# Lesson 6: Printer Friendly

## HTML, XHTML, and CSS



## Chapter 1


**Introduction**

Welcome back! In the early 1990s when the Web was just getting started, most computer monitors weren't capable of showing colors, photos, or fancy print styles. The Web pages of the day were pretty drab academic papers without a whole lot of pizzazz. The HTML language of the time was similar to what you've learned so far in this course (though in the earliest days there were no tags for pictures or tables).

Things have changed a lot since then. Computers are a lot faster and more sophisticated. Computer screens and printers can display photos, fancy print styles, and complex page layouts. People access the Web with all kinds of mobile devices now—not just Web browsers on PCs and Macs.

To keep up with all of that, HTML has evolved over the years. Of course, it didn't just magically evolve on its own. People have driven that evolution. Specifically, the World Wide Web Consortium (also known as the W3C or W3) has driven the evolution, as they are the people who define the standards.

In today's lesson, you'll learn about the past, present, and future of HTML. And more importantly, you'll learn what you need to do right now to make your pages comply with current and future standards.

Let's meet in Chapter 2 and get started!

## Chapter 2

**The Turn of the Century**

In the early days of the web, HTML was a pretty simple language for displaying basic text documents. And the rules were somewhat loose. You could use uppercase or lowercase letters for your tags, and you could leave off the closing </p>, </li>, </h1> tags for many elements. The rules were loose and simple, because the pages were plain and displayed only on monochrome monitors that couldn't even show pictures, colors, or fancy print styles.

As the years went by, computers and monitors kept getting more sophisticated. In the mid 1990s, HTML went through a period sometimes referred to as the era of the *browser wars.* During that phase, software companies that were making the Web browsers were also making up their own HTML tags and attributes, as well as their own rules about how HTML should be written. This was not a good thing, though. As a Web developer, you might create a page that looks great in one Web browser, say, Internet Explorer in Windows, but then you see your page in some other Web browser on someone else's computer, and it looks completely different—and not so good!

By the late 1990s, professionals in the field were going back to the W3C asking for clearly defined standards so everyone could play by the same rules—and so a Web developer could create a page in one Web browser and not worry about it looking completely different in other Web browsers.

**Deprecated Tags and Attributes**

Right about the turn of the century, we started seeing some cleaner, better HTML specs in what's now known as HTML 4.01, XHTML, and HTML5. The cleanup started by *deprecating* some tags and attributes that had found their way into the specs during the browser wars. The W3C determined that those tags and attributes weren't such a good idea, so they recommended phasing them out of the language. That's what *deprecated* means, "no longer recommended," and so you should avoid using them in pages you create now and in the future.

Below, I'll list the deprecated tags and attributes that you *should not* use when creating your Web pages. (The likelihood is that these tags won't have any meaning to you. I list them only for the benefit of those few students who may have learned older, traditional HTML prior to taking this course. And don't worry—you don't need to know what the tags and attributers do, because you won't be using them anyway.)

| Deprecated Tags | Deprecated Attributes |
|---|---|
| <applet> | (Note that *align*, *height*, and *width* are only partially deprecated.) |
| <basefont /> | align (deprecated for all but table cells) |
|  | alink |
| <dir> | background |
| <font> | bgcolor |
| <frame> | clear |
| <frameset> | color |
| <isindex> | compact |
| <menu> | face |
| <s> | height (deprecated for all but the height of images) |
| <strike> | hspace |
| <u> | language |
| <xmp> | link |
| | noshade |
| | nowrap |
| | size |
| | text |
| | version |
| | vlink |
| | width (deprecated for all but the width of images and tables) |

**Proper Nesting of Tags**

As originally defined, the HTML language didn't impose many rules about how we should organize tags. This ended up putting some extra burden on the user agents interpreting that code, making them do a lot of extra time-consuming processing in dealing with missing end tags or improperly nested tags. So starting with version 4 of HTML at about the turn of the century, the W3C started asking authors to properly nest, and close, all paired tags.

To understand what that means, we'll need an example. Let's use a simple, one-line paragraph of text that contains italic (emphasized) words.

