:



This time around you'll use **CSS Positioning**. Unlike your last layout which was essentially a trick that used the margin to add space in the right places, with CSS Positioning, you can actually remove elements from the normal flow that the browser uses to place them on a page, and position them precisely yourself.

Let's review the CSS we have right now, and make one quick change before we add the positioning. Because we're going to place the paragraphs next to each other instead of on top of each other, the first change we'll make is to double the width of the "content" <div>. Modify your code as shown:

CODE TO TYPE:

```
p {  
    width: 400px;  
}  
div#content {  
    width: 800px;  
    margin: auto;  
    padding-top: 20px;  
    text-align: center;  
}
```

Preview the HTML page. You'll see that the paragraphs have scooched over to the left and aren't under the title any more. That's because the <div> they are in is 800px wide instead of 400px wide, and, by default, each paragraph is positioned (flowed by the browser) into the left side of the <div> they're contained in (remember, the browser flows elements onto the page top-down and left to right).

Now, we'll take the second paragraph and move it up next to the first paragraph. To do that, we have to take it out of the normal flow the browser uses and tell the browser we're going to position that paragraph ourselves.

To style that second paragraph, we need a way to identify it uniquely. We identify an element uniquely by giving it an ID. Let's give both paragraphs IDs; we'll call them "col1" and "col2." Modify your code as shown:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title>My Home Page</title>
  <meta charset="utf-8">
  <link rel="stylesheet" href="csslayout.css">
</head>
<body>
  <div id="content">
    <div id="header">
      <h1>My Favorite Drinks</h1>
    </div>

    <div id="drinks">
      <p id="col1">
        
        <br>
        When I wake up in the morning, I really like to have
        a latte. It helps get me going! Caffeine rocks.
      </p>
      <p id="col2">
        
        <br>
        At lunch time, my caffeinated choice is americano with whip.
        Whipped cream, that is. Yum!
      </p>
    </div>
  </div>
</body>
</html>
```

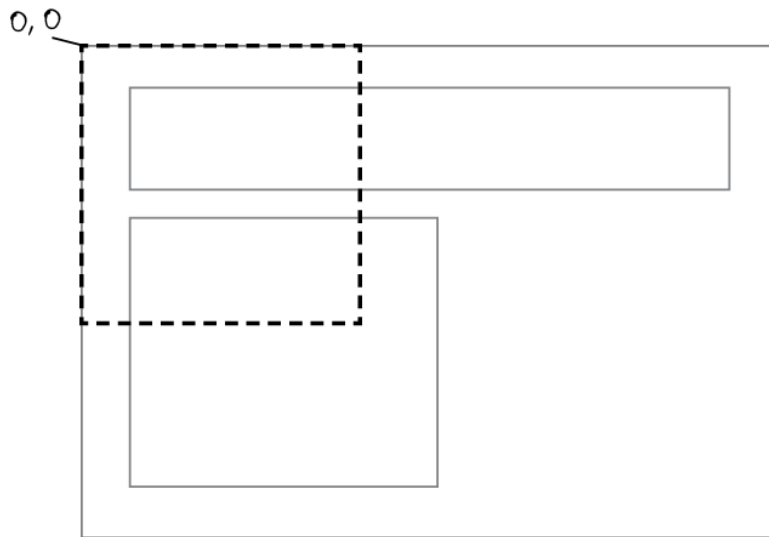
Now that we have a way to identify the paragraph we want to move ("col2"), let's update the CSS to move it over. Modify your code once more as shown:

CODE TO TYPE:

```
div#content {
  width: 800px;
  margin: auto;
  padding-top: 20px;
  text-align: center;
}
p {
  width: 400px;
}
p#col2 {
  position: absolute;
  top: 0px;
  left: 0px;
}
```

position: absolute means that the **top** and **left** coordinates of an element are defined relative to the upper-left corner of the web page, rather than according to the natural flow of elements onto the page (which the browser usually handles for you). By adding **position: absolute** to the rule for p#col2 in the CSS above, we're telling the browser that we're going to position the <p> element, so the browser should not flow it into the page.

By default, the position of an absolutely positioned element is measured relative to the upper left-hand corner of the web page. Think of this as 0, 0 in a coordinate system. We've specified a top position of 0px and a left position of 0px, so the "col2" <p> will appear at the upper left-hand corner of the browser window.

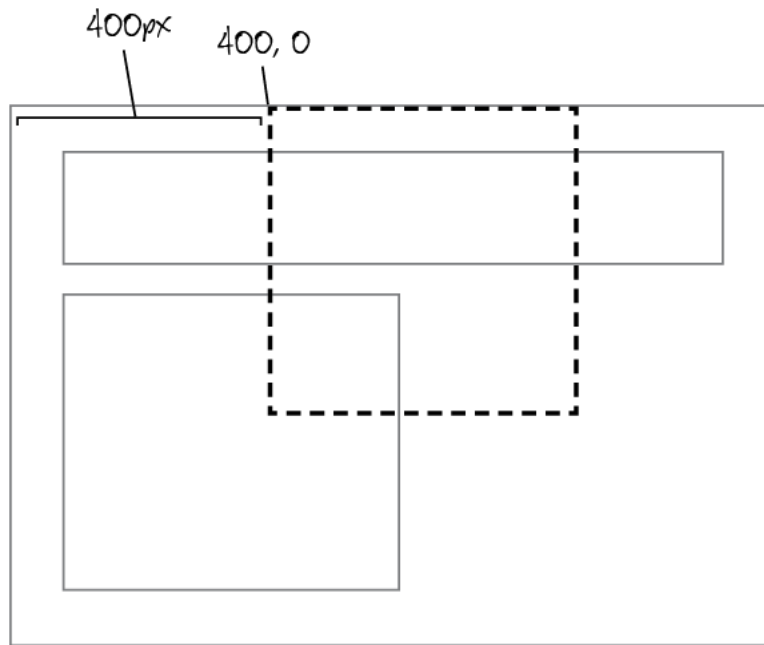


We want our paragraph to be located to the right of our other paragraph. Each paragraph is 400px wide. To move an absolutely positioned element over 400px, we give it a left position of 400px. Modify your code as shown:

CODE TO TYPE:

```
div#content {  
  width: 800px;  
  margin: auto;  
  padding-top: 20px;  
  text-align: center;  
}  
p {  
  width: 400px;  
}  
p#col2 {  
  position: absolute;  
  top: 0px;  
  left: 400px;  
}
```

This will move the paragraph over 400px from the left, but the position is still all wrong because the paragraph is absolutely positioned *relative to the browser window* (actually, it's positioned relative to the <html> element, but for our purposes, that's essentially the same thing because the <html> element always starts in the upper left-hand corner of the browser window):



We really want our paragraph to be positioned relative to the "drinks" <div>. This <div> contains only the two <p> elements. (We put the "drinks" <div> there intentionally, knowing we'd need it to position our <p> elements properly.)

So how do we tell the browser to position our two paragraph elements relative to the "drinks" <div>, with the "col2" <p> element 400px over from the left?

Here's the trick: if you have an absolutely positioned element that is nested inside another positioned element, then the absolutely positioned element will treat the corner of its parent element as the starting point instead of the upper-left corner of the web page.

That means if we position the "drinks" <div>, then we can position the two <p> elements inside it based on the upper-left corner of the "drinks" <div> instead of the browser window.

But we don't want to position the "drinks" <div> absolutely. Why? Because we don't know where it should go. We don't know exactly how high the heading will be, nor exactly (in pixels) how far from the left or top the "drinks" <div> should go to keep it in the center of the browser window. In fact, we want the "drinks" <div> to go exactly where it would normally go if we just let the browser flow it onto the page.

The solution is to use *relative position*. **position: relative** means that the element will be positioned relative to where it would normally be located in the flow of the HTML. Then if you specify a top and left position for that element, it will be moved *relative to its normal position in the flow*. Let's position the "drinks" <div>. Modify your code again, as shown:

CODE TO TYPE:

```
div#content {
  width: 800px;
  margin: auto;
  padding-top: 20px;
  text-align: center;
}
div#drinks {
  position: relative;
}
p {
  width: 400px;
}
p#col2 {
  position: absolute;
  top: 0px;
  left: 400px;
}
```

Now, if you wanted to, you could move the "drinks" <div> around by including a top position and a left position. But for this example, we want the "drinks" <div> to stay exactly where it is—where it is placed into the normal flow of the page

by the browser. We're actually adding **position: relative** to the "drinks" <div> so that we can position each of the paragraphs nested inside it with respect to the upper left-hand corner of the "drinks" <div>.

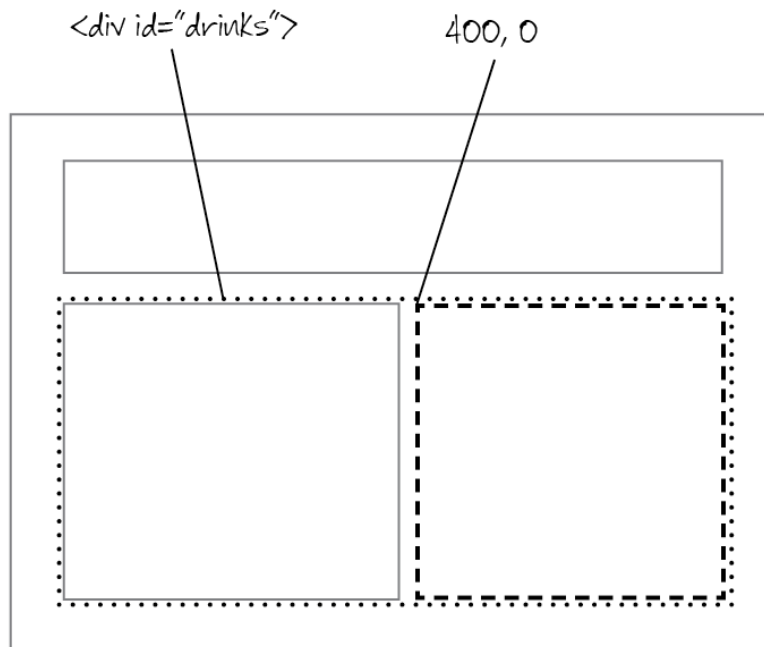
Let's finish up the CSS with the almost final version. Modify your code again as shown:

CODE TO TYPE:

```
div#content {
  width: 800px;
  margin: auto;
  padding-top: 20px;
  text-align: center;
}
div#drinks {
  position: relative;
}
p {
  width: 400px;
  position: absolute;
}
p#col1 {
  top: 0px;
  left: 0px;
}
p#col2 {
  position: absolute;
  top: 0px;
  left: 400px;
}
```

Preview your HTML page. You'll see the two drink paragraphs next to each other, and aligned underneath the heading of the page. Let's review the steps we took:

- We positioned the "drinks" <div> **relative** to its normal position in the flow. We did this so we could position the two paragraphs with respect to the upper left-hand corner of the "drinks" <div> in order to be sure they're positioned in the correctly on the page. (The "drinks" <div> is shown with a dotted line in the diagram below.)
- Then we specified that both paragraphs should be positioned **absolutely**. Because their parent element, "drinks" <div>, is positioned, they will be positioned with respect to it.
- We positioned the "col1" <p> at the upper left-hand corner of the "drinks" <div>.
- We positioned the "col2" <p> at 0px from the top and 400px from the left so it sits just to the right of the "col1" <p> element. (This is shown with a dashed line in the diagram below.)



We can simplify this CSS a little bit by removing the positions that specify 0px, because those are set to 0 by default. Modify your code as shown:

CODE TO TYPE:

```
div#content {
  width: 800px;
  margin: auto;
  padding-top: 20px;
  text-align: center;
}
div#drinks {
  position: relative;
}
p {
  width: 400px;
  position: absolute;
}
p#col1 {
  top: 0px;
  left: 0px;
}
+
p#col2 {
  top: 0px;
  left: 400px;
}
```

When we do that, we're left with this final CSS:

CODE TO TYPE:

```
div#content {
  width: 800px;
  margin: auto;
  padding-top: 20px;
  text-align: center;
}
div#drinks {
  position: relative;
}
p {
  width: 400px;
  position: absolute;
}
p#col2 {
  left: 400px;
}
```

Click preview one more time on your HTML file to see the final result.

Go ahead and experiment with positioning your elements. Try negative numbers as well as positive ones. Positioning is a fairly complex topic, but you've learned enough to create some interesting layouts.

Copyright © 1998-2014 O'Reilly Media, Inc.



This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.
See <http://creativecommons.org/licenses/by-sa/3.0/legalcode> for more information.

HTML and CSS: Forms

Lesson Objectives

When you complete this lesson, you will be able to:

- use forms to receive input from a user through our web pages.
- use the placeholder attribute.
- use the new HTML required attribute.
- use various other attributes such as, check boxes, textarea, and dates.
- style forms with CSS.

Basic Input and Attributes

So far in this course, we've had a one-way conversation with the user: your content is displayed in a page for the user to read.

But we can also receive input from a user through our web pages. In order to do that we use a *form*. Typically, a user enters information into a form, that information is sent to a program running on a server, the program processes the form, pulls out the information, and then does something with it.

For example, you've probably used Google's form to submit a search term. Google's server takes your search term, looks in its database for web pages that match that term, and then sends you back a web page with those search results.


Let's create our own form now. Type in the code below as shown:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title>Forms</title>
  <meta charset="utf-8">
</head>
<body>

  <form method="get" action="https://students.oreillyschool.com/coderunner/formtester.php">
    <label for="color">What is your favorite color?</label>
    <input type="text" name="color" id="color">
    <input type="submit" value="Submit">
  </form>

</body>
</html>
```

Save it in your **/htmlcss1** folder as **forms.html** and click **Preview** . You'll see an *input* box in the web page where you can type your favorite color, and a button that you can click to *submit* the form. The browser submits the form to the URL that's in the **action** attribute of the form element. As we said before, this is typically the URL of a script program that takes the values you type into a form and processes them in some way.

Note

We've created a basic server script that only displays the data you entered in the form. There's plenty of fun to be had with forms without complicated server scripts. They'll let you get the hang of writing forms and a feel for what to expect if and when you do get to hook them up to a real server script. For the rest of this lesson, you can continue to use this basic server script, or just leave the action attribute empty so you can focus on building the forms themselves.

The **method** attribute is used to specify one of two methods for submitting a form: **get** or **post**. You'll need to know a lot more about these methods once you start submitting forms to real server scripts, but for now just think of them as

two different ways to submit data. We'll be using **get** in this lesson.

We've got two `<input>` elements in this form; one is a text input and the other is a submit button. We use the **type** attribute to determine which type of input we want. There are quite a few different input types; we'll cover a couple of them in this lesson.

The **submit** input type automatically creates a button in the form. For this `<input>` element, we also used a **value** attribute with a string value of "Submit." This is the text that appears on the button. You can make the text say anything you want (within reason). Try changing it now to "Click Me!"

Along with the two `<input>` elements, we also used a **<label>** element to give the text `<input>` a label with instructions for the user. The text `<input>` has an id of "color," and in our `<label>` element we specified a **for** attribute and used the id of the text `<input>` as the value of the **for** attribute. We do this to specify that the label is associated with a specific `<input>` element. This is especially important for accessibility, because it enables screen readers (a screen reader is a software application that attempts to identify and interpret what is being displayed on the screen) to make sure they are reading the right label when the user tabs into an `<input>` element.

The `<input>` element with the id "color" also has a **name** attribute, which is also set to "color." The name attribute is used by the script to which you submit the form. The script uses the name of the `<input>` element to get its value. Don't worry about that for now though. For our purposes in this course, just be aware that any `<input>` element you want the script to be able to access must have a name (and typically that name will have the same value as the id).

Placeholder Attribute


There's a new attribute in HTML5 for `<input>` elements known as the **placeholder** attribute. Try updating your HTML to use this attribute now:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title>Forms</title>
  <meta charset="utf-8">
</head>
<body>

  <form method="get" action="https://students.oreillyschool.com/coderunner/formt
ester.php">
    <label for="color">What is your favorite color?</label>
    <input type="text" name="color" id="color" placeholder="blue">
    <input type="submit" value="Submit">
  </form>

</body>
</html>
```

Feel free to substitute any color for "blue" above. Click [Preview](#) . The placeholder attribute gives the user a hint about the kind of content they should put into the input box. The input box displays your placeholder string in grey. Now, click in the input box; the text goes away and you can type in any color you want. The value isn't really there (if you submit the form without typing anything, the placeholder value doesn't get submitted); it's just a hint for the user.

Required Attribute


HTML5 has introduced the new **required** attribute. This is a *boolean* attribute, which means that you don't have to supply a value for the attribute; the attribute is either present or not. If it is there, then it's active.

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title>Forms</title>
  <meta charset="utf-8">
</head>
<body>

  <form method="get" action="https://students.oreillyschool.com/coderunner/formt
ester.php">
    <label for="color">What is your favorite color?</label>
    <input type="text" name="color" id="color" placeholder="blue" required>
    <input type="submit" value="Submit">
  </form>

</body>
</html>
```

Click **Preview**  and then click **Submit** without entering a color. By adding **required** to this `<input>` element, we're saying that the user *must* enter a value for this form element before submitting the form. Depending on your browser, you'll see different behavior, but you might see a little popup that indicates that you need to complete that field before submitting the form. If a browser does not support the required attribute, the form will be submitted as usual.

Note

Not all browsers have implemented the placeholder and required attributes yet. If you're using the most recent versions of Safari, Chrome, Firefox, or Opera, the placeholder attribute should work. If you're using Chrome, Firefox, or Opera, the required attribute should work. More browsers will support more features in the future, so keep trying, and be aware that not all users will have access to these new features if you use them to create a web page.

Other Kinds of Inputs

Aside from basic text input, there are several other types of inputs you can use.

