Creating Web Pages (HTML): Lesson 8

CLOSE

Alan Simpson

Lesson 8: Full-Screen View

This view has opened in a new window and will stretch to fit any screen size (large or small). It displays all of this lesson's components. To return to the normal classroom, please click the "close" button or manually close this window.

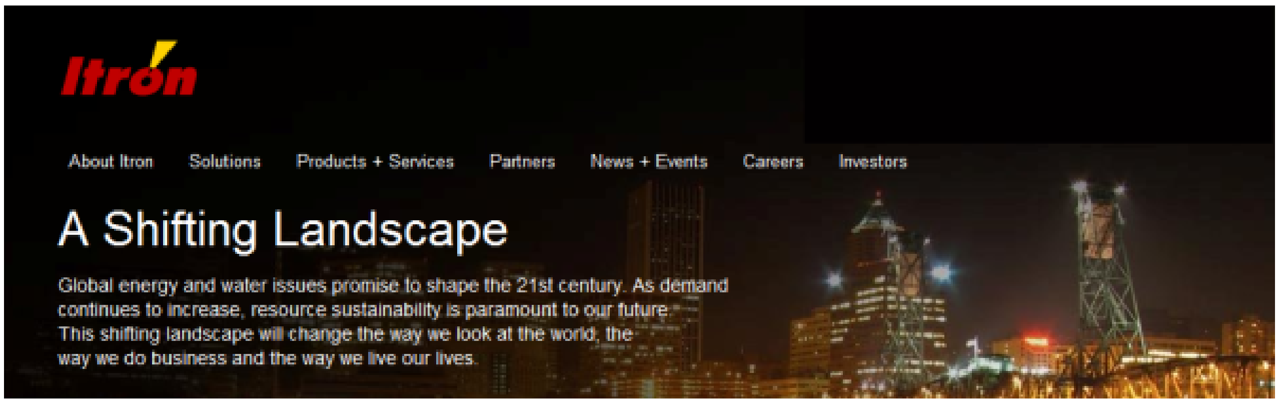
Chapter 1

Introduction

If you've been enjoying this course so far (and I hope you have!), you're really going to have fun with this lesson. Today you're going to learn about *page layout*—what it means and how it's done in Web pages. Many people think page layout is one of the most exciting parts of creating Web pages.

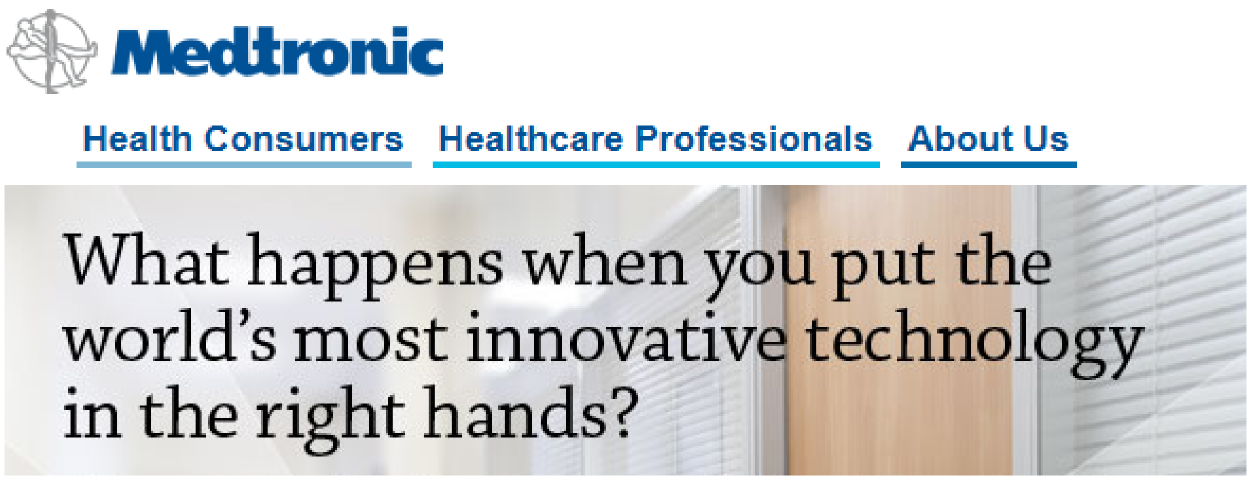
With the index.htm and recipe.htm pages you created, you got some hands-on practice creating different kinds of design elements using HTML tags and CSS. And while they're perfectly acceptable as Web pages, most websites actually use a *layout* that makes it easy to recognize what site you're in—and easy to get from one page to the next within the site (assuming the site has more than one page).

As examples of what I mean by layout, take a look at the following sample pages. (They're just snapshots, not actual pages, so clicking on them won't take you to another page.)

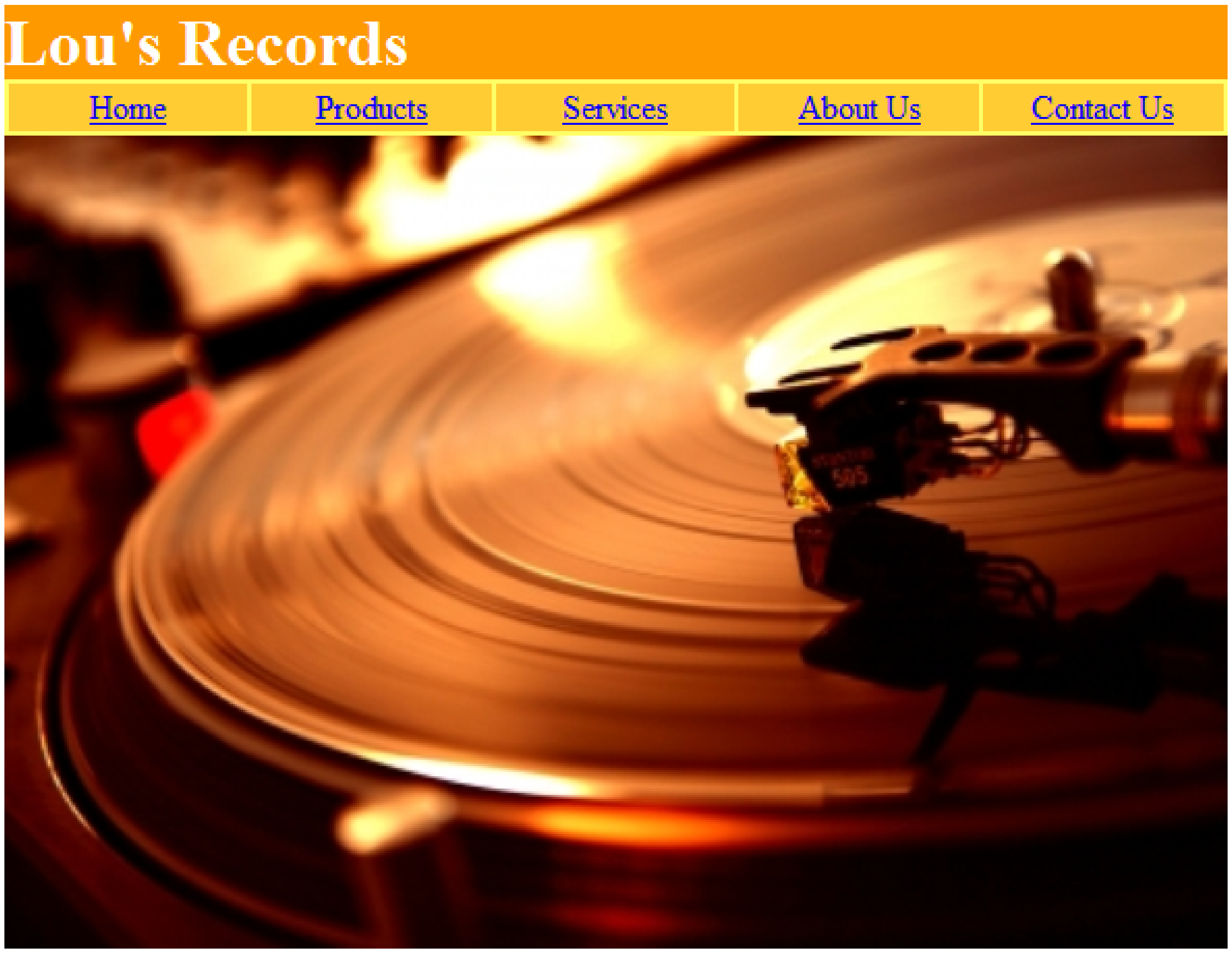


Global energy and water issues promise to shape the 21st century. As demand continues to increase, resource sustainability is paramount to our future. This shifting landscape will change the way we look at the world, the way we do business and the way we live our lives.

