

Intermediate CSS3 and HTML5: Lesson 7

Alan Simpson

Chapter 1

Introduction

In today's lesson, we're going to start looking at techniques for creating fancier navigation bars and tools. But first, let's make sure everybody is clear on some basic concepts. There's no rule that says a website has to have a certain number of pages. It can have one page, thousands of pages, or any number in between. One thing is for sure, though—if the site contains more than one page, you should provide some kind of navigation tool that makes it easy to get from one page to the next within the site.

The exact appearance of the navigation bar is entirely up to you. And there's certainly no rule that says you need to link to every page within your site from the home page. Apple's website (www.apple.com) is a good example of that. There are no doubt hundreds of pages within their site. But they make no attempt to make them all accessible from the home page. The last time I checked, they were using a simple navigation scheme where the top of each page shows a navigation bar with links to a few major areas within the site.



Sample navigation bar

As we progress through this chapter, I'll use a sample, hypothetical navigation bar to illustrate some coding and styling techniques. Keep in mind that your websites are your creations. Your site, like any other site, can have any number of pages. So as we go through the chapter here, try to keep in mind that these are general examples that you can apply to a site where you have few pages. But exactly how many items you can fit across a horizontal navbar depends on how wide your layout is, and how wide the words in your items are.

We'll look at several different designs, too. Try to keep in mind that there's no right one, wrong one, good one, bad one, or one that you're required to use. We're just exploring possibilities of things you can try. It's food for thought on your own design.

What's With href="#"?

In this lesson and in many other tutorials and references online, you'll often see `<a>` tags with `href="#"`. That `#` is just a temporary placeholder for a nonexistent page. In order for the link to actually take you from the current page to another page, you'd have to replace `#` with the filename or URL of the page to which the link should take the user. But I have no way of knowing how many pages you intend to put in your site or what their filenames might be. So I can't put in actual filenames. The `#` is just there as a placeholder to let us design some navigation bars in advance of having actual pages to link to. At some point, after you've created all your pages, you would have to replace all the `href="#"` placeholders with links to actual pages in your site. So with that bit of info behind us now, let's head over to Chapter 2 and start exploring some ways to create cool-looking and functional navigation bars.

Chapter 2

Creating a Basic Navigation Bar

To practice with different looks and styles of navigation bars, I suggest you start with a new page that contains only tags needed for the navbar plus the HTML5 required tags and helper code we've discussed. We won't use MyTemplate2.htm right now because it's already set up for a vertical navbar. And my goal here isn't for everyone to create exactly the same page or site in a paint-by-numbers manner. Rather, my goal here is to show you the possibilities, so you can create what you want, when you want. So, I suggest that you follow these steps to get started:

1. Open your text editor (Notepad, TextEdit, or whatever you normally use).
2. Type, or copy and paste in, the following tags, which are basically the bare minimum tags for any HTML5 page, plus tags for an internal style sheet so we can play around with some style rules, a nav section for a practice navbar, and some styles and JavaScript to help older browsers handle newer HTML5 tags correctly.

```
<!DOCTYPE html>
<html>
<head>
  <title>Nav Practice</title>
  <style type="text/css">
    /* Make HTML5 layout elements block elements for older browsers */
    header, nav, aside, article, footer, section {
      display: block;
    }
  </style>
</head>
<body>
  <script>
    //Make older browsers aware of new HTML5 layout tags
    'header nav aside article footer section'.replace(/\w+/g, function (n) {
document.createElement(n) })
  </script>
  <!-- Navigation bar -->
  <nav>
    <a href="#">Home</a>
    <a href="#">Products</a>
    <a href="#">Services</a>
    <a href="#">About</a>
```

```
<a href="#">Contact Us</a>
</nav>
</body>
</html>
```

Save that page as navpractice.htm in your HTML5 Intermediate folder. When you first look at it in a browser, you will just see the links with the default styling.

[Home](#) [Products](#) [Services](#) [About](#) [Contact Us](#)

Links (default styling)

They look that way because, by default, links are inline elements (they flow like regular text). They're underlined and blue for unvisited pages, maroon for visited pages.

To start styling, you can create a style rule that applies to `<a>...` tags in the nav section only (so the styling doesn't affect other links you might have on the page outside that section). Use the descendant selector `nav a` for the style rule (which means `<a>` tags inside the nav section). Like all style rules, you must place this one inside a style sheet. Since we're just using a practice page, you can put it in the style rule that's already in the page, just above the `</style>` tag that ends that internal style sheet, as you see below.

```
<style type="text/css">
  /* Make HTML5 layout elements block elements for older browsers */
  header, nav, aside, article, footer, section {
    display: block;
  }
  /* Style for <a> tags in the nav section */
  nav a{
  }
</style>
```

Inside that style rule, you can style your `<a>...` tags however you like. For starters, let's give them a background color, remove the underline, and maybe change the text color, too. Feel free to use whatever colors you like. I'll use gray and white. Some Web browsers show a dotted border around a link when you click it. That doesn't always look so great in highly-styled navbars. You can get rid of that dotted box by adding `outline:none` to your style rule.

```
/* Style for <a> tags in the nav section */
nav a {
  background-color: gray;
  color: white;
  text-decoration: none;
  outline: none;
```

```
}
```

Save the page and open it in the browser, or reload or refresh it in the browser. The links will look something like this (nothing too exciting yet).