Checkboxes and Radio Buttons

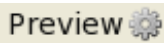
The input type **checkbox** creates a checkbox item in the form to allow the user to select an item from a list.

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title>Forms</title>
  <meta charset="utf-8">
</head>
<body>
  <form method="get" action="https://students.oreillyschool.com/coderunner/formt
ester.php">
    <label for="color">What is your favorite color?</label>
    <input type="text" name="color" id="color" placeholder="blue" required>

    <p>Are you...</p>
    <input type="checkbox" id="funny" name="personality" value="funny">
      <label for="funny">funny</label>
    <input type="checkbox" id="smart" name="personality" value="smart">
      <label for="smart">smart</label>
    <input type="checkbox" id="shy" name="personality" value="shy">
      <label for="shy">shy</label>
    <input type="checkbox" id="outgoing" name="personality" value="outgoing">
      <label for="outgoing">outgoing</label>
    <br>
    <input type="submit" value="Submit">
  </form>

</body>
</html>
```

Click . You see a list of items, each with a checkbox to its left. Click an item's checkbox and a check mark will appear. Try clicking on more than one item; you can select multiple checkboxes, as you might expect.

Let's go over a few things about the form. This time, we put the labels after the inputs so that the checkboxes would appear to the left of each label. How you design your form is up to you, but make sure you consider the usability of your design and what users will expect in different situations; make sure it's clear which checkbox goes with which item.

Also, notice that each input has both an **id attribute** and a **name attribute**. Why both? Well, we need the **id** attribute so we can associate each label with an input (remember, each label goes with only one form control, and ids must be unique). We need the **name** attribute so that we can indicate that all the checkboxes belong together. This is important on the server side when processing a form: if the checkbox inputs that are grouped together have the same **name**, then the server script knows that all the choices are, in this case, related to personality. If you have multiple groups of checkboxes or other controls in your form, this will become really important.

Finally, each checkbox has a different value in the **value attribute**. This is how you allow the user to select different items. Each value in a group of checkboxes should be different, so that in the server script, you can distinguish between responses.

Note

We are using a basic PHP program to process the form submission, so if you select more than one check box and Submit, only the last box checked will be returned. If you want to see multiple results, you have to name the field as an array in the PHP script and change the HTML code to reflect that, using **personality[]** as the name of each of the checkboxes instead, like this:
<input type="checkbox" id="funny" name="personality[]" value="funny">.

Now, try changing "checkbox" to "radio" as the input type of the form controls. Try selecting different items. With type **checkbox**, you can select any or all of the items; with type **radio**, you can only select one at a time.

Just like with checkboxes, it's important to give every radio button in a group the same **name**.

Textarea

Textarea is another useful form control. You can use a `<textarea>` element within a form whenever you need to allow multiple lines of input:

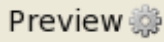
CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title>Forms</title>
  <meta charset="utf-8">
</head>
<body>
  <form method="get" action="https://students.oreillyschool.com/coderunner/formt
ester.php">
    <label for="color">What is your favorite color?</label>
    <input type="text" name="color" id="color" placeholder="blue" required>

    <p>Are you...</p>
    <input type="checkbox" id="funny" name="personality" value="funny">
      <label for="funny">funny</label>
    <input type="checkbox" id="smart" name="personality" value="smart">
      <label for="smart">smart</label>
    <input type="checkbox" id="shy" name="personality" value="shy">
      <label for="shy">shy</label>
    <input type="checkbox" id="outgoing" name="personality" value="outgoing">
      <label for="outgoing">outgoing</label>
    <br>

    <p>Tell us a little about yourself:</p>
    <textarea name="textarea" rows="12" cols="38">
    </textarea>
    <input type="submit" value="Submit">
  </form>

</body>
</html>
```

Click . You'll see a big area (with 12 rows and 38 columns) appear in your page where you can type in a lot of text.

Notice that unlike the `<input>` element, which is a void element, the `<textarea>` element has both opening and closing tags. If you want to provide some default text for the user in the `<textarea>` element, you can type the text in the content of the element. Try that now.

Dates

There are several input types that are new in HTML5 and allow you to specify certain kinds of input more precisely. For example, the input type **date** implies that a text input field should be used for a date. Try adding a date input now:


CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title>Forms</title>
  <meta charset="utf-8">
</head>
<body>
  <form method="get" action="https://students.oreillyschool.com/coderunner/formt
ester.php">
    <label for="color">What is your favorite color?</label>
    <input type="text" name="color" id="color" placeholder="blue" required>

    <p>Are you...</p>
    <input type="checkbox" id="funny" name="personality" value="funny">
      <label for="funny">funny</label>
    <input type="checkbox" id="smart" name="personality" value="smart">
      <label for="smart">smart</label>
    <input type="checkbox" id="shy" name="personality" value="shy">
      <label for="shy">shy</label>
    <input type="checkbox" id="outgoing" name="personality" value="outgoing">
      <label for="outgoing">outgoing</label>
    <br>

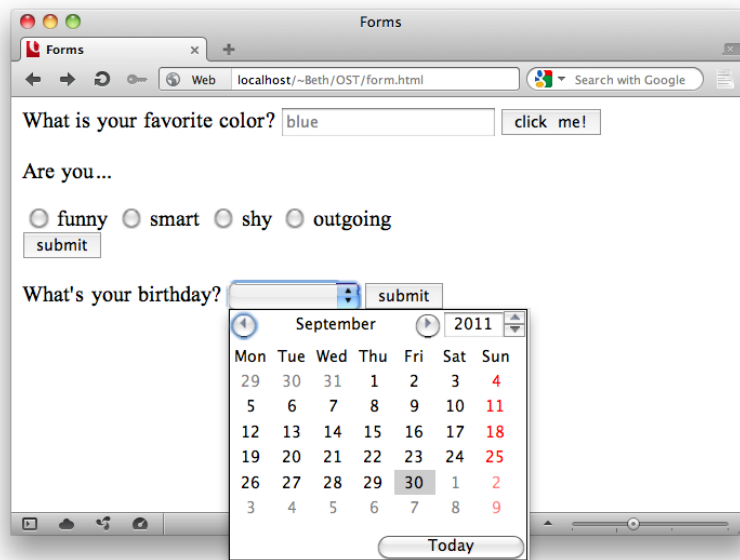
    <p>Tell us a little about yourself:</p>
    <textarea name="textarea" rows="12" cols="38">
    </textarea>

    <p><label for="date">What's your birthday?</label></p>
    <input name="date" id="date" type="date">
    <input type="submit" value="Submit">
  </form>
</body>
</html>
```

Click **Preview** . Now, click in the date input field. Depending on which browser you're using, you'll see different behavior.

Note

As of this writing, the date input field has yet to be fully implemented in some browsers. You'll see an input field, but the behavior will differ from browser to browser. For example, in Chrome and Safari, you'll get up/down arrows that help you select a date in the correct date format. In Firefox, you may see a field that looks and behaves just like a regular text input field. And in current versions of Opera, you'll see a really nice date picker pop-up. If you have Opera installed, it's worth trying it!



Styling Forms with CSS

You can use all your CSS skills to style forms too. Let's try styling a simple form now. We'll use our very first form as an example:

CODE TO TYPE:

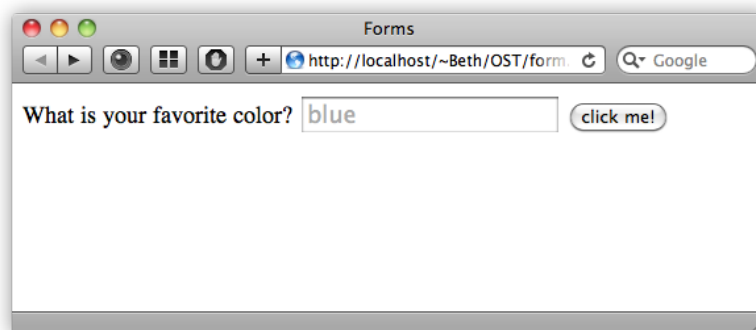
```
<!doctype html>
<html lang="en">
<head>
  <title>Forms</title>
  <meta charset="utf-8">
</head>
<body>
  <form id="stylish" method="get"
    action="https://students.oreillyschool.com/coderunner/formtester.php">
    <label for="color">What is your favorite color?</label>
    <input type="text" name="color" id="color" placeholder="blue" required>
    <input type="submit" value="Submit">
  </form>
</body>
</html>
```


You probably want to keep your original forms.html, so save this new one (in your **/htmlcss1** folder) as **formstyle.html**. Make sure you have the id "stylish" in the <form> element. We'll need this to select this form for styling (although if it's the only form in your page, you can also use the tag name "form" to select it in CSS). Now let's add some CSS. You can add this CSS in a <style> element in the <head> of your document, or link to an external CSS file; it's up to you. For now, let's assume that you'll just add a <style> element. Type the code below as shown:

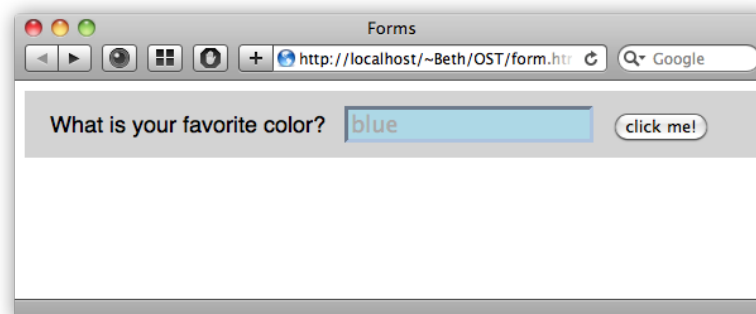
CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title>Forms</title>
  <meta charset="utf-8">
  <style>
    form#stylish {
      background-color: lightgrey;
      padding: 10px 20px 10px 20px;
      font-family: Helvetica, sans-serif;
    }
    input#color {
      margin-left: 10px;
      margin-right: 10px;
      background-color: lightblue;
      border: 3px inset #adc3df;
      color: darkblue;
    }
  </style>
</head>
<body>
  <form id="stylish" method="get"
    action="https://students.oreillyschool.com/coderunner/formtester.php">
    <label for="color">What is your favorite color?</label>
    <input type="text" id="color" placeholder="blue" required>
    <input type="submit" value="Submit">
  </form>
</body>
</html>
```