First sample Web page

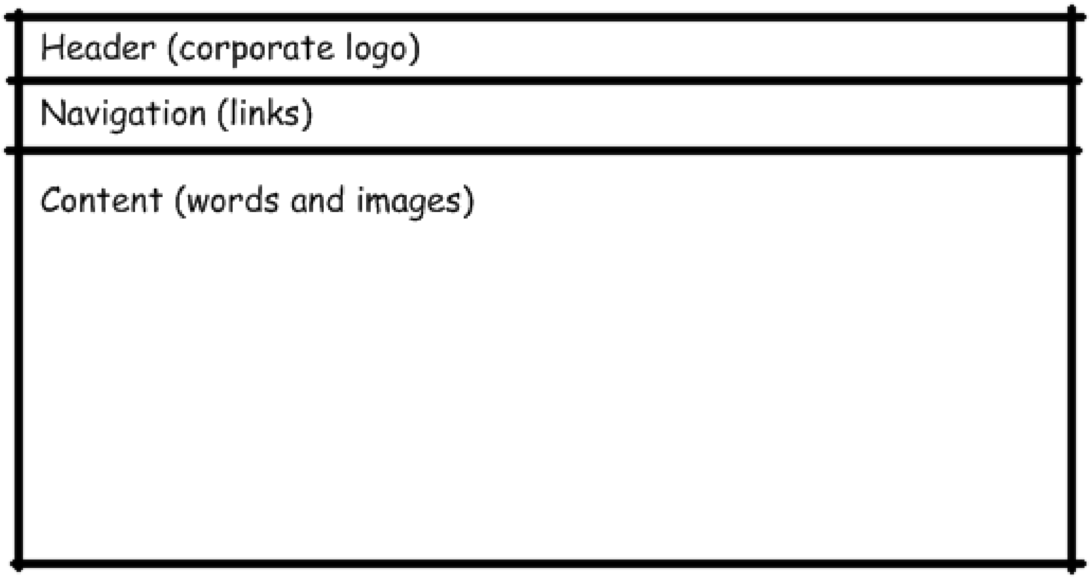

Medtronic
Health Consumers
Healthcare Professionals
About Us
What happens when you put the world's most innovative technology in the right hands?

Second sample Web page



Third sample Web page

Even though the content (specific words and pictures) on each page is different, the layout of each page is similar. If you remove the specific words and pictures, that layout looks like this.



Sample layout

As you can see, there are three sections (also called *page divisions*). The top section, or division, is usually referred to as the *header* or *branding bar.* Studies have shown that the first place most people look when viewing a page is the top left corner. And so the header (or branding bar) often contains brand-name information about the site, such as a corporate logo.

If the site contains multiple pages, the *navigation bar* section comes next, under the header, and it contains links to other pages within the site.

The third section down is sometimes called the *main content* or *content* area. And that section can contain anything—text, pictures, lists, tables—whatever it takes to best convey the information you're trying to convey.

To define each page division, you'll use <div>...</div> tags, where *div* is short for *page division*. So let's mosey over to Chapter 2 now and get started creating a page layout with div tags!

Chapter 2

Create a Page Layout

Modern Web pages are often organized into a page layout that consists of multiple sections or page divisions. In HTML, we use these two simple tags to define each page division:

```
<div>


</div>
```

The <div> tag marks the top or start of a division, and </div> marks the end of the division.

Since any page might contain multiple divisions, we often use multiple pairs of div tags in a page. To give each one a unique identity that we can apply styling to, we assign a name using the id= attribute, followed by the name enclosed in quotation marks. The name can be anything you like, so long as it starts with a letter and doesn't contain any blank spaces. Even though you have some flexibility in choosing those id names, professional developers don't usually make up totally random names like *goober* or *wingnut* or *div1*, as those names don't convey any useful information about the purpose of the division. It's better to come up with names that provide some information about what the div contains.

For example, the first division on the top of the page is sometimes referred to as the *page header*, and so the div is also often named *header* as below.

```
<div id="header">


</div><!-- End header -->
```

As in all tag pairs, the opening tag (<div>) starts the element. The closing tag (</div>) marks the end of the element. So any tags between the <div> and </div> tag make up the content for the header division.

The comment after the closing </div> tag is optional. As we've discussed previously, comments are always optional, because they're just notes to you. But putting a closing comment after the closing /div> tag helps to remind you which <div> tag it is closing, and this can come in handy later when there's a lot of code and content between those tags.

Tip

If you use comments, make sure you type them correctly. The comment must start with <!-- (no spaces) and end with --> (no spaces).

The next page division under the header is often the navigation bar, or *navbar* for short. Its name is often *navbar* or *nav* or something similar, and its tags might look like this:

```
<div id="nav">


</div><!-- End nav -->
```

In between those tags, you'd put the code and content for a navigation bar. You'll see exactly how a little later in this lesson. But for now, let's stay focused on the divisions. As always, the comment after the closing tag is optional, but it's a good idea, because it makes it easy to see which division that </div> tag is closing.

Note

HTML5 has <header> and <nav> tags that can be used in place of divs. But HTML5 is still very much in the early experimental stages and doesn't work in all current browsers. So many Web developers continue to use named divs, like those above.

After the navigation bar comes the main content area. Its name is often *main* or *content*, and its tags look like this:

```
<div id="main">


</div><!-- End main -->
```

So basically, we use a pair of div tags to define each major page division in the layout. We assign an id name to each division, because (as you'll see shortly), that allows us to create a CSS style rule for each one.

Note that these new tags don't replace or undo anything you've already learned in this course. Every page that you create still requires all of the mandatory tags you've learned about in previous lessons, including the html, head, title, and body tags.

Let's go ahead and create a page that contains all of the mandatory tags as well as tags for three page divisions. We'll start with the code and a little temporary placeholder text. Follow these steps:

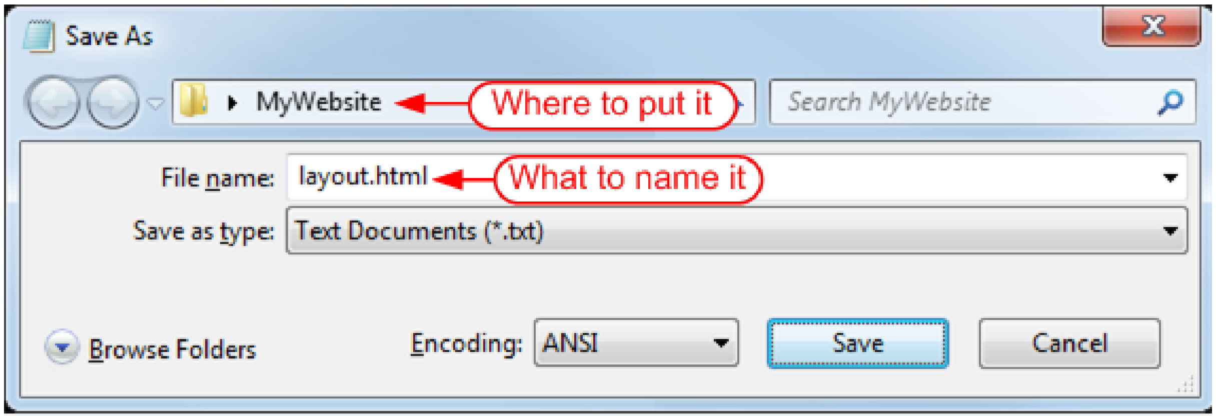
1. Open the program you normally use to create and edit Web pages. (If you've been following along in this course without any prior experience, that's probably Notepad in Windows or TextEdit on a Mac).

2. Type (or copy and paste) all of the following code into the page:

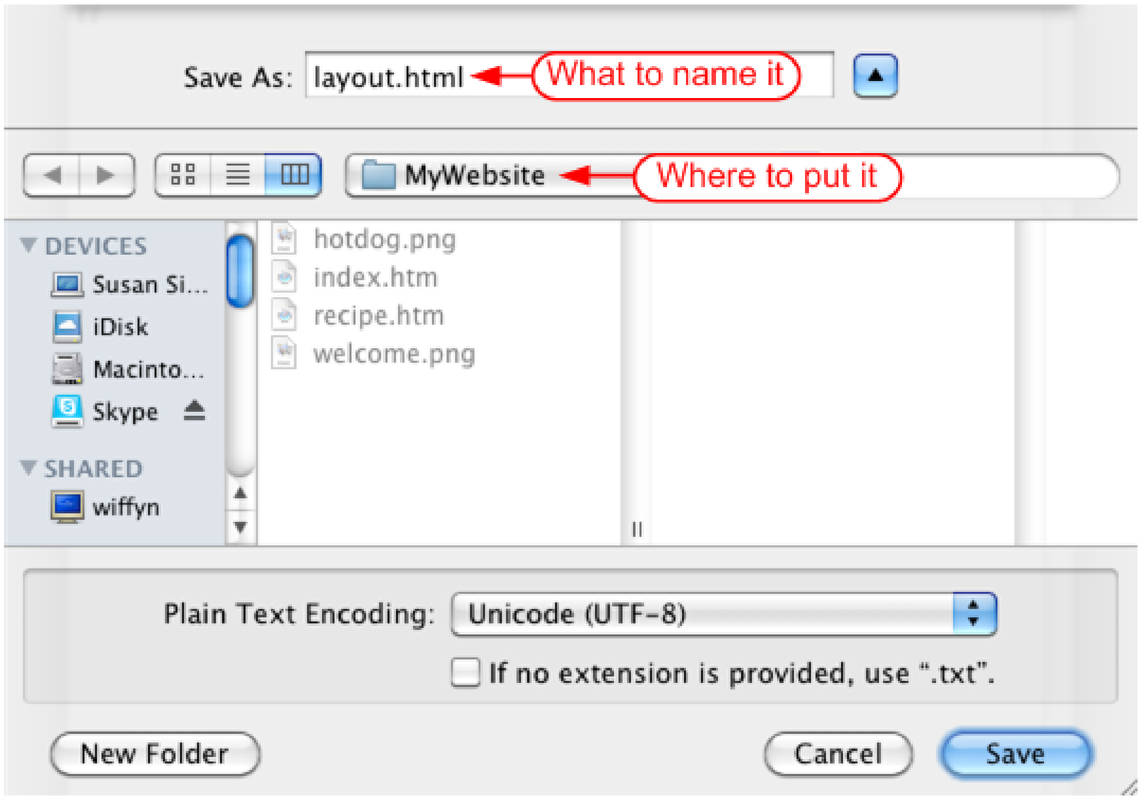
```
<!DOCTYPE html>

<html>

<head>

    <title>My Layout</title>

</head>

<body>

   <!-- Header division -->

   <div id="header">

       Header content goes here

   </div><!-- End header -->

   <!-- Navbar division -->

   <div id="nav">

      Links go here.

   </div><!-- End nav -->

   <!-- Main content division -->

   <div id="main">

      Main content goes here.

   </div><!-- End main -->

</body>

</html>
```

3. Close the editor, and choose **Save** (or **Yes**) when asked about saving your changes.

4. As always, save it in your MyWebsite folder, and name this one *layout.html,* as in the Windows or Mac Save As dialog shown below.