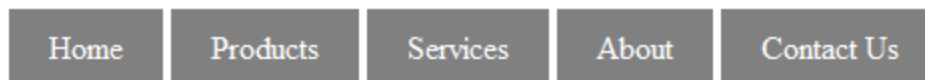


Links with a little CSS styling

To make them look more like buttons (and also larger so they're easier for people using touchscreens to tap), you can add some padding. How much is entirely up to you. But as an example, let's just say you decide to add 10 pixels of vertical (top and bottom) padding, and 20 pixels of horizontal (left and right) padding. Add a padding descriptor to the style rule as below.

```
/* Style for <a> tags in the nav section */
nav a {
  background-color: gray;
  color: white;
  text-decoration: none;
  outline: none;
  padding: 10px 20px;
}
```

Save the page and reload and refresh in the browser, and now the links look a bit more like buttons.



Links with some padding added

To make them look more like a regular navbar, it might be nice to close up the gaps and just put a thin border between each item. To do that, you can convert them to block elements using `display: block`, then float them to the left, which puts them right up against each other. Then you can use the `border-right` property to put a thin solid border on the right side of each one. Just add three new descriptors to the style rule, as you see below.

```
/* Style for <a> tags in the nav section */
nav a {
  background-color: gray;
  color: white;
  text-decoration: none;
  outline: none;
  padding: 10px 20px;
  display: block;
  float: left;
```

```
border-right: solid 1px silver;
}
```

Save and reload or refresh, and the navbar has a cleaner new look.

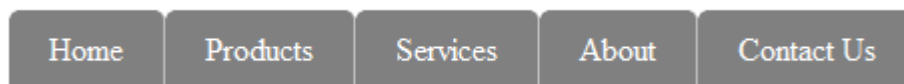


Tightened up the buttons

And last but not least, if you like the look of tabs rather than buttons, you can use border-radius to round the top left and top right corner of each button. Just add a couple more lines of code to the style rule as below.

```
/* Style for <a> tags in the nav section */
nav a {
  background-color: gray;
  color: white;
  text-decoration: none;
  outline: none;
  padding: 10px 20px;
  display: block;
  float: left;
  border-right: solid 1px silver;
  border-top-left-radius: 5px;
  border-top-right-radius: 5px;
}
```

Save and refresh or reload the page, and voila, you have a nice-looking set of tabs for navigation.



Tabs

Of course, you can design your navbar however you like. There's no set-in-stone right way, wrong way, good way, or bad way to design all navigation bars. And we'll do plenty more with this one. But I want you to notice how I really designed it step-by-step. We wrote a little code, checked our work, then wrote or changed a little code, and checked our work again. This is how virtually all professional design and coding is done.

The common beginner mistake is to write a mountain of code without checking your work nearly often enough, then looking at the page in the browser, seeing it's not what you'd anticipated, and then feeling helpless because you have this big mountain of messy out-of-control code and you don't even know where to begin to straighten it out. Not good. But if you write a little code, check your work in the browser,

write a little more code, check your work in the browser, and you do that often, it's a lot easier. Because if something goes haywire, it's probably in the little bit of code you just wrote (assuming everything was fine up to that point). So you can focus your debugging efforts on just the little bit of code you wrote since the last time everything checked out okay.

Also, checking your work often helps you see how each and every CSS property:value pair is contributing to the final design. So checking your work often isn't just a good, practical, and professional way to write code. It's also the best way to *learn* how to write code. But like I said, there's a lot more we can do with this nav bar we're building. Let's head over to Chapter 3 and play around some more.

Chapter 3

Controlling Widths and Hover Effects

So far, we've been playing around with a navbar, and we made the links look like tabs. Right now, we have unequal widths because each tab is only as wide as it needs to be to contain its content. That's because we never specified a width for each one. If you want them to be equal widths, that's a relatively easy thing to do. I would recommend you start by setting the side padding to zero in each one, because padding would otherwise get added to whatever width you specify. So basically, you would change the current padding descriptor from **padding: 10px 20px;** to **padding 10px 0;** to keep the top and bottom padding the same. Then you can give each a specific width using the width: property. There isn't a one-size-fits-all choice, of course. A good width depends on the words in your tags, how many tabs there are, and the size of the font. But I'll use 100 pixels for this example. And if you want the text centered in each tab or button, you'll need to add text-align:center, too, as you see below.

```
/* Style for <a> tags in the nav section */
nav a {
    background-color: gray;
    color: white;
    text-decoration: none;
    outline: none;
    padding:10px 0;
    display: block;
    float: left;
    border-right: solid 1px silver;
    border-top-left-radius: 5px;
    border-top-right-radius: 5px;
    width: 100px;
    text-align: center;
}
```

With that, each tab is equal in width.