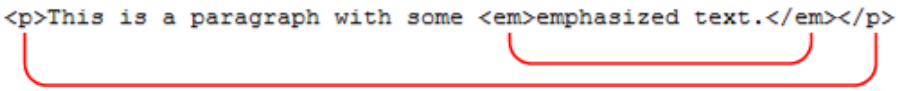
This is a paragraph with some *emphasized text.*

Paragraph with some italic text

To tag that paragraph correctly, we would put both tags for the emphasized text inside the tags for the paragraph.

**Reminder**

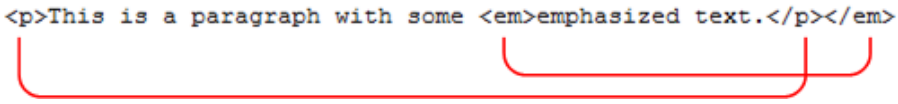
The ... tags mark the beginning and end of a paragraph. The ... tags mark two types of text: text that is spoken with emphasis by screen readers for the blind, and text shown in italics on the screen and printed page.




Properly nested tags

We say the tags above are *properly nested* because the ... tags are contained within (nested within) the ... tags. That's good—just the way you want it.

The wrong way to do it would be to put the closing tag for the em element outside the closing paragraph for the paragraph. The difference becomes more apparent if you envision lines connecting each opening tag with its closing tag.




Tags *not* properly nested

The above example is wrong because the opening  tag is inside the paragraph, but its closing  tag is outside the paragraph. In other words, the em element isn't nested within the paragraph. That's not properly nested and is bad in XHTML.

If you get into the habit of always typing your closing tag right after your opening tag, you won't have to worry about improper nesting. Just don't forget to put the cursor between the two tags before typing the content that belongs between those tags!

The turn of the century did more than deprecate some tags and attributes and clean up the rules of how we should nest tags. It also ushered in a whole new version of HTML called *XHTML*. Meet me in Chapter 3, and we'll discuss what that's about.

## Chapter 3

### The Switch to XHTML

As we discussed in the last chapter, HTML 4.01 cleaned up some problem tags and attributes, as well as the rules about nesting opening and closing tags. But at the time, cutting-edge programmers who were creating new browsers and other user agents were also asking for some tighter rules, so their browsers and other user agents could process pages more efficiently.

Traditional HTML is based on SGML (*Standardized Generalized Markup Language*), which is heavily geared toward print documents commonly found in office environments. Someone suggested creating a version of HTML based on XML, which stands for *eXtensible Markup Language*. XML isn't specifically designed for displaying content on the screen. In fact, its main use is to help transfer data between computers that are otherwise incompatible. It's very good at doing that job, mainly because its rules of syntax, which define how you write the code, are very clear and well defined. The W3C reasoned that carrying those same rules over to HTML could help clean up some of the messiness and unpredictability that was hampering the programmers who were trying to create the newer user agents. And thus was born XHTML—*eXtensible Hypertext Markup Language*.

To hear the name, you might think this is something radically different, requiring you to learn about XML or other exotic languages. But that's not the case at all. The differences between traditional HTML and XHTML are very few. In fact, it's basically just four things:

- Tags and attributes must appear in lowercase letters.

- Closing tags are required for all tag pairs.
- Empty tags end with /> rather than >.
- XHTML documents must have one root element.

The first two items are pretty straightforward. If you learned traditional HTML in the 1990s, you would have learned that it was okay to use uppercase or lowercase letters in your code. And you may have learned that it's okay to leave the </p> tag off the ends of paragraphs and some other tags. For example, this page was okay in early traditional HTML:

```
<HTML>
<HEAD>
  <TITLE>Sample</TITLE>
</HEAD>
<BODY>
  <H1>Sample Page Title
  <P>This is the first paragraph.
  <P>This is the second paragraph.
</BODY>
</HTML>
```

Acceptable code for traditional HTML

That page isn't acceptable by today's standards. The uppercase letters in the tags (<HTML>, <HEAD>, and so forth) aren't allowed. And it's not okay to leave off the closing </h1> and </p> tags on the heading and paragraphs. To abide by the first two rules of XHTML, the page above would need to be rewritten like this:

```
<html>
<head>
  <title>Sample</title>
</head>
<body>
  <h1>Sample Page Title</h1>
  <p>This is the first paragraph.</p>
  <p>This is the second paragraph.</p>
</body>
</html>
```

Lowercase code and closing tags

Fortunately, those first two rules aren't too tough to handle. Just keep an eye on your keyboard's Caps Lock key to make sure you're not typing uppercase letters. And whenever you're typing tags that come in pairs, type the closing tag right after you type the opening tag. That way you won't forget to type the closing tag later.