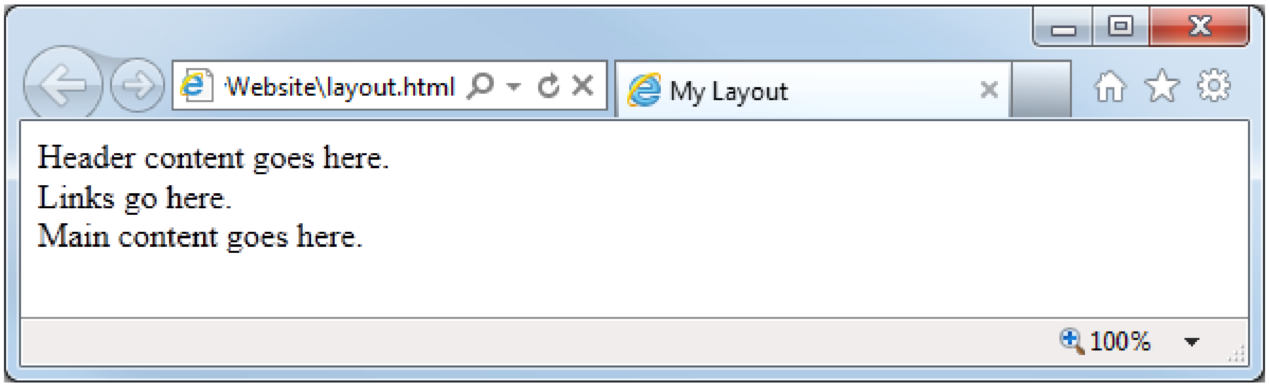
File name: layout.html
Save as type: Text Documents (*.txt)
Encoding: ANSI



Save As: layout.html
Plain Text Encoding: Unicode (UTF-8)
If no extension is provided, use ".txt".

Saving layout.html

4. Once you've closed and saved the page, double-click its filename (**layout.html**) to have a look in a browser. It'll look something like this:



Header content goes here.
Links go here.
Main content goes here.

The layout.html in a Web page

You might be wondering what kind of nincompoop would go to the trouble of typing all that code to end up with three lines of plain, simple text in a Web browser window. But that's just temporary placeholder text so you can see *something* in the browser to check your work. What you see in the browser right now isn't the whole story. What's important is that in your code you've three named divs, each of which defines a major page division in a layout. And that's a great starting point for designing Web pages the way the well-paid pros do.

Creating the Style Rules

The next step is to start *styling* the divisions. Styling has to do with colors and such, and it's typically done with CSS style rules in the professional world of Web design. As you may recall from Lesson 7, a style rule is a set of CSS *property*:*value* pairs that define the style of an element. There's a *selector* that describes the element that the style applies to, followed by the *property*:*value* pairs that describe the style.

Earlier, you saw style rules that apply to different types of html elements. For example, the body{} style rule applies to the body element, which is defined by <body>...</body> tags. And the h1{} style rule applies to h1 headings (any text between <h1>...</h1> tags).

With named divs, the style rule selectors work a little differently. You *don't* want to use *div* as the style rule selector, because that style rule would apply to all divs on the page—not to just one particular page division. When you want a style rule to apply to a specific named element, you need to use the name, preceded by a # sign, as the selector. For example, this style rule applies only to the element named *header*:

```
#header {

}
```

Or in other words, it applies only to the element whose opening tag contains id="header". So any *property*:*value* pairs you put in that style rule apply only to text between these tags in the page:

```
<div id="header">

</div><!-- End header -->
```

Let's examine another example. Take a look at this style rule:

```
#nav {

}
```

The #nav means *apply this style to the element with id="nav" in its opening tag*. So that style rule applies to anything between these tags on the page:

```
<div id="nav">

</div><!-- End nav -->
```

So can you figure out what a style rule like the one below would apply to?

```
#main {

}
```

If you guessed that it applies to any text and content between these tags below, then you guessed right:

```
<div id="main">

</div><!-- End main -->
```

Why? Because the # sign in front of *main* means that it applies to *the element named main*, or in other words, *the element with id="main" in its opening tag*.

Before you try it out, you need to remember a few more things from Lesson 7. For starters, you can't just stick CSS style rules any place you want in the page. You have to put them in an internal style sheet. And that internal style sheet goes between the <head>...</head> tags. Also, that internal style sheet needs to be enclosed in <style type="text/css">...</style> tags, so the Web browser "knows" there's CSS code there.

Note

You can also put style rules in an external style sheet, where they're accessible to multiple pages in a site and can be shared across pages in the site. But in the interest of keeping things simple this early in the learning process, we'll stick with an internal style sheet so you don't have to juggle multiple files.

It's a lot to remember, especially when you're first getting started. But it gets easier with practice. And to get that practice, let's move on to Chapter 3 where you'll create an internal style sheet for our new page layout.

Chapter 3

Styling the Divisions

In Chapter 2, we created a page layout by putting some named page divisions in a Web page. But we only included some styled divisions and temporary placeholder text so far, so it doesn't really look special when we view the Web page in a browser. In this chapter, we're going to take the next step and start styling those page divisions by adding an internal style sheet and some style rules.

Let's get started! Follow these steps:

1. If layout.html is already open in your editor, switch to that window. (Or right-click or CTRL + click **layout.html** in your MyWebsite folder, and choose **Open With**, and then your editor).

   Tip

   If you can't find your editor on the Open With menu, you can just rename your file to layout.htm. Or see the Lesson 8 FAQs for tips on configuring your operating system to add that editor to the menu.

2. Move the cursor just past the </title> tag near the top of the page, and press ENTER to start a new line after that one.

3. Type *<style type="text/css">* (which is the tag to start the internal style sheet), and then press ENTER a couple of times.

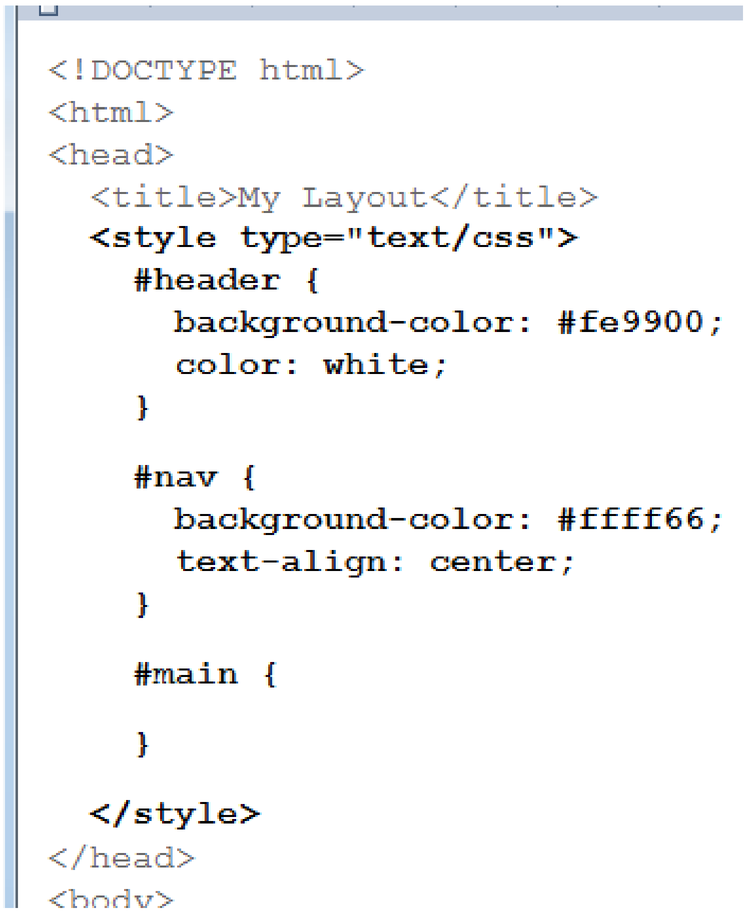
4. Type *</style>* (the closing tag for the internal style sheet), just so you don't forget to type it later.