Equal-width Tabs

The 100px width I chose was pretty much just a wild guess. But there's no need to guess right every time because it's so easy to change the width to some other number. But, if you guess too wide, they may be wider than the containing element and wrap to two rows. Or, if you guess too narrow, tabs with more text might wrap to two lines of text or cut off some of the text if there's no space between words in which to word wrap. For example, 50px would have been too narrow for each tab, creating a bit of a mess, like this:



Tabs too narrow for content

Don't panic if things don't look right on first guess. There are so many ways to fix it. For example, if you really have a lot of tabs and you're having a hard time getting everything to fit, you can do any, some, or all of the following to try to get a better fit:

- Change the tab width (the `width:` property)
- Or, reduce the amount of text on the wider buttons (for example, change *Contact Us* to *Contact*)
- Or, use a `font-size` property in the style rule to reduce the size of the font
- Or, use fewer tabs

Sizing to Fill the Available Space

If you want the tabs to fill the complete width of the navbar, you can set the widths as percents rather than pixels. But make sure you set the side padding to zero, as in the previous example, so that the padding width doesn't get added into the width you specify in the `width:` property. And even with that, the width of the buttons *and* the widths of the borders on the buttons have to total up to 100%. The easiest way to handle that is to use **box-sizing: border-box;** which makes the CSS `width:` property apply to the content width of an element *and* its borders. That way, you don't have to worry about the borders adding more width to the width you specify in your style rule. So you can simply divide the 100 by the number of items to determine how wide each item should be. For example, when there are five tabs as in the example, each would be 20% wide (because $100/5 = 20$). So you would just have to change that `width:100px` to `width:20%` and then add the box-sizing descriptor, like this:

```
/* Style for <a> tags in the nav section */
nav a {
  background-color: gray;
  color: white;
  text-decoration: none;
```

```
outline: none;
padding: 10px 0;
display: block;
float: left;
border-right: solid 1px silver;
border-top-left-radius: 5px;
border-top-right-radius: 5px;
width: 20%;
text-align: center;
box-sizing: border-box;
}
```

In most browsers, the button widths will be calculated automatically to fill the width of the containing element. But there's one small catch. Support for the CSS3 box-sizing property is a little uneven, and so you might not get quite the perfect fit you'd expect in all browsers. In some browsers (particularly older ones), the tabs will wrap to two rows because there isn't room for them to fit in the available space. Fortunately, the fix for that is easy, though it requires adding two more box-sizing descriptors with -moz- and -webkit- vendor prefixes. I've added the necessary code, preceded by an explanatory comment, to this style rule:

```
/* Style for <a> tags in the nav section */
nav a {
    background-color: gray;
    color: white;
    text-decoration: none;
    outline: none;
    padding: 10px 0;
    display: block;
    float: left;
    border-right: solid 1px silver;
    border-top-left-radius: 5px;
    border-top-right-radius: 5px;
    width: 20%;
    text-align: center;
    box-sizing: border-box;
    /* Box sizing for older browsers */
    -moz-box-sizing: border-box;
    -webkit-box-sizing: border-box;
}
```


With that code in place, the navbar should just fill the width of the browser window (however wide that might be), as in the example below.



Navbar as wide as browser window

In a page layout that has a wrapper div and horizontally-oriented navbar, it will be as wide as the nav section itself. We'll be talking more about layouts later in this course. So if you're having some trouble envisioning what that means, don't worry about it. For now, let's just stay with the topic at hand, which is creating and styling navigation bars.

Hover Effects

As you probably know, you can use the `:link`, `:visited`, `:hover`, and `:active` pseudo-classes of the `<a>` tag to define different style rules for different link states. You should always write these style rules in `lva` (`link`, `visited`, `hover`, `active`) order because some browsers won't get it right if you use a different order. The full selectors and when they apply are as follows:

- `a:link`: Applies to pages that the user hasn't visited in the past.
- `a:visited`: Applies to pages that the user has visited in the past.
- `a:hover`: Applies when the mouse pointer is touching the link.
- `a:active`: Applies when the mouse pointer is touching the link and the mouse button is in the down position (or the user is tapping the link on a touchscreen).

In navbars, we don't usually differentiate between visited and unvisited pages within the site. That sort of thing works fine in search engine results where you might get hundreds of pages to visit, and it's useful to have colors to differentiate between visited and unvisited sites. But you'll rarely see it in navbars. With navbars, we usually just have add two style rules with these selectors:

```
nav a:link, nav a:visited{  
  }  
nav a:hover, nav a:active{  
  }
```

Style rule descriptors in the first style rule apply to visited and unvisited pages in the site. Style rules in the second one apply when the user touches the mouse pointer to the link or taps it in a touchscreen. For example, let's say you just want to keep the gray background and white color for visited and unvisited links, and then perhaps you want to use a silver background and black text for the hover effect. You could add two style rules, and maybe a couple of comments, right under the style rule you already have (but still inside the internal style sheet) as you see below.

```

/* Style for <a> tags in the nav section */
nav a {
    background-color: gray;
    color: white;
    text-decoration: none;
    outline: none;
    padding: 10px 0;
    display: block;
    float: left;
    border-right: solid 1px silver;
    border-top-left-radius: 5px;
    border-top-right-radius: 5px;
    width: 20%;
    text-align: center;
    box-sizing: border-box; /* Box sizing for older browsers */
    -moz-box-sizing: border-box;
    -webkit-box-sizing: border-box;
}
/* Unvisited and visited link styling */
nav a:link, nav a:visited {
    background-color: gray;
    color: white;
}
/* Styling for hover and tap */
nav a:hover, nav a:active {
    background-color: silver;
    color: black;
}

```

Save the page and reload or refresh in the browser. At first, there's no change. But if you touch the mouse pointer to a tab, it takes on alternate colors.