The vast majority of XHTML tags come in pairs. But there are a few exceptions, and we call those exceptions *empty tags*. That's a little bit of a misnomer since the tags aren't completely empty like <> is considered empty. Rather, they're empty in that there's no closing tag. The table below lists the most commonly used empty tags, how they look in HTML, and how they look in XHTML. (We haven't discussed all of these tags yet, so you don't need to know or understand the purpose of them now. The brief description is just there for future reference.)

| HTML | XHTML | Purpose |
|---|---|---|
| <br> | <br /> | Line break |
| <hr> | <hr /> | Shows a horizontal rule |
| <img> | <img /> | Shows a graphic image |
| <link> | <link /> | Links to an external file |
| <meta> | <meta /> | Provides information about the page |

Many people type a space before the /> at the end of an empty tag. This is because when XHTML first came out, some old Web browsers that weren't ready for XHTML still worked okay so long as you put in that space. But technically, the space isn't required. And those old browsers are now extinct and nothing to worry about. So everything will be fine whether you put a space in front of /> or not. But that's the only

extra blank space that's allowed. Don't put a space between the / and > at the end of the tag!

The last rule of XHTML is that every document must have exactly one *root element*. It sounds technical—maybe even a little intimidating. Basically, it means that the whole page must be contained in <html>...</html> tags, just like the pages you've already created in this course. But there's actually one thing that goes above the first html tag—a *doctype declaration* that lets the user agent "know" your code is written to XHTML standards.

There are a few different XHTML flavors, and different declarations for different flavors. But don't worry about that for now. Most people just use the following declaration at the beginning of their page to indicate that they're using XHTML.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

You may have noticed the uppercase letters in there, which seem to conflict with XHTML's rule of only using lowercase in tags. But that first line isn't a tag, per se. It's a document declaration that happens to be enclosed in < and > characters. The actual tags don't start until the <html tag.

The html tag is a little different in XHTML too. It contains an attribute like this:

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

The xmlns stands for *XML namespace* and is required in all XML-based documents. In a Web page, it doesn't really do anything other than satisfy the requirement of having a namespace defined. But in other kinds of XML documents, the namespace defines the meanings of tags used in the XML document.

The head and body tags from traditional HTML are still required in XHTML too. So an "empty" page in XHTML (one that contains all the mandatory tags, but no content) looks like this:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<title></title>

</head>

<body>

</body>

</html>
```

It's a lot to remember, and to tell you the truth, I don't know of any Web developers who have even tried. Most just save all of that to a text file on their computer somewhere. Then, when they create a new Web page, they copy and paste all the tags into the new page, rather than try to type them from memory. Copying and pasting is a lot easier, and it avoids typographical errors.

But XHTML isn't the end of the story. . . .

**HTML5**

In 2004, a group who called themselves *WHATWG* (Web Hypertext Application Technology Working Group) began thinking about ways to extend the Web to host more than just text and pictures. They started developing tags and technologies that would allow for better integration of video, audio, and more interactivity, without relying on plug-in players like Adobe Flash, commonly used to play certain kinds of videos on the web.

At the same time, the folks who brought you HTML and XHTML were working on a new flavor of XHTML, referred to simply as *XHTML 2.0*.

As the years went by, it became clear to the W3C that the two groups were duplicating some of their efforts. And, of course, the point of

having a group like the W3C define standards is so that there's just *one* standard that any Web developer can abide by to be sure his or her products will look and act the same across different brands and versions of user agents. So the W3C decided that the solution was to merge the WHATWG and XHTML 2.0 efforts into one. And to make matters even easier, they decided to dub the next developed language simply *HTML5*.

HTML5 is basically XHTML with some new tags from the WHATWG group. HTML5 reached Recommendation status in October 2014. That means the language specification will not change in the future. So web developers, like yourself, can use the language now without having to worry about something within that specification changing in the future.

As we discussed, HTML5 is essentially XHTML with some new tags added in. Like XHTML, an HTML5 page must start with <!DOCTYPE. But the first tags are much simpler than they are in XHTML. And a blank page in HTML5 (again, "blank" means that it contains all of the mandatory tags, but no content) looks like this:

```
<!DOCTYPE html>

<html>

<head>

<title></title>

</head>

<body>

</body>

</html>
```

So what should you do? Even though HTML5 isn't a final spec yet, the move in that direction is happening quickly, and most of the tags in HTML5 are exactly the same as they were in earlier versions. In fact, all the tags you learn about in this course will work in HTML5 as well as earlier versions of HTML). But since the doctype for HTML is so much easier to remember and type than the earlier ones, and since everything is moving in the direction of HTML5 anyway, I'm going to suggest you use the newer tags when we get to the assignment for this lesson.

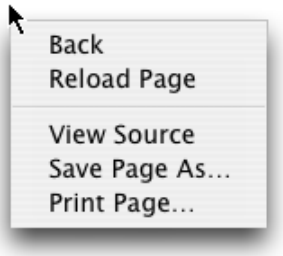
Keep in mind too that when I refer to HTML in this course, I'm really talking about HTML, XHTML, and HTML5. Because they're all mostly just slightly different "flavors" of the basic HTML language, and the similarities far outweigh the differences. I say, "HTML" (as does everyone in the field) just because it's too tiresome (and pointless) to always say, "HTML, XHTML, and HTML5" (or some such thing).

**Viewing Other People's Code**

You may recall from earlier lessons that the code that defines how a page looks in a Web browser is called the page's *source code*. And when you're out on the Web viewing other people's pages, you can take a look at the source code at any time. In Windows, get the mouse pointer on some neutral area of the page, such as the blank line between two paragraphs (not on a picture, and not on a link), and right-click that spot. On a Mac, CTRL + click that spot. Then, look for View Source (or a similar-looking option) on the shortcut menu that opens.

**Tip**

If you don't see an option for viewing the page source in Safari, choose **Preferences** from the Safari menu, click **Advanced**, then select (check) **Show Develop menu in menu bar**. Close the Preferences dialog. Then CTRL + click any empty spot on the page again and look for **Show Page Source**.




Sample shortcut menu for right-clicking or CTRL + clicking on a Web page

Click **View Source** on the shortcut menu, and a new window opens on the screen, showing the source code for the page. Most will probably have a <!DOCTYPE tag up top, indicating what version of HTML the author used to create the page.

Even though you can see the code, you can't make any changes that will affect the original site (you're only allowed to change the content of your own site). So you can close the window that's showing the code by pressing its Close button (the red circle at top left on a Mac, or the X at the top right in Windows).

When you look at other peoples' source code, you'll likely see a lot of strange-looking code. And not all of it is HTML or XHTML. That's because every Web browser is capable of executing more than just HTML and XHTML. There are other languages they can use. CSS is an important one, because it's the language that lets you do your styling (centering, fonts, colors, and so forth). Ready to find out about CSS? Take a break to stretch your legs, and then meet me in Chapter 4.

## Chapter 4

**Introducing CSS**

Back in 1999 when the W3C started deprecating some tags and attributes, part of their motivation was to achieve *separation of structure and presentation*. In English, that has to do with separating the language that described what an element *is* from the language that described how it *looks*. The concept was already in widespread use in other forms of electronic publishing. And people writing websites wanted to see a similar implementation for Web development.

In Web development, the language we use for structure is HTML (including XHTML). We use HTML tags to define what an element is (for example, a heading, a paragraph, an image, a table, a list). For the presentation side of things, we use another language called *CSS* (Cascading Style Sheets). In other words, we use CSS to describe how an element looks (like font, color, and centering).

CSS isn't an alternative to, or a replacement for, HTML or XHTML. You still create your pages and define your elements using HTML (or XHTML), just as you've been doing since day one in this course. You use CSS in addition to that language to style the appearance of the elements that make up your page.

It may seem as though having two separate languages makes things worse and more complicated for developers. For a beginner just getting started, I can certainly understand why it would seem that way. But there are many advantages to keeping the languages separate, as we'll discuss throughout this course. And in fact, it was the Web developers, the people creating the websites, rather than the programmers who were creating the user agents, who were the driving force behind the creation of CSS. The concept isn't new either. The use of style sheet languages to control styling goes back to the early days of electronic publishing, long before the World Wide Web existed.

**CSS History**

The history of CSS is relatively simple compared to HTML. In 1996, the W3C published the Recommendation for CSS Level1 (often written as CSS1). It contained properties and values for basic things like fonts, colors, and text alignment. In 1998, CSS2 was released and included some new techniques for positioning elements on pages, as well as new properties for aural (audio) media to control the computer voice speaking the text out loud.

Both CSS1 and CSS2 got off to a rocky start, as is often the case with new languages. Experience gained from practical use of those early versions led to CSS2.1, which is the version that's been in widespread use for nearly a decade now.

CSS3 is the next version of CSS that the W3C will release. It offers advanced features like rounded corners, drop shadows, and other features developers want in their site. Like HTML5, CSS3 is not a finished product yet. But like its predecessors, CSS3 is destined to become the premier style sheet language for web design.

**CSS Syntax**

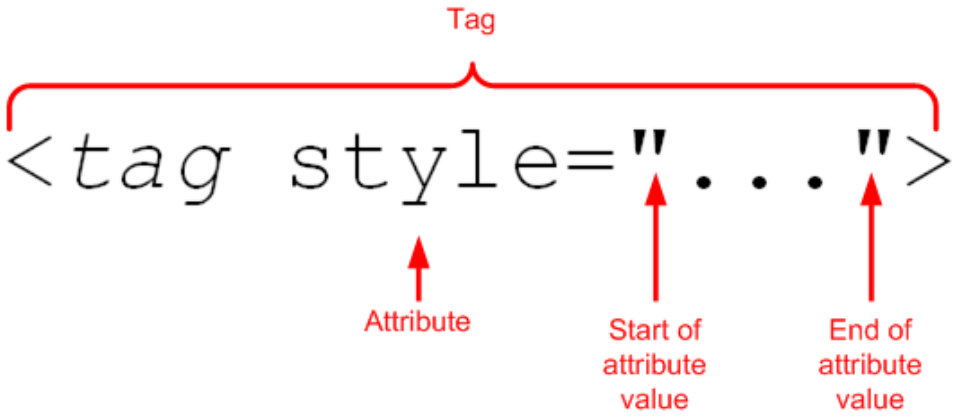
Like all computer languages, CSS follows certain rules of syntax. You basically define a style using this syntax:

```
property:value
```

In this syntax, *property* is any valid CSS property, and *value* is any acceptable value for that property. Notice that you use a colon (:) to separate the property from the value. You can put a space after the colon, if you like. Some developers type a space there to make the code a little easier to read. But that space is optional, never required, and the code behaves exactly the same with or without the space.

There are a couple of places where you can put CSS property:value pairs. In this lesson, we'll focus on the simplest, called the *inline style* where you put the property:value pair right inside the tag of the element you want to style, using a style= attribute. That's quite a mouthful of technical terms, so let's slow down a minute and do a little review.

In HTML, you use tags to define elements. Some tags allow you to use attributes that provide additional information to the tag. The attribute is a word inside the tag, followed by an equal sign and a value enclosed in quotation marks. The attribute we use to style a tag with CSS is the style= attribute. So the general syntax of the tag, with the style= attribute in it, looks like this (there's no actual CSS yet, the example is all HTML still):



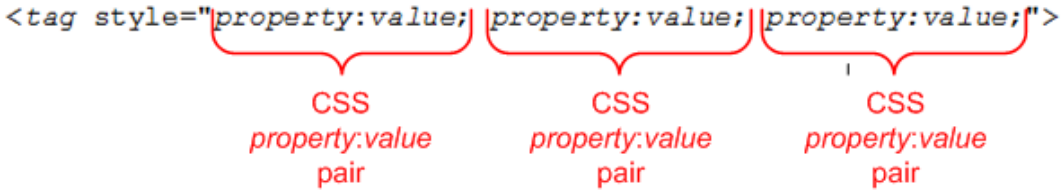

Tag with style= attribute

The *tag* part represents any tag that you may want to apply a style to, like body, h1, h2, p, list, or ul. For a style= attribute, the value between the quotation marks is one or more CSS *property*:*value* pairs. If you want to use just one property:value pair, it goes between those quotation marks like this:


Tag with inline style

As always in syntax charts like the one above, the italic text is just placeholder text. You never type the word *property*, literally. Instead, you type a CSS property there. There are many to choose from, as you'll see shortly. But the *value* is similar. You never type the word *value* there. Rather you type a value that's appropriate for the property you typed. And there are lots of values to choose from too, as you'll see.

You aren't limited to one property:value pair inside the style= attribute. You can use multiple property:value pairs, so long as you separate each pair with a semicolon (;). Take a look at this example:




Tag with inline style that has multiple property:value pairs

You might notice in the example above that there's an extra semicolon at the end of the property:value pairs. That one is optional. The code works the same with or without it. But many people type the semicolon at the end for this reason: If later they decide to add another property:value pair, the semicolon is already there—which makes it less likely that they'll forget to type it at that time.

As with XHTML, the CSS property names and values are always lowercase. Since CSS is entirely separate from HTML, you can't use HTML tags or names as properties or values. You also can't just make up your own property names and values on-the-fly. And it's pretty much impossible to figure them out by guessing!

Your best bet is to learn a few to get started and get some practice with the syntax. And you'll have a chance to do that in a few minutes. In the assignment for this lesson, you'll practice using CSS to center text and pictures in our sample pages.

Let's head over to Chapter 5 and wrap things up.

## Chapter 5

**Summary**

In today's lesson, we covered some historical and current information about HTML and CSS. This is the kind of knowledge that the pros know and assume all of their colleagues also know and understand. To a beginner, it can seem like a whole new world with its own secret language. It's really not necessary to know exact dates, or when different features were first implemented, or anything that specific. But it's good to have a sense of what's been going on in the past, and what's going on now. Here's a quick summary:

- The W3C is the standards organization that defines all Web languages.
- The W3C produced HTML versions 1 through 4 between 1990 and 1999.
- XHTML, a version of HTML that borrows some syntactical rules from the XML language, has been around since the turn of the century.
- HTML5 is the current version of HTML, and CSS3 is the current version of CSS. Virtually all websites being developed today are being written in those versions of the languages.

In the assignment for this lesson, and in future lessons, you'll put your new knowledge to work by going hands-on and adding some CSS to your Web pages. See you there!

## Supplementary Material

### World Wide Web Consortium

http://en.wikipedia.org/wiki/W3C

Click this link for a quick introduction to the who, what, where, when, and why of the W3C.

### XHTML

http://en.wikipedia.org/wiki/XHTML

Check out this site for Wikipedia's encyclopedic description of XHTML and related history.

### HTML5

http://en.wikipedia.org/wiki/HTML5

Here you'll find Wikipedia's article on HTML5.

### Cascading Style Sheets

http://en.wikipedia.org/wiki/Css

You guessed it! Clicking this link takes you to Wikipedia's page on CSS.

## FAQs

**Q:** I'm not sure my index.htm page is current. Can I get a copy of the whole page?

**A:** Sure! Here's the source code for the entire page as things stand at the end of this lesson.