5. Move the cursor so it's inside the internal style sheet (between the <style type="text/css"> and </style> tag).

6. Now, very carefully type all of the following code, pressing ENTER at the end of each line. Don't worry about indenting—that's not important. But what you type matters a lot, especially spelling and punctuation.

```
#header {
    background-color: #fe9900;
    color: white;
}

#nav {
    background-color: #ffff66;
    text-align: center;
}

#main {
}
```

The image below shows the new code typed into the page. As always, I'm showing the new code that you just typed in bold. I'm only showing the section of the page where you added the new code (not the whole page). And as always, indents and blank lines are optional. You can indent, if you like, to make your code look better organized. But it's not required and has no effect at all on how the page looks to a person who's viewing it in a Web browser.



```
<!DOCTYPE html>
<html>
<head>
  <title>My Layout</title>
  <style type="text/css">
    #header {
      background-color: #fe9900;
      color: white;
    }

    #nav {
      background-color: #ffff66;
      text-align: center;
    }

    #main {

    }

  </style>
</head>
<body>
```

Three style rules added to the internal style sheet

The first style rule we added applies an orange color (#fe9900) to the background of the div named *header*, and it sets the text color inside that div to white.

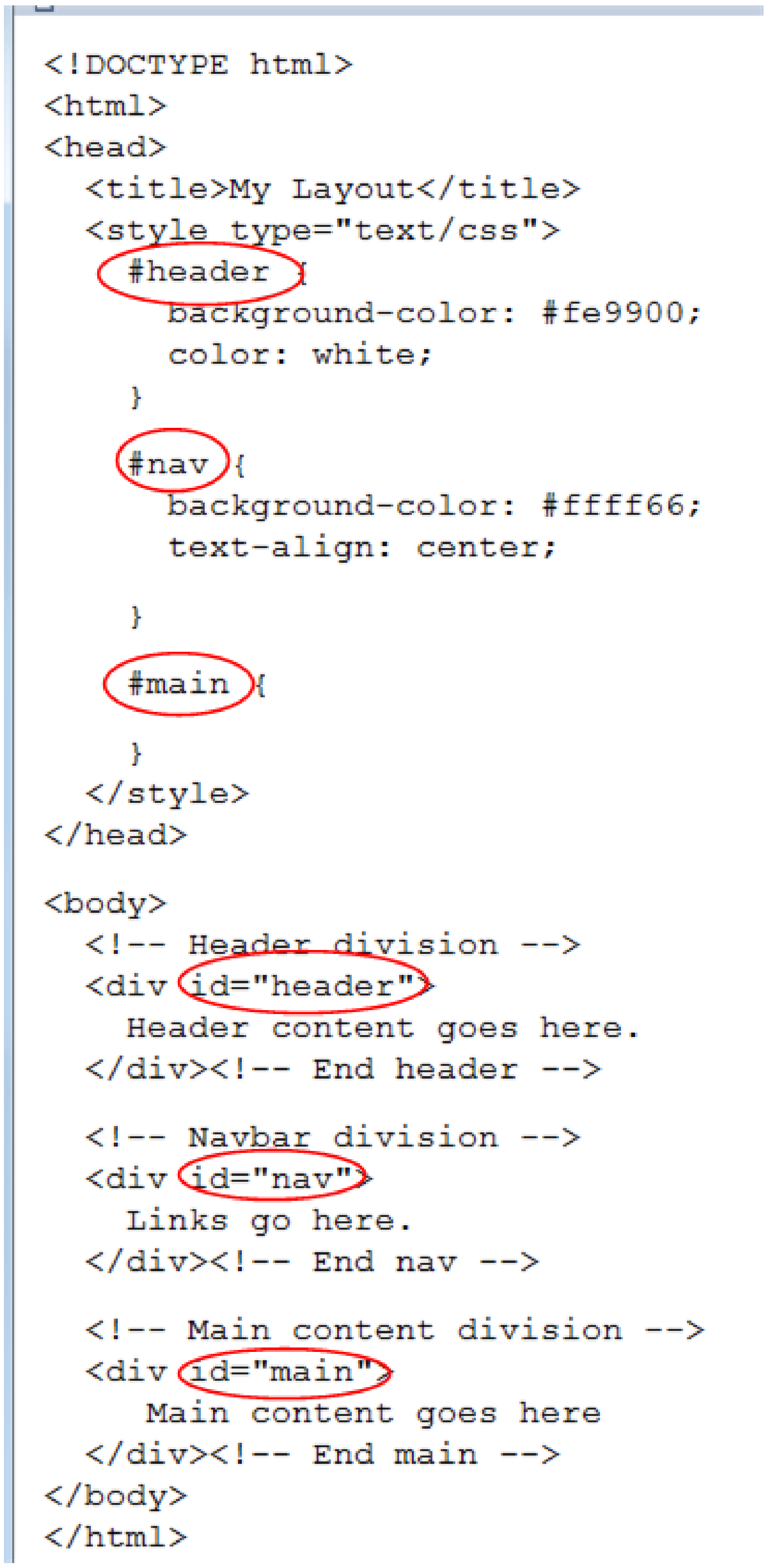
The second style rule applies a yellow shade (#ffff66) to the div named *nav*, and it centers text inside that div.

Remember

Nobody knows or memorizes all the color hex codes. You'll want to add some sites like the ones from the Lesson 7 Supplementary Material to your browser's bookmarks or favorites. This way, it'll be easy to find colors you like!

The third style rule is for the div named *main*. We haven't put any *property*:*value* pairs in that one yet, so it won't have any impact on the style of that div. But the style rule is in place, and you can add *property*:*value* pairs at any time in the future. Right now, it's not doing any harm just sitting there.

Before we move on, please take a close look at the image below to reinforce what you're learning here. It's important to understand that each style rule corresponds to a specific page division. The # at the start of each style rule says "this style rule applies to the element with the id...," and then the id name follows that # character. So the #header{} style rule applies to the div with id="header" in its tag; the #nav{} style rule applies to the div with id="nav" in its name; and #main{} applies to the div with id="main" in its name. Right now the #main{} style rule is just a placeholder, in case we decide to style the main div later.



```
<!DOCTYPE html>
<html>
<head>
  <title>My Layout</title>
  <style type="text/css">
    #header {
      background-color: #fe9900;
      color: white;
    }

    #nav {
      background-color: #ffff66;
      text-align: center;

    }

    #main {

    }
  </style>
</head>

<body>
  <!-- Header division -->
  <div id="header">
    Header content goes here.
  </div><!-- End header -->

  <!-- Navbar division -->
  <div id="nav">
    Links go here.
  </div><!-- End nav -->

  <!-- Main content division -->
  <div id="main">
     Main content goes here
  </div><!-- End main -->
</body>
</html>
```

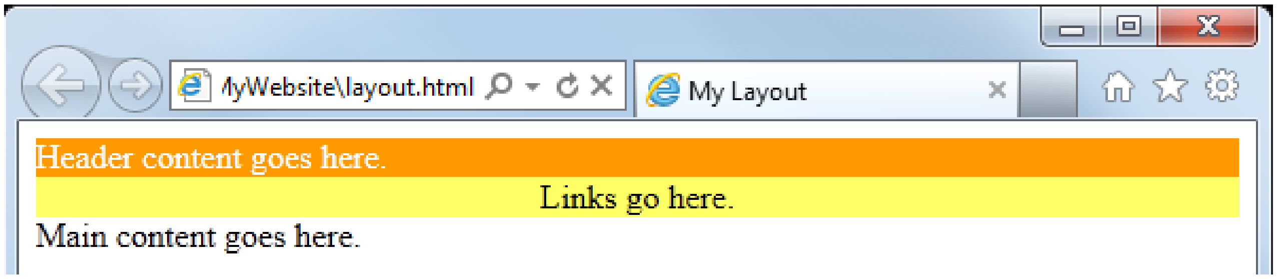
Layout.html with important items circled

Let's have a look in the browser now:

1. Save the changes to the page as appropriate for your editor. (Choose **File > Save** from the menus. Or press CTRL + S in Windows. Press COMMAND + S on a Mac.)

2. Open **layout.html** in the browser. (Or, if layout.html is already open in a browser, click the browser's **Refresh** or **Reload** button.)

If you typed all of the code correctly, you should see the background colors applied to the first two page divisions as below. The text in the header division up top will be white, and the text in the nav bar division will be centered. We haven't styled the main div yet, so it'll still just be the standard black, left-aligned text against a white background.



Header content goes here.

Links go here.

Main content goes here.

A little styling applied to our layout

If yours looks a lot different, it's probably one or more typographical errors in your code. Here are a few common mistakes:

- Forgetting a curly brace
- Using parentheses or square brackets instead of curly braces
- Typing a colon where you need a semicolon (or vice versa)
- Forgetting to type # characters where needed
- Typing five-digit hex color codes instead of six-digit codes (#fff66 instead of #ffff66)

Once you get your page looking like it should in the browser, meet me in Chapter 4 where we'll discuss how to turn a page like layout.html into a real, multipage website that contains whatever you want it to contain!

Chapter 4

Creating a Navigation Bar

Our layout still isn't too impressive yet, because at the moment, it only contains some generic placeholder text. It doesn't really have anything in the way of content. By *content*, I mean the words and pictures that the site is about. And those can be anything your heart desires. You can use any tags and any CSS *property*:*value* pairs you'd like to define and style your content. But you do want to think in terms of page divisions. You don't want to just start throwing stuff into the page at random.