Tab under mouse pointer highlighted

You're not limited to using a background color for the navbar. As with any other element type, you can use a background image instead. And you can change the background image on hover for an even more interesting effect. Follow me to Chapter 4 and I'll show you how.

Chapter 4

Background Images on Navbars

With CSS, you can apply a background image to virtually any element on the page. You learned some advanced techniques for background images in Lesson 2 of this course. Here we'll kick it up a notch—I'll show you how to change background images on hover events. This is a little trickier than it sounds because you have to *preload* the alternate background image with the rest of the page so that it's available on the user's computer the moment the mouse pointer touches the item that shows the alternate image. If you don't do that, the lag time of downloading just that one image the first time it's needed on hover can be enough to disable the effect.

Historically, developers have handled image preloading in a couple of ways. In the mid 1990s, people used the JavaScript language to preload the images to pages. That's not a bad approach, but it does require knowing JavaScript, and a small percentage of browsers have JavaScript disabled, so the effect doesn't work for those. Along with CSS came a CSS-only method of repositioning a single background image, sometimes referred to as the sliding-doors technique or CSS sprites. With that method, both background images were stored in a single image file. To get the hover effect, the position of the background image was changed in the `:hover` style rule. That technique is also good, but it does require advanced graphics editing skills. Today, most developers will use the CSS3 technique of initially assigning two background images to the element in question, so both images are downloaded along with the page. Then, they'd use CSS to control which image is visible on hover and which isn't. So there's no JavaScript involved, and you don't need such advanced graphics-editing skills. You do, however, need background images that are about the height of the tabs or buttons in your navbar.

The kinds of background images we're talking about here are like the ones used in the Apple toolbar that we looked at when we started this lesson. For example, these two, named `navback.png` and `navhover.png`, are available in the `pix2` subfolder you downloaded for this course. Each is 40 pixels tall (the height of a tab), and just 10 pixels wide. The width doesn't really matter, though, because with background images, we can use `background-repeat:repeat-x` to repeat the image horizontally to fill the width of each tab. The difference between the two images is subtle, but it's enough to tell the difference, as you'll see shortly. Of course, in real life, the images can be anything you want them to be. These are just for example.



Sample background images for tabs

To use them as background images, we can use the `background` shorthand property to apply both images at the same time to all the `<a>...` tags in the nav. That way, both images are sure to be downloaded with the page, but initially, only one will show because both are centered (vertically) and repeated horizontally so each tab is filled with the image. Here's the whole style rule again, but the only thing that's really changed is that the `background-color` descriptor has been replaced by the `background` descriptor that defines two background images.

```
/* Style for <a> tags in the nav section */
```

```

nav a {
    background: url(pix2/navback.png) center repeat-x, url(pix2/navhover.png) center
repeat-x;
    color: white;
    text-decoration: none;
    outline: none;
    padding: 10px 0;
    display: block;
    float: left;
    border-right: solid 1px silver;
    border-top-left-radius: 5px;
    border-top-right-radius: 5px;
    width: 20%;
    text-align: center;
    box-sizing: border-box; /* Box sizing for older browsers */
    -moz-box-sizing: border-box;
    -webkit-box-sizing: border-box;
}

```

That alone won't get you any hover effect. But it will make the first background image fill each tab horizontally, as you see below, which is a nice effect in its own right.



Background image on tabs

For the hover effect, all you need to do is assign one image to the link and visited states, the other image to the hover and active states. Use the same style rules as for the background colors on those. Just specify one background image in each instead. You can change the text color using the color: property in the style rule for the hover, too, like this:

```

/* Unvisited and visited link styling */
nav a:link , nav a:visited {
    background: url(pix2/navback.png);
}
/* Styling for hover and tap */
nav a:hover, nav a:active {
    background: url(pix2/navhover.png);
    color: black;
}

```

Save the change and reload or refresh the page in the browser. When you point to a tab, the navhover.png background image will replace the other, giving that tab a slightly different appearance from the others.



Different background image under mouse pointer

Showing Users the Current Page

Many sites show navbars with tabs for every page in the site, including the page that the user is currently viewing. That can occasionally lead to confusion, because when you click the tab that represents the current page, nothing really happens on the page because you've just reloaded the page you're already viewing. Users can be confused by that because they think something is broken when they keep clicking a link and nothing seems to change in the browser. They don't realize they're just going to the same page over and over again.

To help avoid that confusion, some sites will keep the tab that represents the current page highlighted for as long as the user stays on the page. That provides a visual cue that many users eventually realize is telling them which page in the navbar they're currently viewing.