Every browser displays form elements just a little differently. Here's what your form will look like before you add this style (in Safari):



Now click **Preview** . Here's what your form will look like with the style you just added (again, in Safari):



As you can see, you can use CSS to give your form a very different look! First, we styled the entire form, giving it a light grey background color, added a little extra padding, and changed the default font. Then we styled the text input. We added some space on the left and right by increasing the margin, changed the background color of the input field, and

added a border. We also changed the color of the text you type into the input field, so the text you enter is in blue instead of black. Go ahead and give it a try. Play around with the style of this form and some of the others you created during this lesson.

Where Do We Go from Here?

Be sure to experiment more with forms before you go on to the next lab. If you're interested in learning how to process forms in the browser with JavaScript, take a look at one of the upcoming JavaScript courses.

If you're interested in learning how to process forms on the server side and how to write server scripts, check out these courses:

- [PHP/SQL Programming Certificate](#)
- [Perl Programming Certificate](#)

Copyright © 1998-2014 O'Reilly Media, Inc.



*This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.
See <http://creativecommons.org/licenses/by-sa/3.0/legalcode> for more information.*

HTML and CSS: Tables

Lesson Objectives

When you complete this lesson, you will be able to:


- create a table.
- create a header for your table.
- include a caption in your table.
- use multiple columns.
- style your table.
- use tables and forms together.

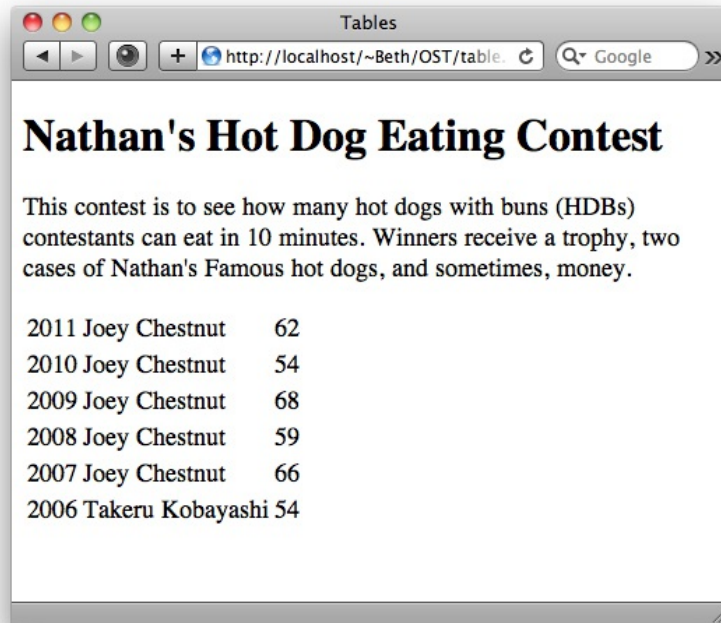
Creating a Table

Sometimes, you'll need to put columnar or tabular data in your web pages. Conveniently, there's an element designed specifically for that: the `<table>` element. Let's create a table right now. For data, we'll use some very important data from Wikipedia: the last six winners of Nathan's Hot Dog Eating Contest! Create this file in the editor below, using HTML syntax:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title>Tables</title>
  <meta charset="utf-8">
</head>
<body>
  <h1>Nathan's Hot Dog Eating Contest</h1>
  <p>
    This contest is to see how many hot dogs with buns (HDBs)
    contestants can eat in 10 minutes. Winners receive a trophy,
    two cases of Nathan's Famous hot dogs, and sometimes, money.
  </p>
  <table>
    <tr>
      <td>2011</td> <td>Joey Chestnut</td> <td>62</td>
    </tr>
    <tr>
      <td>2010</td> <td>Joey Chestnut</td> <td>54</td>
    </tr>
    <tr>
      <td>2009</td> <td>Joey Chestnut</td> <td>68</td>
    </tr>
    <tr>
      <td>2008</td> <td>Joey Chestnut</td> <td>59</td>
    </tr>
    <tr>
      <td>2007</td> <td>Joey Chestnut</td> <td>66</td>
    </tr>
    <tr>
      <td>2006</td> <td>Takeru Kobayashi</td> <td>54</td>
    </tr>
  </table>
</body>
</html>
```

Save it in your `/htmlcss1` folder as `tables.html`, and click [Preview](#) . You'll see a page that looks like this:



To create the table, we start with the `<table>` element. Then, for each row of data, we use a `<tr>` element (table row). Nested within each row are the columns. Each column is specified with a `<td>` element (table data). Each row has the same number of `<td>` elements in it; this is important for keeping the table consistent. The table will have as many columns as the row with the most columns, and will display empty columns for the rows that have fewer `<td>` elements, but it's easier to read (and troubleshoot later) if you use `<td></td>` to indicate which columns you intend to leave empty.

Make sure you close each `<tr>` element after you've specified each row of data, and close the table as well by inserting `</table>` at the end.

We put all of the columns in one line (that is, we put all the `<td>` elements on the same line in the HTML), but you don't have to do this; in fact, it's sometimes easier to make sure you've closed all your elements properly if you split each `<td>` element out into a separate line. It doesn't make any difference to the display or function of the table if you use multiple lines in the code. Try this by changing one row of your table (we'll just show the `<table>` section of the code for now; it's the only part we'll be changing for a while):

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title>Tables</title>
  <meta charset="utf-8">
</head>
<body>
  <h1>Nathan's Hot Dog Eating Contest</h1>
  <p>
    This contest is to see how many hot dogs with buns (HDBs)
    contestants can eat in 10 minutes. Winners receive a trophy,
    two cases of Nathan's Famous hot dogs, and sometimes, money.
  </p>
  <table>
    <tr>
      <td>2011</td>
      <td>Joey Chestnut</td>
      <td>62</td>
    </tr>
    <tr>
      <td>2010</td> <td>Joey Chestnut</td> <td>54</td>
    </tr>
    <tr>
      <td>2009</td> <td>Joey Chestnut</td> <td>68</td>
    </tr>
    <tr>
      <td>2008</td> <td>Joey Chestnut</td> <td>59</td>
    </tr>
    <tr>
      <td>2007</td> <td>Joey Chestnut</td> <td>66</td>
    </tr>
    <tr>
      <td>2006</td> <td>Takeru Kobayashi</td> <td>54</td>
    </tr>
  </table>
</body>
</html>
```

Click . The table in your page looks exactly the same as before. Change it back when you finish.


Note The indentation we use is not strictly necessary for correct page display, but it makes our code easier to follow.