For starters, I'd like to skip the page header and focus on the navigation area, as that's the part that's likely to take the most planning. The idea behind that page division is to present links that lead to other pages within the same site. Of course, a website can have only one page—in which case, there'd be no need for links or a navigation section to put the links into. But since most sites contain multiple pages, you'll often see such links.

To create a navigation bar for your own site, you first have to make a couple of decisions:

- How many pages do I want in my site?
- What will I name those pages?

You don't need to make those decisions right this moment. For the rest of this chapter, I'll provide some general guidelines on how to create a navbar for a site, and how to name the site's pages.

Before we get started, I want to introduce you to the CSS width: property. As its name implies, that property allows you to define how wide you want an element to be. You can specify the width in pixels (using px) or in percent (using

%). The % is good when you want to create equal-width columns, because you just divide 100 by the number of items you want included it the width to determine the percent width of each column. For example, if you were to have five items, each would have a width of 20% (100 divided 5). If you wanted four items, each would have a width of 25% (100 divided by 4). If you have three items, each would have a width of 33.3%. (You can't use fractions like 1/3 or 1/2 in CSS or most other programming languages. You have to use a decimal point.)

The navigation bar on most Web pages contains links to other pages within the site. There's no rule that says your site *must* have multiple pages though. So that division is optional. But for the sake of example, I'll show you how to create a relatively simple, nice-looking navigation bar for a site with five pages in it. I'll use a table cell for each link. To make them equal in width, I'll make each of the five cells 20% the width of the table, so the widths total 100%. Follow these steps to try it out:

1. Open layout.html in your editor (or just switch to that window if it's still open from the previous chapter).

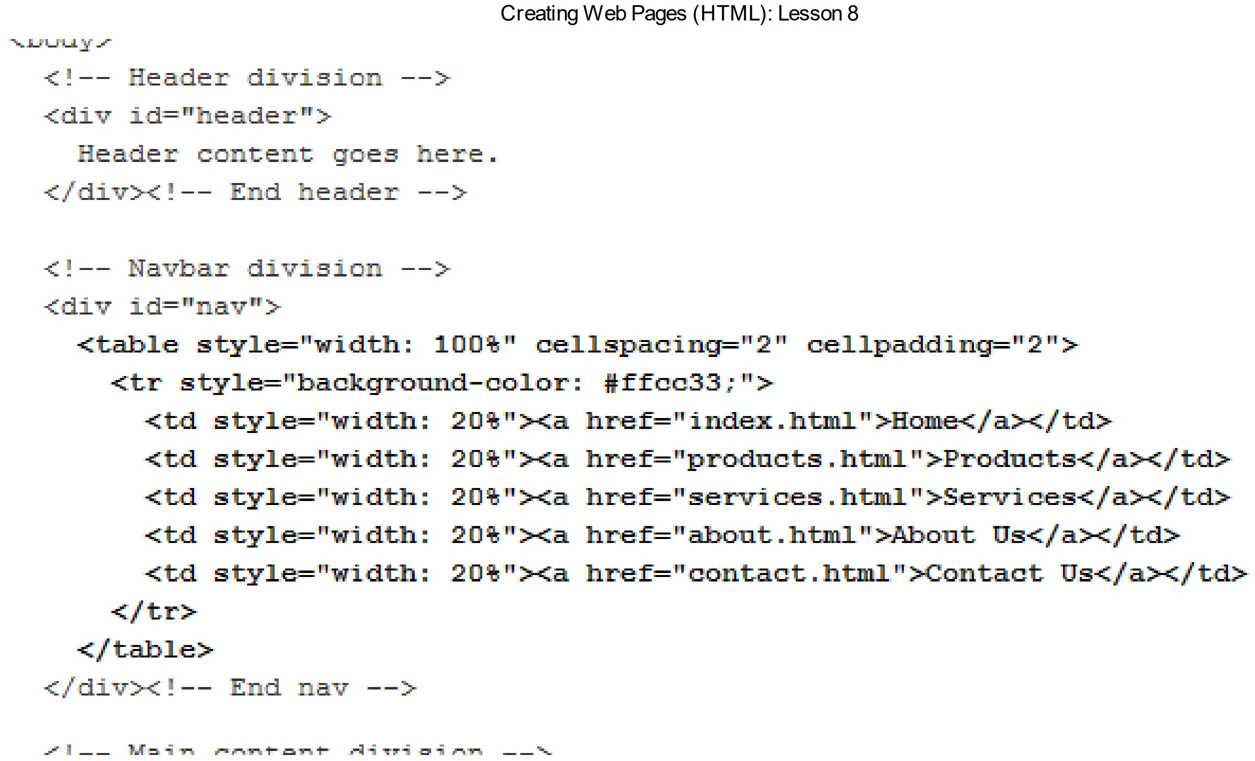
2. Click inside the navigation bar division (between the tags <div id="nav"> and </div><!-- End nav -->).

3. Delete the temporary placeholder text *Links go here*.

4. Type (or copy and paste) all of the following tags and text. As always, you don't need to worry about indenting. But the code will be a little easier to read of you press ENTER after each line as shown here:

```
<table style="width: 100%" cellspacing="2" cellpadding="2">

  <tr style="background-color: #ffcc33;">

    <td style="width: 20%"><a href="index.html">Home</a></td>

    <td style="width: 20%"><a href="products.html">Products</a></td>

    <td style="width: 20%"><a href="services.html">Services</a></td>

    <td style="width: 20%"><a href="about.html">About Us</a></td>

    <td style="width: 20%"><a href="contact.html">Contact Us</a></td>

  </tr>

</table>
```

The image below shows all of the new code in place, darkened against the code that was already in the page.



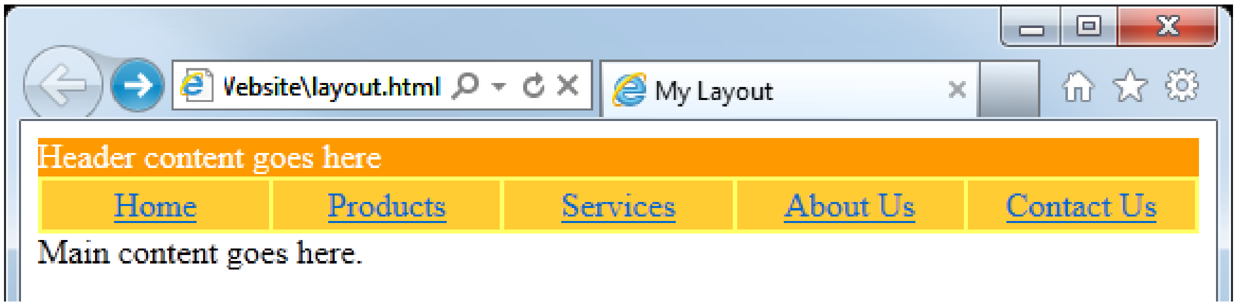
```
  <body>
    <!-- Header division -->
    <div id="header">
      Header content goes here.
    </div><!-- End header -->

    <!-- Navbar division -->
    <div id="nav">
      <table style="width: 100%" cellspacing="2" cellpadding="2">
        <tr style="background-color: #ffcc33;">
          <td style="width: 20%"><a href="index.html">Home</a></td>
          <td style="width: 20%"><a href="products.html">Products</a></td>
          <td style="width: 20%"><a href="services.html">Services</a></td>
          <td style="width: 20%"><a href="about.html">About Us</a></td>
          <td style="width: 20%"><a href="contact.html">Contact Us</a></td>
        </tr>
      </table>
    </div><!-- End nav -->

    <!-- Main content division -->
```

New code in nav div

5. Save the page after typing in the new code.

6. View (or Refresh/Reload) the page in a browser.

If you typed all of the code correctly, the navigation bar division should look like the example below with the Home, Products, Services, About Us, and Contact Us links. Note that clicking a link will just cause an error, because the links point to pages that don't exist yet. That's not a mistake or anything to worry about. We'll create the pages to make the links work a little later. For now, just keep reading!



Header content goes here

Home	Products	Services	About Us	Contact Us

Main content goes here.

Navigation bar in layout.html

A Look at the Code

Let's take a moment to discuss the code. And if you made any mistakes in terms of how your navigation bar looks on your page, this discussion may help you locate and fix your errors.

We used a table to contain the navigation bar. (You learned about tables in Lesson 5.) The table starts with <table> and ends with </table>. The opening <table> tag looks like this:

```
<table style="width: 100%" cellspacing="2" cellpadding="2">
```

The *width:100%* ensures that the table is the full width of the browser window. While not a technical requirement, it does give the navigation bar a nice look and makes it easy to make equal-sized cells for links across the table. The cellspacing and cellpadding attributes add a couple pixels of empty space between and inside the table cells.

The <tr>...</tr> tags define a single table row. In the opening <tr> tag, we use an inline style to set the row's background color to #ffcc33 which is the orange color you see on each button. The lighter yellow color between table cells is the navbar division's background color. We defined this in the #nav style rule by setting its background color to #ffff66.

Each link is contained in a table cell that is 20% the width of the table. The opening tag for each cell is *<td style="width: 20%">*, and the closing tag is *</td>*. I chose 20% because there are five cells, and I want their widths to total 100% to fill the width of the table.