There really isn't anything in CSS, per se, that allows you to say, "When the user is on this page, do this, and when the user is on that page, do that." But you can create a style class and then apply it selectively as you see fit in your site. In fact, if you want to use the same style for the current page that you do for the hover and active states, you can just use that same style rule. All you have to do is add another selector to that style rule. But rather than apply a pseudo-class, apply a regular class name. As always, the class name can be anything you like, as long as it starts with a letter and contains no spaces. So I'll use *currentpage* as the class name. To add it as a selector to the style rule for hover and active, put a comma after the second selector that's already there. Then add the third selector as shown below. You can also change the comment to that effect, as you see below.

```
/* Styling for hover, tap, and current page */
nav a:hover, nav a:active, nav a.currentpage {
  background: url(pix2/navhover.png);
  color: black;
}
```

The new selector says the style applies to nav links during the hover state, during the active state, and also to any `<a>` tag that contains `class="currentpage"`. Right now, no `<a>` tags in the page contain `class="currentpage"`, so the style rule has no immediate effect. But you can use it later, after you've created your pages, to selectively apply that style to just one tab on each page . . . the tab that represents the current page.

For example, when you create your home page (typically named `index.htm` or `index.html`), you can apply the `currentpage` class to (only) the tab that represents that page, like this:

```
<!-- Navigation bar -->
```

```
<nav>
<a href="#" class="currentpage">Home</a>
<a href="#">Products</a>
<a href="#">Services</a>
<a href="#">About</a>
<a href="#">Contact Us</a>
</nav>
```



Note

Right now, I'm still showing the href="#" placeholders because I'm assuming the other pages in the site don't exist yet. But the concept still applies. Eventually, presumably, you'll have created all your pages and filled in the filenames for those pages. Then you can use class="currentpage" in the <a> tag that represents the current page.

In the browser, the tab for the home page will be highlighted automatically, even when the mouse pointer isn't touching it, as you see below.



Tabs when currently viewing the home page

That immediate highlighting will help most users understand that they're currently on the home page (so there's no sense clicking that link since they don't have to go to the page they're already on).

Then, when you create your second page, put the class="currentpage" in the second link only, like this:

```
<!-- Navigation bar -->
<nav>
<a href="#">Home</a>
<a href="#" class="currentpage">Products</a>
<a href="#">Services</a>
<a href="#">About</a>
<a href="#">Contact Us</a>
</nav>
```

When the user is viewing that page, only that link will be highlighted as you see below, again giving the user a visual cue about which page in the site they're currently viewing.



Tabs when currently viewing the Products page

Do the same for each page in your site so that when the user is viewing the Services page, the services tab (only) is highlighted. When the user is viewing the About tab, the About tab (only) is highlighted, and so forth. You can still apply the style rule to the hover and active states as well, to make it easier to see where the mouse pointer is located. It's not a technical requirement or anything to do these things. But it is a courtesy that many users will appreciate because they make your site easier to use and navigate.

Let's head over to Chapter 5 and wrap up what you've learned in this lesson.

Chapter 5

Conclusion

Today's lesson was all about navigation bars. Even before you know what pages you'll have in your site, you can start working on your navigation bar design by setting up a set of "dummy links" that use `href="#"` to link to nowhere. The links won't be functional. But they don't need to be functional when all you're concerned about is the visual look and feel of them.

Since some people are likely to visit your site using touchscreens, it's always a good idea to make your links fairly large. You can make the clickable area of a link larger by applying CSS padding to it, or by specifying a width or height. While you're free to design your navigation bars as you see fit, here are some general guidelines, reviewing what you've learned in this lesson.

- Use standard `<a>...` tags for the links.
- Using CSS, apply fonts, text decoration, color, as you see fit.
- Use CSS to apply padding (or width and height).
- Optionally, use `display:block` and `float:left` to tighten any gaps between links.
- Optionally, use the CSS border property to apply border lines.
- Optionally, use `border-radius` to round the top corners for tabs, or all four corners for rounded buttons.
- Use the `:link` and `:visited` pseudo-classes to define styling for the regular appearance.
- Use the `:hover` and `:active` pseudo-classes to define styling for when the mouse pointer is hovering over an item or the user taps the item.
- You can apply background images to navbar links using the same CSS properties you use to apply a background image to anything else.

- You can create a style class to highlight any individual tab, and then apply it to one tab per page—the tab that represents the current page.

In the next lesson, you'll learn even more advanced navigation techniques using drop-down and flyout menus. See you there!

Supplementary Material

[HTML <a>](#)

http://www.w3schools.com/tags/tag_a.asp

Click this link for a good basic tutorial on <a>... tags.

[CSS Links](#)

http://www.w3schools.com/css/css_link.asp

Here's another good basic reference for the CSS links and pseudo-classes, which you might want to keep in your browser's bookmarks or favorites for future reference.

[CSS Navigation Bar](#)

http://www.w3schools.com/css/css_navbar.asp

Here's a relatively simple navbar that uses a list (... tags) which isn't really necessary unless you're going to add drop-down menus. But it still provides a good reference point for remembering the basics of the styling.

[CSS3 Box-Sizing Property](#)

http://www.w3schools.com/cssref/css3_pr_box-sizing.asp

Here's a quick tutorial on the CSS3 box-sizing property.

FAQs

Q: I think I may have messed up my MyTemplate2.htm page. Can I get a clean copy?