```
<!DOCTYPE html>

<html>

    <head>

    <!-- Title in browser window -->

    <title>My Website</title>

    </head>

        <body style="text-align:center;">

            <!-- Main page title -->

            <h1>Welcome to My Site</h1>

            <!-- Welcome picture -->

            <p><img src="welcome.png" alt="Welcome" /></p>

            <!-- Paragraphs -->

            <p>This is text on my home page. I am a sentence tha
t contains some <strong>boldface</strong> text and some <em>ital
ic</em> text.</p>

            <p>This is a practice paragraph that contains multip
le sentences. A longer paragraph like this will allow me to see
<em>word wrap</em> in action in my Web browser. Word wrap means
that the text will wrap to fit the width of the browser window.
So long as the browser window is a reasonable width, text will n
ot shoot off past the right edge of the window forcing me to scr
oll to the right to see it. The wrapping occurs at spaces betwee
n words at the ends of lines. That prevents any individual word
from being split across two lines.</p>

            <!-- Link to the recipe page -->

            <p><a href="recipe.htm">See My Hot Dog Recipe</a></p
>

        </body>

</html>
```

**Q:** Can I also get all of the source code for the recipe.htm page?

**A:** You bet. Here's recipe.htm in all its glory (through Lesson 6).

```
<!DOCTYPE html>

<html>

    <head>
```

```
    <!-- Title for browser window -->

    <title>My Favorite Recipe</title>

    </head>

        <body>

            <!-- Main title -->

            <h1>Hot Dogs</h1>

            <!-- Hot dog picture -->

            <p><img src="hotdog.png" alt="Hot dog" width="100" /
></p>

            <p>Here is my gourmet hot dog recipe.</p>

            <!-- Subheading -->

            <h2>Ingredients</h2>

            <!-- Unordered (bulleted) list -->

            <ul>

            <li>Hot dogs</li>

            <li>Hot dog buns</li>

            <li>Mustard, relish, chopped onions</li>.