Inside each table cell is a link. You learned about links in Lesson 3. Right now, they are somewhat hypothetical, because most refer to a page that doesn't exist yet in your site. For example, this link (below) refers to a page named *products.html* that we never created:

```
<a href="products.html">Products</a>
```

So when you click the link text (the underlined word *Products*) in the Web browser, you get an error message. So why would one create a link to a page that doesn't exist, knowing it will cause an error when someone clicks the link? Well, for one thing, we don't have to worry about anyone (except ourselves) clicking the link right now, because the site hasn't been published to the Web yet. And for another, after we finish up the layout.html page, we can use it as a template for all of the pages in the site. That way, rather than painstakingly typing each page from scratch, we can start each page using code that's in the layout page. And that can save a lot of typing, a lot of testing, and a lot of finding and correcting errors!

A Look at the Page Names

Even though those pages don't exist, you'll notice I used relatively short, simple filenames like *index.html*, *products.html, services.html, about.html,* and *contact.html.* You're free to come up with whatever pages you want for your site, and you can name them pretty much whatever you want. But remember, every Web page must have an .htm or .html extension. There's no difference and no advantage or disadvantage to either one, so feel free to use whichever extension you like.

Remember too that even though your particular operating system probably allows for long file names with spaces, Web development isn't just about your computer. It's about all computers—everywhere in the world! And they don't all have the same operating system as yours. So when naming your files, it's best to stick to short, simple names with no blank spaces. And since there are operating systems in the world that are case-sensitive to filenames, it might be a good idea to define and stick to some simple rules, like only using lowercase letters in your file names.

For now, that's enough to get you started with page layouts. Meet me in Chapter 5, and we'll summarize what you've learned today.

Chapter 5

Conclusion

You learned a lot about page layout today! We started out discussing div tags and how we can use these tags to organize a website's pages into major divisions or sections. Then, we looked at creating style rules for your divisions, so you can style them exactly the way you want to. Let's review what we talked about.

To define a division in a Web page, we use <div>...</div> tags. Because a page usually consists of multiple divisions, we often assign each division a unique identity (name) using an id= attribute. The name you give to a division can be any name you like, so long as it starts with a letter and doesn't contain any blank spaces.

Professional developers usually name their divisions to match their purpose or role in the page. For example, the first division is often named heading using <div id="header"> as the tag. The next division, which often contains links to all of the pages within the site, is called the *navigation bar*. And the div tag for that one might be <div id="nav"> or something similar. Then, the main content for the page comes next and is often enclosed in a division that starts with <div id="main">.

The main reason for giving each division a name is so that you can create a style rule for each, which allows you to define its coloring and other stylistic features. Normally, the selector for a style rule is the tag that you want the style rule to apply to. For instance, when creating a style rule for h1 elements, you just use h1 as the selector. When you want to associate a style rule with an id name rather than an element type, you have to use a # sign in front of the name to indicate an id. So the selector for an element that contains *id="header"* is *#header*. The selector for an element that contains *id="nav"* is *#nav*, and the selector for an element that contains *id="main"* is *#main*.

Excellent work today. In this lesson, you got some practice doing layouts the way the pros do it. It's a start, but we're not done yet. In the assignment for this lesson, you'll replace the placeholder text in the header div with a business name. Then, we'll continue our work on layout.html in Lesson 9. See you there!

Supplementary Material

The Div Tag for Page Divisions

http://www.debbietdesigns.com/learn/32/xhtml-div-tag/

Click this link for a quick tutorial on using div tags for page divisions.

Using <div> Tags for Layout

http://www.beginnersguidetohtml.com/guides/css/layout/div-tags

Here you'll find another page about using div tags for page layout.

CSS Basics

http://www.debbietdesigns.com/learn/34/css-basics/

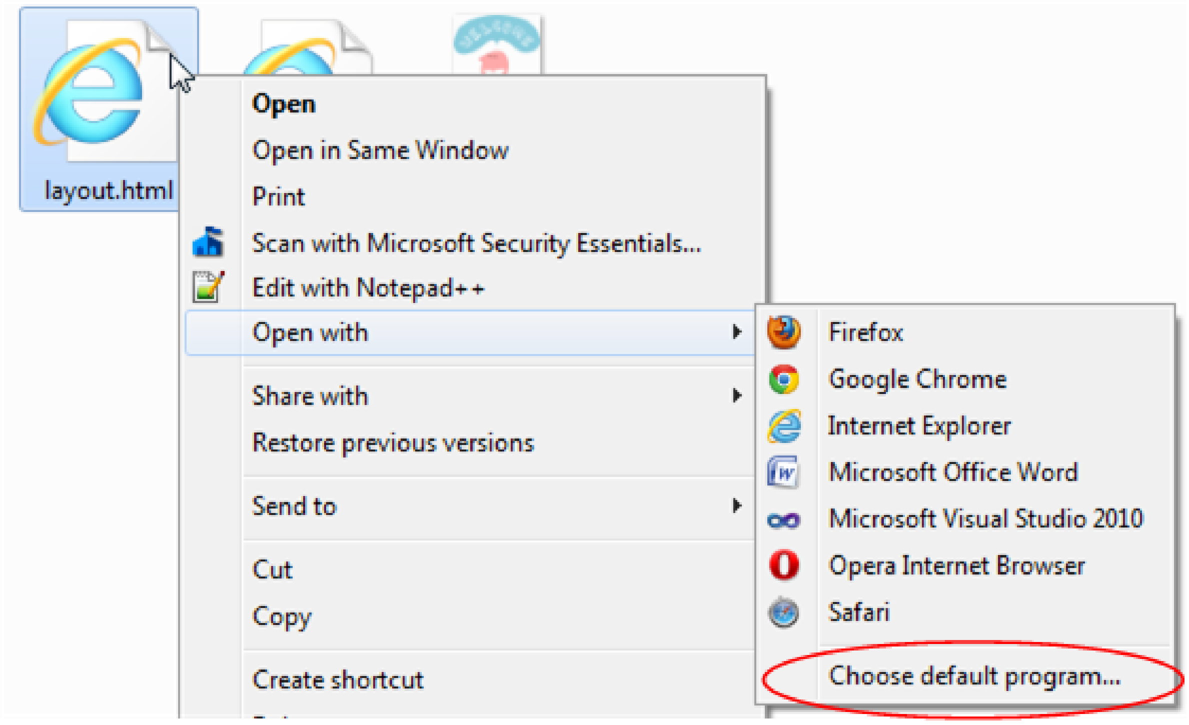
This page provides a gentle tutorial on CSS for beginners.

FAQs

Q: When I right-click layout.html and choose Open With, I don't see Notepad as an option. What gives?

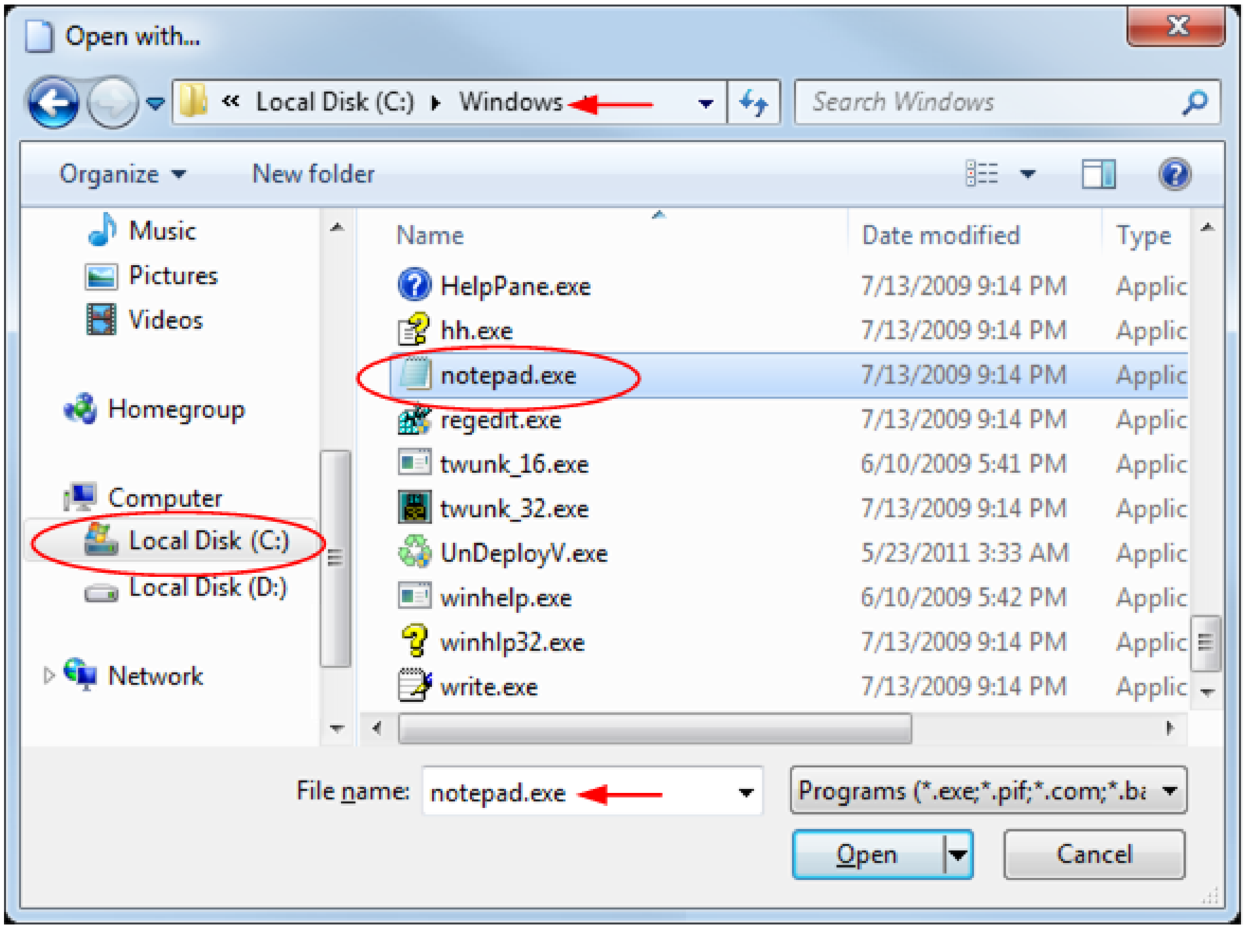
A: Most operating systems will associate the .html extension with the default text editor by default. But if yours isn't configured that way, it's nothing to worry about. You can use .htm rather than .html as the filename extension for all your web pages, and just leave it at that. Or you can change your operating system to include your favorite text editor as an option for opening .html files. If you want to do the latter, make sure you have at least one file with a .html extension (like layout.html) to work with. Then you can add Notepad as a program for opening that file type. Here are the steps using Windows 7 as an example, though the steps are similar for other versions of Windows:

1. In the folder that contains that file, right-click the icon for the .html file, choose Open With, then click Choose Default Program from the bottom of the menu.



Right-click a .html file, choose Open With

2. In the Open With dialog box that opens, look for Notepad. If you see Notepad listed, skip to Step 8 now. If you don't see Notepad listed, continue to Step 3 below.
3. Click **Browse**.
4. In the navigation pane at left, click the icon that represents your Windows drive (typically **Local Disc (C:)** and showing a Windows logo). If necessary, you can click the triangle next to **Computer** to expand that and see icons for all your computer's drives.
5. In the content pane to the right double click the **Windows** folder icon to open the Windows folder.
6. Scroll down and click on **notepad.exe** once to select it. The Address bar up top should show your hard drive and the Windows folder name, the File name box should show notepad.exe as the name of the selected file in that folder.



Name	Date modified	Type
HelpPane.exe	7/13/2009 9:14 PM	Applic
hh.exe	7/13/2009 9:14 PM	Applic
notepad.exe	7/13/2009 9:14 PM	Applic
regedit.exe	7/13/2009 9:14 PM	Applic
twunk_16.exe	6/10/2009 5:41 PM	Applic
twunk_32.exe	7/13/2009 9:14 PM	Applic
UnDeployV.exe	5/23/2011 3:33 AM	Applic
winhelp.exe	6/10/2009 5:42 PM	Applic
winhlp32.exe	7/13/2009 9:14 PM	Applic
write.exe	7/13/2009 9:14 PM	Applic

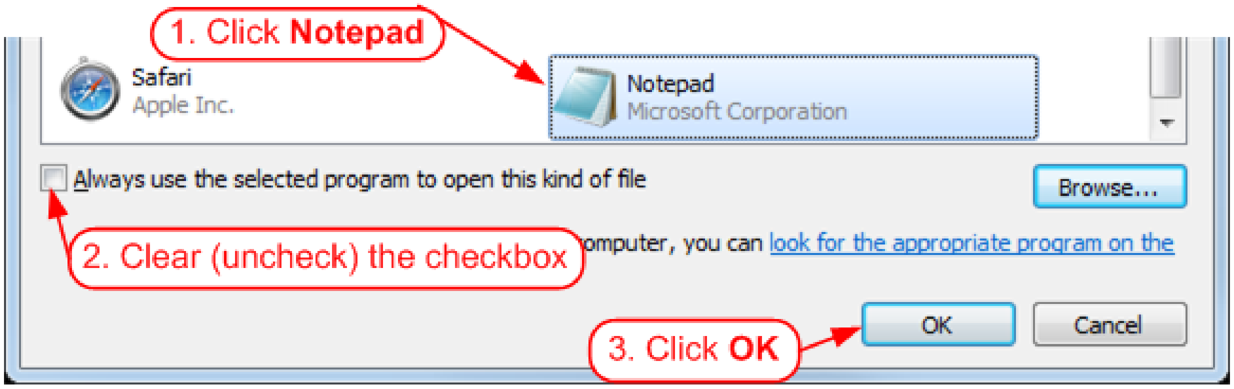
File name: notepad.exe

Programs (*.exe;*.pif;*.com;*.ba

notepad.exe in Windows folder selected

7. Click **Open**.

8. Click **Notepad** to select it, and then uncheck the checkbox for the options that reads "Always use the selected file to open this kind of file", then click **OK**.



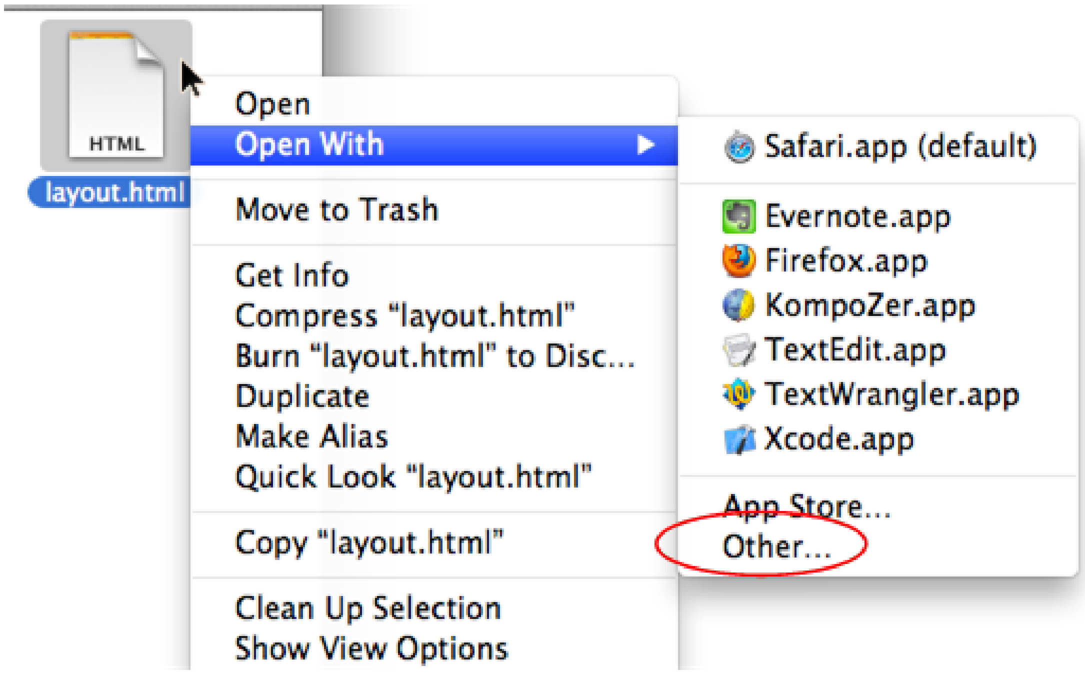
Always use the selected program to open this kind of file

omputer, you can look for the appropriate program on the

Notepad an option on the Open With menu

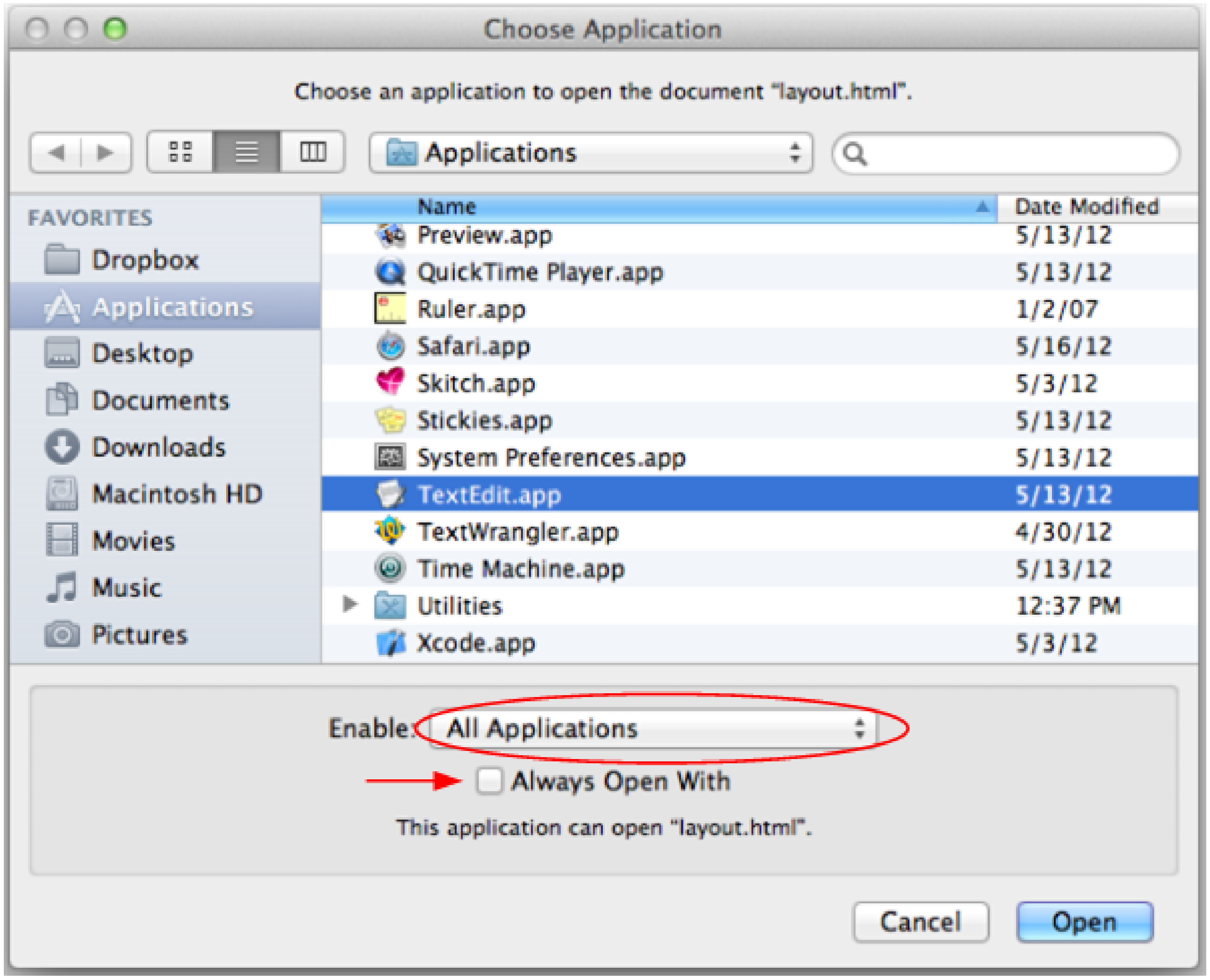
The file will probably open immediately in Notepad. You can just close it if you don't need to edit the program right now. The steps apply to all .html files, so you need never repeat the steps in the future. If you did the steps correctly, when you right-click any .html file and choose Open With, you should see Notepad as an option for opening the file for editing. Double-clicking a .html file should still open it in your default web browser.