A: Sure. But don't forget that the fonts will only work if you also have the Calligraffiti fonts in the same folder as the page and style sheet.

```
<!DOCTYPE html>
<html>
<head>
  <title>MyTemplate2</title>
```



```
<!-- Link to external style sheet -->
<link href="styles.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <script>
    //Make older browsers aware of new HTML5 layout tags
    'header nav aside article footer section'.replace(/\w+/g, function (n) {
document.createElement(n) })
  </script>
  <div id="wrapper">
    <header>
      <h1>Sample H1 Heading</h1>
    </header>
    <nav>
      <a href="#">Home</a>
      <a href="#">Products</a>
      <a href="#">Services</a>
      <a href="#">About</a>
      <a href="#">Contact Us</a>
    </nav>
    <article>
      Top
      <p>Paragraph 1</p>
      <p>Paragraph 2</p>
      <p>Paragraph 3</p>
      <p>Paragraph 4</p>
      <p>Paragraph 5</p>
      <p>Paragraph 6</p>
      <p>Paragraph 7</p>
      <p>Paragraph 8</p>
      <p>Paragraph 9</p>
      <p>Paragraph 10</p>
      <p>Paragraph 11</p>
      <p>Paragraph 12</p>
      <p>Paragraph 13</p>
      Bottom
    </article>
    <footer>
```

```
    footer
  </footer>
</div><!-- End wrapper -->
</body>
</html>
```

Q: How about styles.css, can I get a clean copy of that too?

A: You bet, here it is:

```
/* styles.css */
/* Downloadable Font from FontSquirrel */
@font-face {
  font-family: 'Calligraffiti';
  src: url('Calligraffiti-webfont.eot');
  src: url('Calligraffiti-webfont.eot?#iefix') format('embedded-opentype'),
  url('Calligraffiti-webfont.woff') format('woff'),
  url('Calligraffiti-webfont.ttf') format('truetype'),
  url('Calligraffiti-webfont.svg#CalligraffitiRegular') format('svg');
  font-weight: normal;
  font-style: normal;
}
body {
  background-image: url(pix2/hubblesmall.png), url(pix2/stars.png);
  background-repeat: no-repeat;
  background-position: center;
  background-attachment: fixed;
  background-size: contain, cover;
  background-color: black;
}
/* Fixed-position sized to edges of viewport */
#wrapper {
  position: fixed;
  top: 10%;
  left: 10%;
  right: 10%;
  bottom: 10%; /* White for older browser, rgba for newer */
}
```

```
    background: white;
    background: rgba(255,255,255,.6);
    border-radius: 10px;
}
/* Make HTML5 layout elements block elements for older browsers */
header, nav, aside, article, footer, section {
    display: block;
}
/* Style that applies to both header and footer */
header, footer {
    position: absolute;
    height: 15%;
    width: 100%;
    padding: 1%;
}
header {
    top: 0;
    left: 0;
    border-radius: 10px 10px 0 0;
}
footer {
    bottom: 0;
    left: 0;
    border-radius: 0 0 10px 10px;
}
/* This nav section is on the side, positioned absolutely */
nav {
    position: absolute;
    top: 15%;
    left: 0;
    bottom: 15%;
    width: 15%;
    padding: 1%;
}
/* Style for <a> tags in the nav section */
nav a {
    font-family: 'Calligraffiti';
```

```

    font-weight: bold;
    background: url(pix2/navback.png) center repeat-x, url(pix2/navhover.png) center
repeat-x;
    color: White;
    text-decoration: none;
    outline: none;
    padding: 10px 0;
    display: block;
    border-right: solid 1px silver;
    border-radius: 5px;
    width: 85%;
    text-align: center;
    box-sizing: border-box; /* Box sizing for older browsers */
    -moz-box-sizing: border-box;
    -webkit-box-sizing: border-box;
}
/* Unvisited and visited link styling */
nav a:link, nav a:visited {
    background: url(pix2/navback.png); /* Make background image height equal to button
height */
    background-size: auto 100%;
}
/* Styling for hover and tap */
nav a:hover, nav a:active, nav a.currentpage {
    background: url(pix2/navhover.png);
    color: black;
    background-size: auto 100%;
}
/* Main article sized to fit inside header, footer nav */
article {
    position: absolute;
    top: 15%;
    left: 15%;
    right: 0;
    bottom: 15%;
    padding: 10px;
    overflow: auto;
    background: rgba(255,255,255,.6);

```

```
}  
/* Level-1 Headings */  
h1 {  
    font: 36pt/36pt Calligraffiti, Fantasy;  
    text-shadow: -3px -3px #777;  
    margin: 8px 0 0 0;  
}
```

Q: How would I center or right-align a navbar?

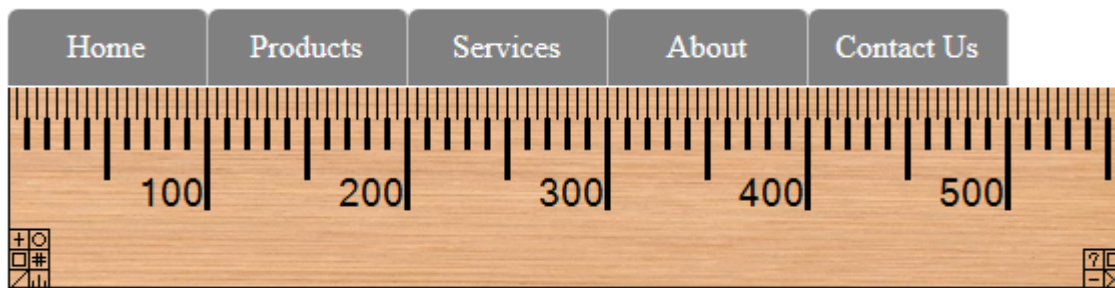
A: When you use `float:left` to keep the buttons or tabs close together, centering and right-aligning can be tricky because the float will seem to take precedence over every attempt to center (and there's no such thing as `float:center`). However, I just said it's tricky, not impossible. It can be done. The first step is to figure out exactly how wide the tabs or buttons are. And by that, I mean their total width, not their individual widths. That's necessary because in order to center them, you'll have to put them in a div that's just wide enough to contain them (no more, no less), and then center that div. If you can't figure out the combined widths mathematically, you can usually figure it out with a little trial-and-error guesswork. Or you can use a screen ruler to measure. There are many free screen rulers available online for Mac and Windows. Just search Google for *free screen ruler* to see what's out there. There are also *screen calipers* for those who are accustomed to measuring things with calipers.