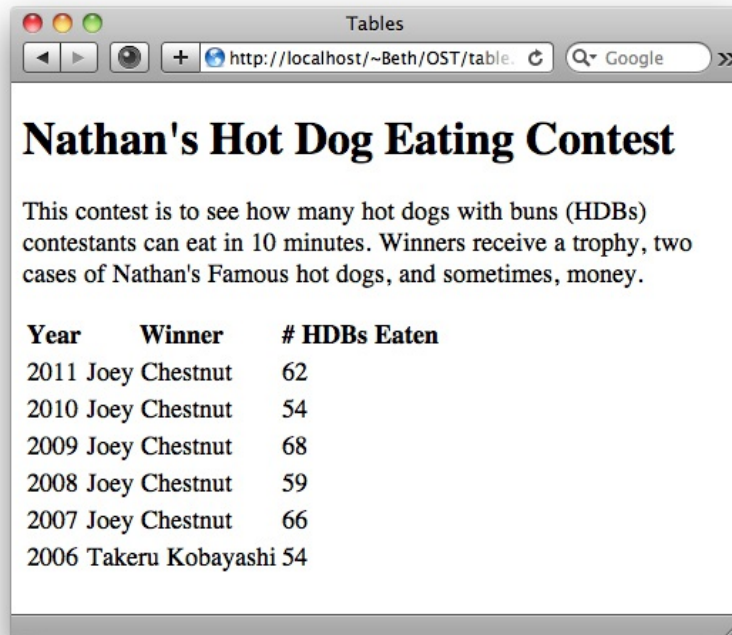
Table Headers

You can probably guess what kind of data is in each column of your table, but it would be better if you could actually *show* it. You can do that using the `<th>` element (**t**able **h**earer). Add a new row to the top of the table with information about the data in each column:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title>Tables</title>
  <meta charset="utf-8">
</head>
<body>
  <h1>Nathan's Hot Dog Eating Contest</h1>
  <p>
    This contest is to see how many hot dogs with buns (HDBs)
    contestants can eat in 10 minutes. Winners receive a trophy,
    two cases of Nathan's Famous hot dogs, and sometimes, money.
  </p>
  <table>
    <tr>
      <th>Year</th> <th>Winner</th> <th># HDBs Eaten</th>
    </tr>
    <tr>
      <td>2011</td> <td>Joey Chestnut</td> <td>62</td>
    </tr>
    <tr>
      <td>2010</td> <td>Joey Chestnut</td> <td>54</td>
    </tr>
    <tr>
      <td>2009</td> <td>Joey Chestnut</td> <td>68</td>
    </tr>
    <tr>
      <td>2008</td> <td>Joey Chestnut</td> <td>59</td>
    </tr>
    <tr>
      <td>2007</td> <td>Joey Chestnut</td> <td>66</td>
    </tr>
    <tr>
      <td>2006</td> <td>Takeru Kobayashi</td> <td>54</td>
    </tr>
  </table>
</body>
</html>
```

Click **Preview** . The items in the table headers are bold and centered; that's the default style that browsers use to display table headers. Here's what the table looks like with table headers in Safari:




Captions

Next, let's give our table a title by using the `<caption>` element. This element must be placed immediately after the opening `<table>` tag.

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title>Tables</title>
  <meta charset="utf-8">
</head>
<body>
  <h1>Nathan's Hot Dog Eating Contest</h1>
  <p>
    This contest is to see how many hot dogs with buns (HDBs)
    contestants can eat in 10 minutes. Winners receive a trophy,
    two cases of Nathan's Famous hot dogs, and sometimes, money.
  </p>
  <table>
    <caption>Winners, 2006-2011</caption>
    <tr>
      <th>Year</th> <th>Winner</th> <th># HDBs Eaten</th>
    </tr>
    <tr>
      <td>2011</td> <td>Joey Chestnut</td> <td>62</td>
    </tr>
    <tr>
      <td>2010</td> <td>Joey Chestnut</td> <td>54</td>
    </tr>
    <tr>
      <td>2009</td> <td>Joey Chestnut</td> <td>68</td>
    </tr>
    <tr>
      <td>2008</td> <td>Joey Chestnut</td> <td>59</td>
    </tr>
    <tr>
      <td>2007</td> <td>Joey Chestnut</td> <td>66</td>
    </tr>
    <tr>
      <td>2006</td> <td>Takeru Kobayashi</td> <td>54</td>
    </tr>
  </table>
</body>
</html>
```


Click **Preview** . Your caption will be centered at the top of the table. (Don't worry, we'll style the caption and the table shortly to make this all look better.)

Spanning Multiple Columns

It's good practice to identify where your data is sourced (and use only public data unless you have specific permission!). In our case, the table data is from Wikipedia, so let's add a row at the bottom of the table to give this information and provide a link to the Wikipedia page:


CODE TO TYPE:

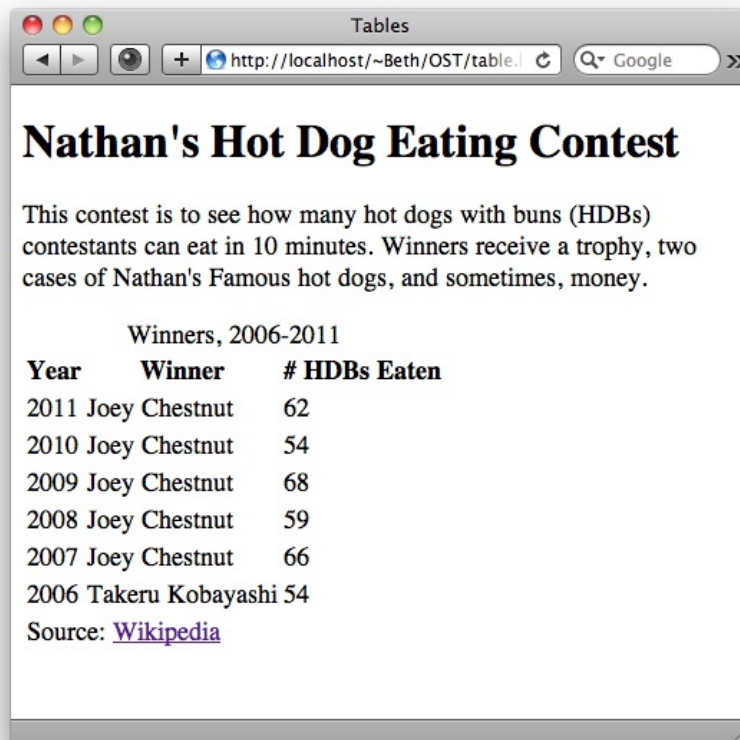
```
<!doctype html>
<html lang="en">
<head>
  <title>Tables</title>
  <meta charset="utf-8">
</head>
<body>
  <h1>Nathan's Hot Dog Eating Contest</h1>
  <p>
    This contest is to see how many hot dogs with buns (HDBs)
    contestants can eat in 10 minutes. Winners receive a trophy,
    two cases of Nathan's Famous hot dogs, and sometimes, money.
  </p>
  <table>
    <caption>Winners, 2006-2011</caption>
    <tr>
      <th>Year</th> <th>Winner</th> <th># HDBs Eaten</th>
    </tr>
    <tr>
      <td>2011</td> <td>Joey Chestnut</td> <td>62</td>
    </tr>
    <tr>
      <td>2010</td> <td>Joey Chestnut</td> <td>54</td>
    </tr>
    <tr>
      <td>2009</td> <td>Joey Chestnut</td> <td>68</td>
    </tr>
    <tr>
      <td>2008</td> <td>Joey Chestnut</td> <td>59</td>
    </tr>
    <tr>
      <td>2007</td> <td>Joey Chestnut</td> <td>66</td>
    </tr>
    <tr>
      <td>2006</td> <td>Takeru Kobayashi</td> <td>54</td>
    </tr>
    <tr>
      <td>
        Source: <a href="http://en.wikipedia.org/wiki/Nathan's_Hot_Dog_Eating_Contest"
        >Wikipedia</a>
      </td>
    </tr>
  </table>
</body>
</html>
```

Click **Preview** . What happened? We added a row with only one piece of data (one column), while the rest of our table rows each have three columns. We want our bottom row to go across all three columns—how do we do that? Using the *colspan attribute* in the `<td>` element:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title>Tables</title>
  <meta charset="utf-8">
</head>
<body>
  <h1>Nathan's Hot Dog Eating Contest</h1>
  <p>
    This contest is to see how many hot dogs with buns (HDBs)
    contestants can eat in 10 minutes. Winners receive a trophy,
    two cases of Nathan's Famous hot dogs, and sometimes, money.
  </p>
  <table>
    <caption>Winners, 2006-2011</caption>
    <tr>
      <th>Year</th> <th>Winner</th> <th># HDBs Eaten</th>
    </tr>
    <tr>
      <td>2011</td> <td>Joey Chestnut</td> <td>62</td>
    </tr>
    <tr>
      <td>2010</td> <td>Joey Chestnut</td> <td>54</td>
    </tr>
    <tr>
      <td>2009</td> <td>Joey Chestnut</td> <td>68</td>
    </tr>
    <tr>
      <td>2008</td> <td>Joey Chestnut</td> <td>59</td>
    </tr>
    <tr>
      <td>2007</td> <td>Joey Chestnut</td> <td>66</td>
    </tr>
    <tr>
      <td>2006</td> <td>Takeru Kobayashi</td> <td>54</td>
    </tr>
    <tr>
      <td colspan="3">Source:
        <a href="http://en.wikipedia.org/wiki/Nathan's_Hot_Dog_Eating_Contest">Wikipedia</a>
      </td>
    </tr>
  </table>
</body>
</html>
```

Click **Preview** . The last row of the table spans across all three columns!




Styling Tables

Our table is just about complete at this point, but it could look a lot better. As you might expect, we can style this table using CSS. Again, you can add your style in the HTML file or in a separate CSS file. We'll just show the CSS in our examples here to save space:

CODE TO TYPE:


```
td {  
    padding: 10px;  
}  
th {  
    padding: 5px;  
}
```

Make sure you add this style in the `<head>` element in your document. Click [Preview](#) . Each of your table cells now has more space, which makes the table a bit easier to read. We added 10 pixels of padding (on each side) to the table data (`<td>` element) items, and 5 pixels of padding (again, on each side) to the table header (`<th>` element) items.

We can make the table easier to read by changing the background color of every other row. This is a common display technique for tabular data. We'll use the `:nth-child` CSS selector to do it.

CODE TO TYPE:

```
td {  
    padding: 10px;  
}  
th {  
    padding: 5px;  
}  
tr:nth-child(even) {  
    background-color: lightgrey;  
}
```

Click [Preview](#) . Do you see every other row in your table with a background color of light grey?

The **:nth-child** selector might look a little weird to you. It begins with a colon and has parentheses, which is kind of odd, right? The **:nth-child** is used to modify another selector, in our case, the **tr** selector. What it means is that for all the **<tr>** elements in the table, our program will select the even ones. So if there are six **<tr>** elements in the table, the **tr:nth-child(even)** selector will select the second, fourth, and sixth **<tr>** elements.

Of course, you can select the odd items instead, using **:nth-child(odd)**. Try it, and you'll see why we used "even" in this example (it's because we wanted our table headers to have a white background). If we use (odd) instead of (even), it doesn't work because the table headers aren't data.


Note The **:nth-child** selector is only supported in Internet Explorer 9 and later, so if you're using an earlier version of Internet Explorer, try another browser.

The **:nth-child** selector is also very handy with lists! Try it the next time you create a list by using the selector **li:nth-child(even)** in your CSS. You'll see how this is much easier than adding a class to every other list item like we did earlier!

Our table is starting to look a whole lot better, don't you think? Let's style the caption next. We'll turn it into something that looks more like a title for the table:

CODE TO TYPE:

```
td {
    padding: 10px;
}
th {
    padding: 5px;
}
tr:nth-child(even) {
    background-color: lightgrey;
}
caption {
    font-weight: bold;
    font-size: 130%;
    border-bottom: 1px solid black;
}
```

Click **Preview** . You'll see that the caption is now bold, in a bigger font, and has a black line under it. Experiment a bit with the CSS for the caption if you like!

Lastly, let's style the last row of the table where we've added the source we used for the table data. To select the last row, we'll add a class to that particular **<tr>** element, and use that class to select the row for styling.


CODE TO TYPE:

```
<table>
  <caption>Winners, 2006-2011</caption>
  <tr>
    <th>Year</th> <th>Winner</th> <th># HDBs Eaten</th>
  </tr>
  <tr>
    <td>2011</td> <td>Joey Chestnut</td> <td>62</td>
  </tr>
  <tr>
    <td>2010</td> <td>Joey Chestnut</td> <td>54</td>
  </tr>
  <tr>
    <td>2009</td> <td>Joey Chestnut</td> <td>68</td>
  </tr>
  <tr>
    <td>2008</td> <td>Joey Chestnut</td> <td>59</td>
  </tr>
  <tr>
    <td>2007</td> <td>Joey Chestnut</td> <td>66</td>
  </tr>
  <tr>
    <td>2006</td> <td>Takeru Kobayashi</td> <td>54</td>
  </tr>
  <tr>
    <td class="source" colspan="3">Source:
      <a href="http://en.wikipedia.org/wiki/Nathan's_Hot_Dog_Eating_Contest">Wikipedia<
/a>
    </td>
  </tr>
</table>
```

Now that we've got a class to use to select the last row of the table, we can style it:

CODE TO TYPE:

```
td {
  padding: 10px;
}
th {
  padding: 5px;
}
tr:nth-child(even) {
  background-color: lightgrey;
}
caption {
  font-weight: bold;
  font-size: 130%;
  border-bottom: 1px solid black;
}
td.source {
  font-style: italic;
  background-color: white;
  border-top: 1px dotted black;
}
```

Click . The style of the source information is now italicized and the row has a dotted black border above it to separate it from the rest of the data.

Why do you think we added a background color property to the rule for **td.source**? What happens if you take that property out?

We're done styling this table, and it looks much better than it did before we added style. Continue experimenting with your table and try using other styles. Here's what your table should look like now (in Safari):

Nathan's Hot Dog Eating Contest

This contest is to see how many hot dogs with buns (HDBs) contestants can eat in 10 minutes. Winners receive a trophy, two cases of Nathan's Famous hot dogs, and sometimes, money.

Winners, 2006-2011

Year	Winner	# HDBs Eaten
2011	Joey Chestnut	62
2010	Joey Chestnut	54
2009	Joey Chestnut	68
2008	Joey Chestnut	59
2007	Joey Chestnut	66
2006	Takeru Kobayashi	54

Source: [Wikipedia](#)

Tables and Forms

When you are using forms, you're entering data, so tables and forms often go well together. Let's create a simple form for entering hot dog data for people, with input fields for their name and the greatest number of hot dogs they've eaten at one sitting. For this part of the lesson, you can either create a new file or modify the one you've already got. We'll proceed as if you're creating a new file:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title>Tables</title>
  <meta charset="utf-8">
</head>
<body>
  <form method="get" action="">
    <label for="name">Name:</label>
    <input type="text" id="name">
    <label for="numhotdogs">Number of hot dogs:</label>
    <input type="number" id="numhotdogs">
  </form>
</body>
</html>
```


Save it in your **/htmlcss1** folder as **hotdogform.html**. We've got a form with two labels and two input fields. (We won't bother with an action for the form since we're just experimenting, using a table to structure form data). We'll use the first label and input field pair for the name. The `<input>` element has a type of "text" and an id of "name," so the label that goes with it has the value "name" in the **for** attribute. Similarly, the input field for the number of hot dogs has an id of "numhotdogs," so the label that goes with it has "numhotdogs" as the value for its **for** attribute.

The input type for the number of hot dogs is a new input type, **number**. This is a new type in HTML5 and in many browsers will be displayed as a field with an up/down arrow next to it that you can use to select a number. Many browsers will also require typing a number into this field and will not let you submit the form if you type in text.

Can you see how you might structure this form as a table? We've got two items: the labels, which are like table headings (they identify what kind of data the corresponding input field is) and the input fields, which are the location where the table data will be entered. Let's structure the table so that the headings (the labels) are in the first column and the data (the input fields) are in the second column (while we're at it, we'll also add a caption to identify the table):

CODE TO TYPE:


```
<!doctype html>
<html lang="en">
<head>
  <title>Tables</title>
  <meta charset="utf-8">
</head>
<body>
  <form method="get" action="">
    <table>
      <caption>Hot Dog Data Entry</caption>
      <tr>
        <th><label for="name">Name:</label></th>
        <td><input type="text" id="name"></td>
      </tr>
      <tr>
        <th><label for="numhotdogs">Number of hot dogs:</label></th>
        <td><input type="number" id="numhotdogs"></td>
      </tr>
    </table>
  </form>
</body>
</html>
```

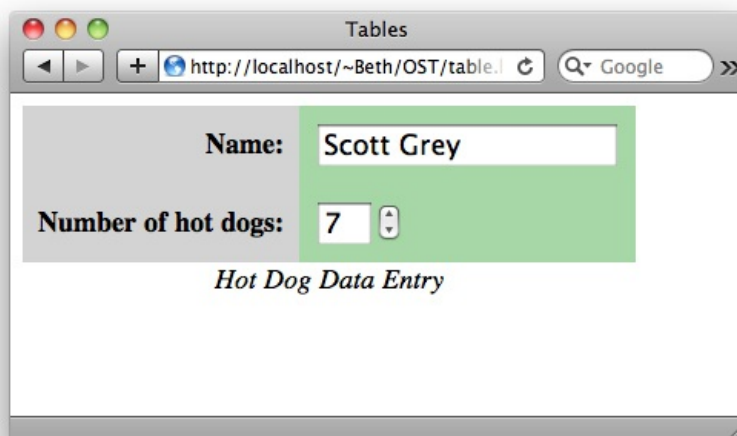
Click **Preview** . The first row of the table provides a place to enter the name of the person eating hot dogs, with the label in the first column and the input field in the second column. The second row is for the number of hot dogs. We're mixing the various table-related elements with the various form-related elements to get both the form structure we need and the table structure we want.