If you run into a similar situation on a Mac, open the folder that contains the .html file, right-click (or CTRL+Click) the .html file, point to Open With, and verify that the app you need is not already listed on the Open With menu. (Don't add an application an second time if it's already on the Open With menu).



Open With in Mac OS X

If you don't see the app you want to use to open the file, click **Other** at the bottom of the window, and scroll through the list of available apps. If you don't see the one you're trying to add to the Open With menu, choose All Applications from the drop-down at the bottom of the dialog box. Click the name of the app you want to add to the Open With menu. Make sure the **Always Open With** checkbox is not checked. Then click **Open**.



Add an app to Open With in Mac OS X

When you're done, the app you selected will appear on the Open With menu when you click any .html file.

Q: I've read that you shouldn't use tables for page layout—that modern websites use table-less layouts. But you used tables in these layouts. What gives?

A: A table-less page layout isn't about abandoning tables altogether. In the early days of the Web, people would use a single table to define the whole page layout. There were no div tags (or CSS) for defining page divisions. So people used table rows and table cells to define page divisions. A table-less page layout is about not doing that anymore and instead using a named div to define each page division (as we've done in this lesson). Within any given page division, it's still perfectly normal and acceptable to use a table to organize information inside the division. And that's what we're doing in this lesson. We're not using one huge table to define the entire page layout. Rather, we're just using tables inside page divisions to organize content within that division.

Q: I can't quite seem to get my layout.html page together. Is there someplace I can see and copy the whole thing?

A: Sure! Below is how layout.html stands at the end of this lesson and in the assignment for this lesson.

```
<!DOCTYPE html>

<html>
```

```
<head>

  <title>My Layout</title>

  <style type="text/css">

    #header {

      background-color: #fe9900;

      color: white;

    }

    #nav {

      background-color: #ffff66;

      text-align: center;

    }

    #main {

    }

  </style>

</head>

<body>

  <!-- Header division -->

  <div id="header">

    <h1 style="margin:0">Lou's Records</h1>

  </div><!-- End header -->

  <!-- Navbar division -->

  <div id="nav">

    <table style="width: 100%" cellspacing="2" cellpadding="2">

      <tr style="background-color: #ffcc33;">

        <td style="width: 20%"><a href="index.html">Home</a></td>

        <td style="width: 20%"><a href="products.html">Products</a

        <td style="width: 20%"><a href="services.html">Services</a
        <td style="width: 20%"><a href="about.html">About Us</a></

        <td style="width: 20%"><a href="contact.html">Contact Us</

      </tr>

    </table>

  </div><!-- End nav -->

  <!-- Main content division -->

  <div id="main">

     Main content goes here.

  </div><!-- End main -->

</body>

</html>
```

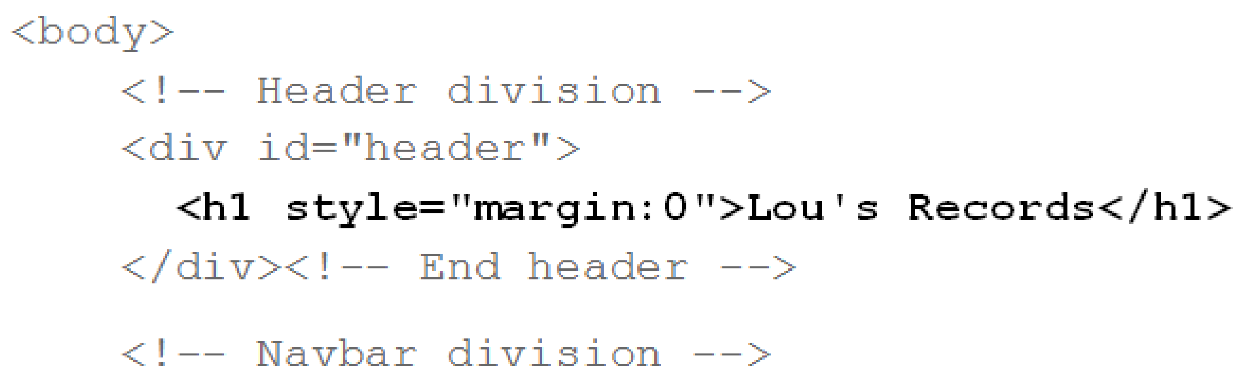
Assignment

So far, the page header in layout.html just contains some placeholder text. In this assignment, we'll replace that with a business name. And that name, of course, can be whatever you want it to be. I'll use *Lou's Records* as a working example, but you should feel free to use a name of your own choosing. Follow these steps:

1. Open **layout.html** for editing in Notepad, TextEdit, or whatever you normally use to create and edit Web pages.
2. Put the cursor inside the header div, and delete the placeholder text that reads *Header content goes here*.
3. With the cursor still in between the <div id="header"> and </div><!-- End header -->, carefully type the following:

```
<h1 style="margin:0">Lou's Records</h1>
```

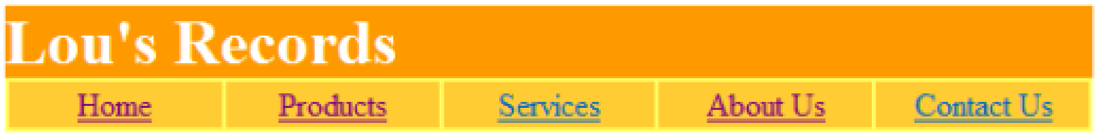
Verify that you typed the code correctly inside the header page division, as shown in this image:



```
<body>
    <!-- Header division -->
    <div id="header">
      <h1 style="margin:0">Lou's Records</h1>
    </div><!-- End header -->

    <!-- Navbar division -->
```

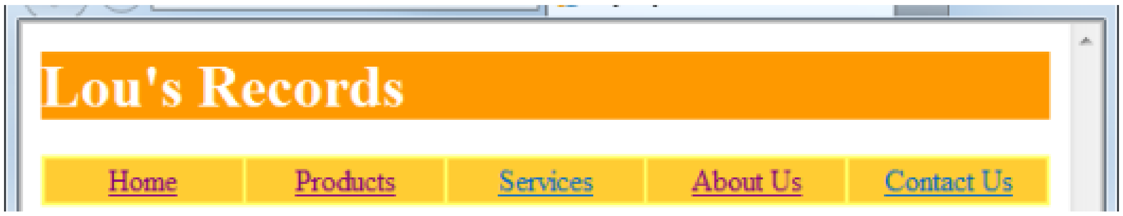
Correct code

4. Save the page.
5. Open **layout.html** in a Web browser. Or if it's already open in a Web browser, Refresh or Reload the page there. If you typed the code correctly, the new text should appear above the navigation bar, as in the image below.



New text in page header

You may be wondering why I put style="margin:0" in the <h1> tag for the heading. That has to do with the fact that HTML headings (h1, h2, h3, and so forth) have some empty margin space below them. You don't notice so much when the background is white. But with a colored background, like in our page header div, that margin shows up as an empty blank line under the heading (see image below).



Empty white space below heading

Putting style="margin:0" in the h1 tags removes the default margins from the heading, thereby removing the strange empty white space below the heading and bringing the navbar up right under the orange header division.

If you prefer to center the text in the header, just add *text-align:center* to your #header style rule, as below:

```
#header {

background-color: #fe9900;

color: white;

text-align:center;

}
```

Back to top

CLOSE