Caution

Screen rulers and calipers can be useful, but they aren't a requirement for this course. If you have no experience downloading and installing apps and programs and you don't know about the potential dangers of doing so, it would probably be best if you don't pursue it, unless you can get some expert help.

If you have or find a screen ruler that you like, measurement is pretty easy. Open the page that contains the item you want to measure. In the browser, make sure you press `CTRL + 0` (Windows) or `COMMAND+0` (Mac) to set the magnification to normal (100%). And make sure you use the zero key on the keyboard, not the letter "O" on the keyboard. That's important because most browsers allow you to change the magnification by press `CTRL ++` / `CTRL +-` or `COMMAND ++` and `COMMAND +-`, and other means, such as holding down the `CTRL` key while spinning the mouse wheel, or pinching and spreading on a touchscreen). The ruler measurement will be wrong if the magnification isn't set to 100%. Pressing `COMMAND + 0` or `CTRL + 0` in the browser window is a quick and easy way to make sure the magnification is set to 100%. Then, open up your screen ruler and make your measurement. In the example below, I'm using a screen ruler to measure a sample set of tabs whose width happens to be about 500px.



Tabs are 500 pixels wide

So whether you just take a wild guess or measure, the next step is to wrap the tabs in a div that has a CSS class name. For example, below I've added div tags to wrap them inside a div with the class-named tabs (just a name I picked out of a hat). That div must be contained within the nav section, as you see below.

```
<!-- Navigation bar -->
<nav>
  <div class="tabs">
    <a href="#">Home</a>
    <a href="#">Products</a>
    <a href="#">Services</a>
    <a href="#">About</a>
    <a href="#">Contact Us</a>
  </div><!-- End tabs div -->
</nav>
```

Next, we need a style rule for that div. Like any style rule, it has to go in a style sheet. Since we're styling a div styled with class="tabs," the selector can be div.tabs. You can give it your measured width or take a wild guess width. Include margin:0 auto; in the style rule to center the tabs. You can also add overflow:hidden; which will make that containing div have the same height as the tabs. To make the containing div easier to see, you can put a temporary border around it. I often use a red dashed border for that sort of thing, because it's easy to see. So here's how that style rule should look. You can type or copy and paste that rule just above the </style> tag that marks the end of the internal style sheet in the navpractice.htm page.

```
/* Center the tabs */
div.tabs{
  width:500px;
  margin:0 auto;
  overflow:hidden;
  border:dashed 1px red;
}
```

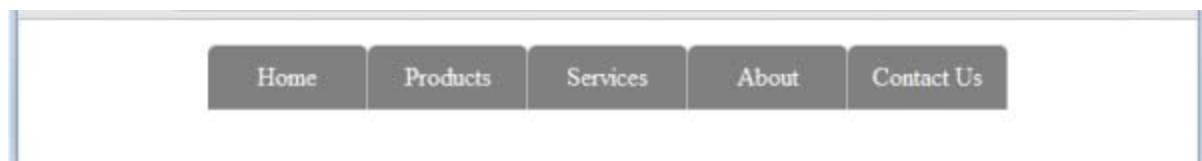
If you happen to get the width just right, the div's border will just fit snugly around the tabs when you save the change and reload or refresh the page.



Red border snug around tabs

If you make the div too narrow, the tabs will wrap to two or more rows. If you make it too wide, there will be extra space past the last tab and the tabs will look off-center. But that's no biggie because it's so easy to change. If your div is too wide, try a smaller number for the width. If it's too narrow, try a larger number for the width. With a little trial and error, you should be able to get a snug fit around the tabs as you saw in the example above.

Once the red dashed border fits snugly around the tabs, you can remove the temporary border by removing the `border:dashed 1px red;` descriptor. Assuming you did everything else correctly, the tabs should be centered horizontally within their containing element (which, in our working example, is just the browser window as you see below).



Centered navbar

Again, centering is just another option—it's not required. This last passage was really just a means of showing a strategy for centering a navbar that tends to float to the left because you used `float:left` in your styling. And incidentally, if you wanted to right-align that whole set of tabs, you'd just have to change `margin: 0 auto;` to `float:right`, as you see below.

```
/* Right-align the tabs */
div.tabs {
  float:right;
  overflow: hidden;
  border:dashed 1px red;
}
```

Assignment

In this lesson, we created a horizontal navbar with tabs. But with just a few modifications to the CSS, we can convert it to a vertical stack of buttons with a different font in our sample template, like this:



Buttons in left column of MyTemplate2

Right now in the MyTemplate2.htm page, we just have the word Nav in the navigation section as a placeholder. You can replace that word with the exact same tags we used in this lesson for our working example. Or, if you already know what pages you might have in your own site, you could create links for those.