Now let's style this table/form. Type the code as shown below:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title>Tables</title>
  <meta charset="utf-8">
  <style>
    table {
      border-collapse: collapse;
    }
    th {
      background-color: lightgrey;
      text-align: right;
      padding: 10px;
    }
    td {
      background-color: #a6d7a6;
      padding: 10px;
    }
    caption {
      caption-side: bottom;
      font-style: italic;
      font-size: 80%;
    }
  </style>
</head>
<body>
  <form method="get" action="">
    <table>
      <caption>Hot Dog Data Entry</caption>
      <tr>
        <th><label for="name">Name:</label></th>
        <td><input type="text" id="name"></td>
      </tr>
      <tr>
        <th><label for="numhotdogs">Number of hot dogs:</label></th>
        <td><input type="number" id="numhotdogs"></td>
      </tr>
    </table>
  </form>
</body>
</html>
```

Click **Preview** . Your table and form should look something like this (Safari):



Nice, right? You've seen most of the CSS we used to style this table already, so we'll focus on the properties you

haven't used before.

OBSERVE:

```
table {  
  border-collapse: collapse;  
}  
th {  
  background-color: lightgrey;  
  text-align: right;  
  padding: 10px;  
}  
td {  
  background-color: #a6d7a6;  
  padding: 10px;  
}  
caption {  
  caption-side: bottom;  
  font-style: italic;  
  font-size: 80%;  
}
```

The **border-collapse** property is used to remove the default space that the browser adds between the cells of a table (this is the border of each `<td>` or `<th>` element, and you can style it using the `border` property). By removing the border completely, the cells are positioned right next to one another with no space in between them. To see the difference this property makes, try taking out the `border-collapse` property from the table rule and see what happens. Which look do you like better? There is no right or wrong answer, it's all about what works best for your table.

The **caption-side** property gives you control over where the table caption is positioned with respect to the table. In the previous example, we used the caption like a table title by keeping it in the default position above the table and styling it to look like a heading. But sometimes you'll want the caption beneath the table; this is typically how table captions appear in articles or books. To position the caption beneath the table, simply set the **caption-side** to "bottom" as we did here. We also reduced the font size a little and used italic style to make it look a bit more like a caption.

Wow, we've covered a lot of ground with forms and tables so far! Take a break (and maybe even have some tea) before moving on to the last lesson in the course! You're doing great! Hang in there for one more lesson before the final project.

Copyright © 1998-2014 O'Reilly Media, Inc.



This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.
See <http://creativecommons.org/licenses/by-sa/3.0/legalcode> for more information.

HTML and CSS: Standards and Validation

Lesson Objectives

When you complete this lesson, you will be able to:

- validate your HTML pages.
- incorporate accessibility standards into your web pages.
- update your content.

Validating Your Pages

We've already learned that the W3C is the body that creates and manages the HTML and CSS standards that browser-makers implement. They're the people who determine what it means for your HTML pages to be *valid*. But, other than reading the HTML standard and being really careful about how you write your HTML, how do you know your document is valid?

The W3C has an online validator you can use to validate your page. The easiest way to use it (for small HTML files) is to copy and paste your HTML into their online form. Let's create a small HTML file to test:

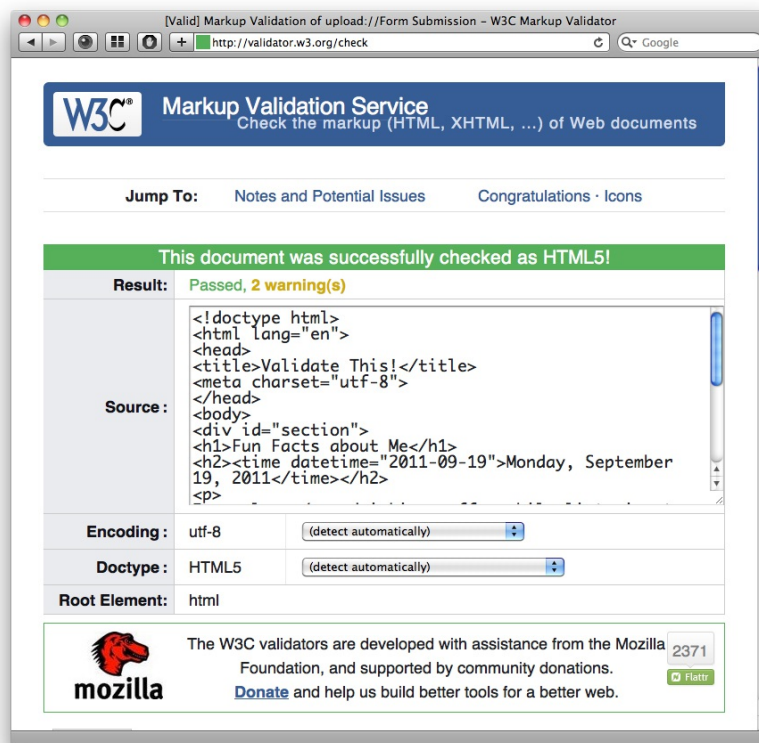
CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title>Validate This!</title>
  <meta charset="utf-8">
</head>
<body>
  <div id="section">
    <h1>Fun Facts about Me</h1>
    <h2>
      <time datetime="2011-09-19">Monday, September 19, 2011</time>
    </h2>
    <p>
      I <em>love</em> drinking coffee while listening to music.
      This morning, I'm listening to:
    </p>
    <ul>
      <li>Pat Metheny</li>
      <li>The Beatles</li>
    </ul>
  </div>
</body>
</html>
```

Save it in your **/htmlcss1** folder as **funfacts.html**. Now we'll validate your HTML:

1. Select all the HTML text in the file (Ctrl+A).
2. Copy it by pressing [Ctrl+C] (PC) or [Command+C] (Mac).
3. Go to http://validator.w3.org/#validate_by_input.
4. Choose "Validate by Direct Input" if it's not already selected.
5. Paste your HTML into the form by pressing [Ctrl+V] (PC) or [Command+V] (Mac).
6. Click the **Check** button.

If you've been careful about typing in your HTML, you'll see a message with a green background that says "This document was successfully checked as HTML5!"



You might also see two warnings; one of those is related to the fact that the validation tool is still in development, and the other is related to the character encoding that is assumed for the text you've pasted into the window (if you upload your file, then the validator can check the file directly for the character encoding and you won't see this warning).


Now let's add an image to this HTML. Update your HTML to include this paragraph and image:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title>Validate This!</title>
  <meta charset="utf-8">
</head>
<body>
  <div id="section">
    <h1>Fun Facts about Me</h1>
    <h2>
      <time datetime="2011-09-19">Monday, September 19, 2011</time>
    </h2>
    <p>
      I <em>love</em> drinking coffee while listening to music.
      This morning, I'm listening to:
    </p>
    <ul>
      <li>Pat Metheny</li>
      <li>The Beatles</li>
    </ul>
    <p>
      while drinking a latte. Here's a picture of my coffee:
    </p>
    
  </div>
</body>
</html>
```

I believe you still have the coffee2.jpg image in your images folder, but if you want to link to the image instead, you can right-click on the image, copy the URL, and use the absolute URL in the src attribute:



Go ahead and click **Preview**  so you can see what the page looks like. It should look fine in your browser. Now follow the steps above and try to validate this HTML (or if you still have the validation page open, paste in the new code, scroll down, and click **Revalidate**).

If all goes according to plan, you should see an error message with a red background: "Error found while checking this document as HTML5!" To find out what went wrong, scroll down until you see the heading "Validation Output" and look at the error:

Validation Output: 1 Error

❌ **Line 22, Column 30: An img element must have an alt attribute, except under certain conditions. For details, consult guidance on providing text alternatives for images.**

``

This tells us that the **alt** attribute of the `` element is *required* in order for the document to be valid. We mentioned previously that the **alt** attribute is used to specify a description of an image. This is particularly useful if, for some reason, the image doesn't load properly and, more importantly, for accessibility. The browser will display the text in the **alt** attribute if the image doesn't load properly, and screen readers will read this text to help the user understand the content of the image.

Add the **alt** attribute to your image, like this:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title>Validate This!</title>
  <meta charset="utf-8">
</head>
<body>
  <div id="section">
    <h1>Fun Facts about Me</h1>
    <h2>
      <time datetime="2011-09-19">Monday, September 19, 2011</time>
    </h2>
    <p>
      I <em>love</em> drinking coffee while listening to music.
      This morning, I'm listening to:
    </p>
    <ul>
      <li>Pat Metheny</li>
      <li>The Beatles</li>
    </ul>
    <p>
      while drinking a latte. Here's a picture of my coffee:
      <br>
      
    </p>
  </div>
</body>
</html>
```

Revalidate the page. Is it fixed?

Experiment with your HTML. Try "breaking" things and see what the validator thinks is valid and not valid. You'll find that you can make a lot of mistakes in your HTML and still get a valid document. For instance, you can leave off a closing tag (go ahead and remove one of the `` tags) and your HTML will still validate. Does it validate if you remove the opening `` tag instead?

All About Standards

HTML5 is designed to be forgiving. Prior to HTML5, HTML was heading in a much more rigorous direction. You may have heard about HTML 4.01 "strict" and XHTML. These standards required you to write your page very carefully and, in order to validate your page, follow the rules precisely.

At this point, you might be asking yourself several questions:

- If HTML5 doesn't care so much about the rules, why should I care?
- How do browsers render (display) my page correctly if I don't follow the rules?
- Is validation important?
- What is XHTML?

These are all great questions! Let's start by considering where we are now with HTML5 compared to where we were just a short while ago with HTML 4.01 strict and XHTML.