            </ul>

            <!-- Subheading -->

            <h2>Directions</h2>

            <!-- Ordered (numbered) list -->

            <ol>

            <!-- Special character for degrees below -->

            <li>Preheat grill to 350&deg;.</li>

            <li>Place dogs on grill, roll occasionally for even
cooking.</li>

            <li>Place cooked dogs in buns.</li>

            <li>Apply mustard, relish, and onions to taste.</li>

            </ol>

            <p>Eat and Enjoy!</p>

            <!-- A table with single-line borders and right-alig
ned numbers -->

            <table border="1" cellspacing="0" cellpadding="4">

            <!-- Row 1, header cells -->
```

```
            <tr><th>Ingredient</th><th>Calories</th></tr>

            <tr><td>Hot Dog</td><td align="right">150</td></tr>

            <tr><td>Bun</td><td align="right">90</td></tr>

            <tr><td>Ketchup</td><td align="right">10</td></tr>

            </table>

            <!-- Links to other pages -->

            <p><a href="http://www.allrecipes.com">More Great Re
cipes</a> <a href="index.htm">Home</a> </p>

        </body>

</html>
```

## Assignment

The information in today's lesson is important, especially if you're thinking of getting into Web development professionally. But we didn't get to do anything in the way of hands-on. So let's rectify that now. To bring our sample pages into the 21st century, we need to define a doctype at the top of each. And to get a little hands-on practice with CSS, we'll use it to center some text in your home page.

I'm going to suggest that you use the HTML5 document type, since it's the easiest to type. You know that HTML5 is highly experimental and not widely used in production websites, but don't worry. I won't have you use any experimental HTML5 tags inside the page. Inside the pages for this course, we'll stick to the core HTML that's common to XHTML and HTML5, to ensure that everything looks and works right on your screen.

For your index.htm page, I'd like you to add <!DOCTYPE html> as the very first item on that page. Then, inside the <body> tag on that page, please insert an attribute that reads (exactly) style="text-align:center;" after the word *body* but still inside the > at the end of the tag (see image below). Don't remove anything from the page. Just add the new code shown in bold below. Doing this doesn't require any new skills, so if you can do it without peeking at the instructions, great! But if you need help, you'll find some instructions near the end of this assignment.

**Note**

In the image below, the comments starting with <!-- and ending with --> are from the Lesson 5 FAQs. They're not required. Like all comments, they're just notes to you and are completely optional. See the Lesson 6 FAQs for the latest version of both of our sample pages.

```
<!DOCTYPE html>
<html>
  <head>
    <!-- Title in browser window -->
    <title>My Website</title>
  </head>
  <body style="text-align:center;">
    <!-- Main page title -->
    <h1>Welcome to My Site</h1>
```

New code to add to index.htm

After you've made the changes to index.htm and saved those changes, open it in a browser. The text-align:center CSS style will center everything on the page (see image below). Even the paragraph text is centered. That may or may not be what you want. But don't worry about that. You'll learn how to get very precise control over text alignment in the next lesson.


Index.htm in a browser after changes

In recipe.htm, centering everything won't work out too well, because there are too many different kinds of elements on that page. So for now, we won't restyle anything. But just to make sure you're using 21st century programming techniques, please add a <!DOCTYPE html> tag to the top of that one too. Again, don't delete anything from the page. Just add that one tag to the top. If you need help, see the instructions at the end of this assignment.

```
<!DOCTYPE html>
<html>
<head>
  <!-- Title for browser window -->
  <title>My Favorite Recipe</title>
</head>
<body>
  <!-- Main title -->
  <h1>Hot Dogs</h1>
```

Top of recipe.htm in a browser after changes

Close recipe.htm when you've finished, and choose Yes when asked about saving your changes. You don't need to check you work in a Web browser, because we didn't make any stylistic changes to the page. It'll still look the same as it did at the end of Lesson 5. In this assignment, we just brought the source code into the 21st century by adding a doctype declaration to the top of the page.


Back to top



web-1