Then, in the style sheet, styles.css, you could use almost the exact same style rules for nav a{} elements that we used in navpractice. The width of each button would probably be best at about 85%, and you can round all the corners for a button appearance. I also used the Calligraffiti font and boldface in the example above. Feel free to take a shot at it on your own if you like. If you need some help, use the steps below.

Here are the steps for adding the fancy navigation buttons to the MyTemplate2 page:

1. Open your HTML5 **Intermediate folder**, right-click (or CTRL + Click) **MyTemplate2.htm**, choose **Open With**, and then your editor name.
2. Locate the <nav>...</nav> tags in the page, and remove the placeholder word Nav from between those tags.
3. Type, or copy and paste the following text where that placeholder text was so all the links are between the <nav>...</nav> tags.

```
<a href="#">Home</a>
<a href="#">Products</a>
<a href="#">Services</a>
<a href="#">About</a>
<a href="#">Contact Us</a>
```


Close and save the page. If you want to take a quick look in the browser, double-click the page's icon. The unstyled links should be in the left column, as you see below. If your links don't show, click Reload or Refresh in the browser to make sure you're seeing the latest code.



Unstyled links on the page

These aren't the best-looking links in the world, so we need to do some CSS styling. We're using the styles.css style sheet for our CSS styling in this course, so we'll need to add the CSS code to that one. Here are the steps:

1. In your HTML5 Intermediate folder, right-click or CTRL + Click **styles.css**, choose **Open With**, and then your editor name.
2. To keep the style sheet organized, we'll put the new style rules for the nav link right after the style rule for the nav section itself. So put the cursor just to the left of the comment that starts with **/* Main article** and press ENTER once or twice to add a blank line or two.
3. Copy and paste all three nav `a{}` tags from NavPractice (or from below) into styles.css, right where you inserted new lines.

```
/* Style for <a> tags in the nav section */
nav a {
    background: url(pix2/navback.png) center repeat-x, url(pix2/navhover.png) center
repeat-x;
    color: white;
    text-decoration: none;
    outline: none;
    padding: 10px 0;
    display: block;
    float: left;
    border-right: solid 1px silver;
    border-top-left-radius: 5px;
    border-top-right-radius: 5px;
    width: 20%;
```

```

text-align: center;
box-sizing: border-box; /* Box sizing for older browsers */
-moz-box-sizing: border-box;
-webkit-box-sizing: border-box;
}

/* Unvisited and visited link styling */
nav a:link , nav a:visited {
background: url(pix2/navback.png);
}

/* Styling for hover and tap */
nav a:hover, nav a:active, nav a.currentpage {
background: url(pix2/navhover.png);
color: black;
}

```

4. Save the style sheet.

Open the page in the browser, click Reload or Refresh if necessary. It's not too pretty, but it's a little closer to what we want.



Partially-styled links in MyTemplate2

The main problem with the links is that each is way too narrow. And they're in a column where they should be stacked vertically. So let's take care of that now:

1. In styles.css, in the nav a{} style rule, remove the line that reads float:left;.
2. In that same style rule, change **width:20%;** to **width:85%;**.
3. To make each link look more like a button rather than a tab, round all four corners by replacing these two lines:

```

border-top-left-radius: 5px;
border-top-right-radius: 5px;

```

... with this one line:

```
border-radius: 5px;
```

4. Save the style sheet, and reload or refresh the page in the browser.

That should look better:



That's better

And finally, if you want to get really fancy, you can use the Calligraffiti from Lesson 6 in place of that default font. Perhaps the biggest challenge there is just spelling the font name right. And I would make it boldface, too. So you can type or copy and paste these two lines into the nav a{} style rule. I used Cursive as the default font type for browsers that are unable to use the downloaded Calligraffiti font.

```
font-family: 'Calligraffiti';  
font-weight: bold;
```

And here's one last finishing touch I might recommend: We never actually gave the buttons a specific height. Rather, we relied on top and bottom padding to provide height in this example. That's not necessarily a bad thing. Though it doesn't mean that the height of each button can vary slightly depending on the font in use. And the background image won't automatically adjust to variations of height. But, we can use background-size to set the height of the background image to 100%, so its height always matches the height of the button. Since background-size is new in CSS3, I wouldn't recommend putting in the shorthand syntax, as older browsers don't handle that shorthand syntax well when you put the background size there. I would just add it to the style rules for the link/visited and hover/active states, like this:

```
/* Unvisited and visited link styling */  
nav a:link , nav a:visited {  
    background: url(pix2/navback.png);  
    /* Make background image height equal to button height */
```

```
background-size: auto 100%;
}
/* Styling for hover and tap */
nav a:hover, nav a:active, nav a.currentpage {
background: url(pix2/navhover.png);
color: black;
background-size: auto 100%;
}
```

Anyway, if you got it all right, the fancy font should show up in the nav buttons, and the hover effect will apply when you touch the mouse pointer to a button.



Nav buttons done