Prior to HTML5, the standard for HTML was heading towards XML in the form of XHTML. XML is a strict language, and for good reason: XML is used to create documents that are processed by computer (to exchange files on everything from library books to national security information!) and, as you probably know, computers are extremely precise. They don't tolerate mistakes well. XML files are usually validated against a grammar or "schema" that specifies rules for how the elements are used in the document, so the authors of those XML files know they are exactly correct.

Browsers have similar rule checkers built into them so they can parse your HTML to figure out how to display your document. Traditionally people have not been careful about how they write their HTML, so browsers have always included and allowed lots of fudging of the rules in order to display our pages, even if they weren't written correctly.

So, given that browsers know how to fudge the rules, why should you care about the rules? Well, the main reason is this: if you write your pages correctly, every time, then you'll know the browser is going to display the page correctly for all (or at least, most) of the people viewing your pages.

That's why we've been following slightly stricter rules than the current standard for HTML5. We're following rules from HTML 4.01 strict and XHTML, to get into good habits from the beginning.

Now, what is XHTML? XHTML is a version of HTML written in XML. Because HTML is very similar to XML, the differences between HTML and XHTML aren't that great; the main difference is that if you write XHTML, you *must* follow the rules for writing documents in order for your document to be valid XML.

One advantage to XHTML is that because you're actually writing XML, you can create documents that can be used with systems that know how to handle XML documents. Depending on what kind of business you are in, this can be useful.

For the most part, web developers and web content creators aren't creating "documents." They're creating informational pages or even web applications like games, utilities, or social media. Think of the web sites you use every day; most of them (except perhaps news sites) don't have much to do with traditional "documents."

So, what does this all mean for you? What we recommend is that you use the standard that is most appropriate for the project you're working on; HTML5 for most uses, and XHTML 1.1 for specific uses when you need to be able to validate your documents for some reason. If you go with HTML5, then you can get away with being a bit sloppy, but if you are careful about how you write your HTML, then you will have a greater chance of your web page working as you'd expect it to, especially when you start using CSS and JavaScript more. If you go with XHTML 1.1 or XHTML5 (a hybrid of HTML5 and XHTML) then you need to learn a bit more about XML and how it works. In this course, we focus on HTML5, but keep using the good habits we're learning and adhere to HTML 4.01 and XHTML standards.

To learn more about XHTML, read the [W3C XHTML 1.1 standard document](#).

To learn more about XHTML5, read this [blog post](#) about it.

Accessibility standards

Your content should be accessible to a wide audience regardless of physical abilities. The W3C has specific

guidelines for making your pages accessible to users with disabilities.

People may be visually, hearing, or physically impaired, or technologically limited, but still want to access your content. There are quite a few things you can do to help with that. And, many of the things you can do to make your web page more accessible will also help users on mobile devices too.

To help you get a feel for how your site might appear to someone using assistive devices like a screen reader or a zoom aid, try using your computer and/or browser's built-in Universal Access functions.

For example, if you use Safari, you can enable a screen reader using the **Edit | Speech | Start Speaking** menu option. The screen reader will start reading the current web page. You can also see how your web page will look to people who use increased font sizes by default, by going to the **Safari Preferences | Advanced** panel and increasing the minimum font size.

In Firefox, you can set the font size using the **Preferences | Content** panel, and set accessibility features in the **Preferences | Advanced** panel.

Also, check your computer's Universal Access System Preferences; there are often tools there to help users with disabilities.

Some other ways you can make your site more accessible:

- Always provide text alternatives for images. Be sure the text describes both the graphic and its purpose.
- If your image is a link, include a description/purpose of the link destination.
- Use relative font sizes. Screen enlargers will display the page more accurately. To specify relative font sizes in your CSS, you write properties like **font-size: 120%**; rather than as an absolute size (like **font-size: 12px**).
- Write hyperlink text that makes sense when read out of context.
- Don't use an element because of the *look* it gives your page—you can always change that with CSS—use the element that has the right meaning for the purpose. In particular, you'll want to use headings properly (<h1>, <h2> and so on), because people using screen readers will often jump between headings as a way to find specific content quickly.
- Use labels with your form controls. If you use labels correctly, it can really help people in understanding and correctly filling out forms.
- Use access keys in your links.
- Use color with care, not only for the visually impaired, and color blind.

There are many other ways you can help people with disabilities use your pages; search the web for "tips for making your web site more accessible" and you'll find a wealth of information beyond the W3C guidelines.

Access Keys

For users with limited mobility, using a mouse to click through your site can be difficult. Access keys are similar to the keyboard shortcuts we're all familiar with (for example, copying and pasting), except they are tied to your navigation menu. You can set these keys using the **accesskey** attribute in the anchor tag.

OBSERVE:

```
<a href="http://www.oreillyschool.com" accesskey="h">Home</a>
```

Access keys on a Windows-based machine using IE, Safari, Konqueror, or Chrome are activated by pressing **Alt+Accesskey**.

- Windows: Internet Explorer users may also need to press **Enter** to activate a link.
- Windows: Firefox, press **Shift+Alt+Accesskey**.
- Mac: Firefox 14, press **Ctrl+Alt+Accesskey**.
- Mac: Firefox, press **Ctrl+Accesskey**.
- Mac: Safari, press **Ctrl+Option+Accesskey**.

Think about how you might inform users about your access keys. You could list them on your page, provide a link to a separate page where they are listed, or using CSS styles to style the access key letter.

Create a Great Web Site and Keep It Current

You now have many of the basic skills you need to go out and create a great web site. Even if all you do is put up a personal web page, it's worth keeping it up to date by using the latest standards, making sure that all your links work, and updating the content regularly.

If you link to an external site and they change their pages, you'll end up with a broken link. To avoid that, be sure to check your external links often. You can do that manually or run your pages through [W3C's link checker](#).

Just a Few Loose Ends

Favicons

Have you ever wondered about that little icon that's in the title bar for some websites? That's called a *favicon*. It's a 16 x 16 pixel image associated with a website. Maybe you'd like to include one for your site. Here's how:

First create and upload an .ico image. Then add this code to the head of your web pages:

```
<link rel="icon" type="image/ico" href="http://location/favicon_name.ico">
```

Replace the value of the **href** attribute, *http://location/favicon_name.ico*, with the location and name of your favicon file. The favicon won't show in the Preview window, so be sure to save your file and view it from your computer or online.

More <meta> Tags

Aside from the <meta> tag that's required for creating a valid HTML5 document, and the <meta charset="utf-8"> tag that specifies the character set, there are several other <meta> tags that you might find useful. Here are a few of the most common ones:

- The **name** <meta> can be used to specify a description, keywords, and author for your page. For example:
<meta name="description" content="O'Reilly School HTML and CSS course">
<meta name="keywords" content="HTML, CSS">
<meta name="author" content="Elisabeth Robson">
- The **http-equiv** <meta> can be used to refresh a page regularly. For example:
<meta http-equiv="refresh" content="300">
tells the browser to refresh the web page every 5 minutes. Keep in mind that **refresh** is NOT something you'll want to use regularly if you want your web page to be accessible, because it takes control of the page refresh away from the user. Use this <meta> tag carefully and with good reason. **http-equiv** can also specify a redirect **location**, which can be useful if you need to direct users from an old web site to a new one. For example:
<meta http-equiv="location" content="URL=http://oreillyschool.com">

Alright then. You're almost done with the course! Coming up next: your final project. Have fun with it, and good luck!

Copyright © 1998-2014 O'Reilly Media, Inc.



This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.
See <http://creativecommons.org/licenses/by-sa/3.0/legalcode> for more information.

HTML and CSS: Final Project

Lesson Objectives

When you complete this lesson, you will be able to:

- create a website that include elements and properties discussed in both the HTML and CSS portions of this course.
-

Final Project

This lesson is your final project for the course. The goal of this project is to create an entire website. You can make this website for any subject you wish. If you are having trouble coming up with an idea, you might consider creating an online portfolio and resume. Such a website might include:

- A main page with general information about you
- Links to pages with the following information about you:
 - work experience
 - education
 - interests
 - past and current websites—you can include the pages you created throughout this course
 - links to your favorite websites

For the sake of evaluation, your website must include elements and properties discussed in both the HTML and CSS portions of this course. You should use an external CSS file for use in all the pages in your site. You are encouraged to create files that observe good programming practices and your pages should validate using the [W3C online validator](#). In addition, you should use good names for ids and classes, and use indentation to make your code as readable as possible.

The website should be well-organized, professional, easy to navigate, and aesthetically pleasing, with no typos or syntax errors. The site should also work consistently in the following browsers:

- Internet Explorer on Windows
- Firefox on Windows
- Safari on Mac
- Chrome on Mac

Note

NO WYSIWYG EDITORS. This means no Microsoft Expression Web, MSWord, or Dreamweaver. If you cannot use CodeRunner, you must use a plain text editor (like WordPad, NotePad or TextPad) to create your pages.

Be creative and have fun! You want to present yourself in a professional yet friendly way, so feel free to express yourself!

Copyright © 1998-2014 O'Reilly Media, Inc.



This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.
See <http://creativecommons.org/licenses/by-sa/3.0/legalcode> for more information.