

Advanced HTML & CSS



Learn by doing step by step exercises.
Includes downloadable class files that work on Mac & PC.

EDITION 5.0



Published by:

Noble Desktop

185 Madison Ave, 3rd Floor

New York, NY 10016

nobledesktop.com

Copyright © 2015–2024 Noble Desktop NYC LLC

Publish Date: 7-08-2024

All rights reserved. No part of this book may be reproduced or transmitted in any form by any means, electronic, mechanical, photocopy, recording, or otherwise without express written permission from the publisher. For information on reprint rights, please contact

hello@nobledesktop.com

The publisher makes no representations or warranties with respect to the accuracy or completeness of the contents of this work, and specifically disclaims any warranties. Noble Desktop shall not have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the instructions contained in this book or by the computer software and hardware products described in it. Further, readers should be aware that software updates can make some of the instructions obsolete, and that websites listed in this work may have changed or disappeared since publication.

Adobe, the Adobe Logo, Creative Cloud, and Adobe app names are trademarks of Adobe Systems Incorporated. Apple and macOS are trademarks of Apple Inc. registered in the U.S. and other countries. Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the U.S. and other countries. All other trademarks are the property of their respective owners.

Table of Contents

SETUP & INTRODUCTION

Downloading the Class Files	7
------------------------------------------	---

Before You Begin	9
-------------------------------	---

Topics: Choosing a code editor to work in
Installing & Setting up Visual Studio Code
Installing & using the iOS Simulator (Mac Only)

INFO & EXERCISES

SECTION 1

Exercise 1A: Starting a New Website from Scratch & Emmet Shortcuts	13
-------------------------------------------------------------------------------------	----

Topics: Creating a new HTML page with Emmet shortcuts
Adding content & styling it

Exercise 1B: Variable Fonts (from Google Fonts) & Unitless Line-Height	21
-----------------------------------------------------------------------------------------	----

Topics: Adding custom web fonts from Google Fonts
Using font-weight
Unitless line-height

Exercise 1C: Styling the Navbar	29
----------------------------------------------	----

Topics: Styling the Navbar
Hover & Focus Link Styles

Exercise 1D: CSS Box Model: Content-Box vs. Border-Box	33
---------------------------------------------------------------------	----

Topics: How border-box is different than content-box
Visualizing the box model in Chrome's DevTools
Fixing spacing issues around images
Flexible padding: using percentage amounts

SECTION 2

Exercise 2A: SVG Sizing (Width & Height) and Embedding SVG	41
-----------------------------------------------------------------------------	----

Topics: How Width & Height Affect Sizing of SVG
Embedding SVG (instead of linking)
Styling SVG using CSS
Using `currentColor`

Table of Contents

Exercise 2B: CSS Position Property 49

Topics: Static position & the normal document flow

Relative position

Absolute position

The dynamic duo: relative parent, absolute child

Fixed position

Exercise 2C: Fixed Position Navbar, RGBA, & Backdrop Filters 57

Topics: Creating a fixed navbar on wide & tall screens

RGBA color

Blurring & desaturating a background with CSS backdrop filters

Exercise 2D: Improvements for Mobile Devices 63

Topics: Preventing mobile text size adjustment

Preventing a fixed navbar on short screens

Dealing with the iPhone's dynamic island (or notch)

CSS calc()

SECTION 3

Exercise 3A: Background Gradients & Text Gradients 69

Topics: CSS background gradients

CSS text gradients

Exercise 3B: Multiple Backgrounds, Viewport Sizing (vw), & Clamp 75

Topics: Multiple backgrounds on a single element

Colorizing a photo by overlaying a transparent gradient

Using viewport sizing units (vw)

Using Clamp to set a min & max font size

Exercise 3C: Creating Columns with Flexbox 79

Topics: Displaying content as columns using Flexbox

Vertical alignment with Flexbox

Exercise 3D: Pseudo-Elements & the Content Property 85

Topics: Using pseudo-elements

The content property

Seeing pseudo-elements in Chrome's DevTools

Table of Contents

SECTION 4

Exercise 4A: Attribute Selectors 89

Topics: Adding link icons with attribute selectors

“Ends with” attribute selector

“Begins with” attribute selector

“Contains” attribute selector

Exercise 4B: Relational Selectors 95

Topics: Using first-child & last-child

Using first-of-type

Using nth-child

Direct child/descendant selectors

Exercise 4C: CSS Variables (Custom Properties) 101

Topics: Defining & using CSS variables

The power of inheritance

Exercise 4D: Light & Dark Modes using CSS 109

Topics: Styling webpages for light & dark modes

Previewing light & dark modes in Chrome DevTools

“Alt” text for embedded SVGs

SECTION 5

Exercise 5A: Creating Forms with HTML 115

Topics: Creating a form & text inputs with labels

Adding an email input, submit button, textarea, & select menu

Fieldset, legend, & radio buttons

Checkbox vs radio button

Exercise 5B: Styling Forms (with Attribute Selectors) 125

Topics: Styling form elements

Targeting inputs with attribute selectors

Exercise 5C: Relative Sizes: Em and Rem 131

Topics: Em units

Rem units

SECTION 6

Exercise 6A: Off-Screen Side Nav Using Only CSS 137

Topics: Responsive off-screen navigation

Toggling the navigation with a checkbox

CSS transitions

Table of Contents

Exercise 6B: Box-Shadow, Text-Shadow, & Z-Index 147

Topics: Using the CSS box-shadow property

Changing an element's default stack order with position and z-index

Inset shadows

Adding drop shadows to text with CSS text-shadow

Layering multiple text-shadows for a detached outline effect

Exercise 6C: CSS Transitions 153

Topics: Using CSS transitions to animate elements on hover

Adding easing

Custom easing with Ceaser

Exercise 6D: CSS Transforms with Transitions 159

Topics: Testing transforms using the DevTools

Adding a scale transform & transitioning it

Transform origin

Adding a rotate transform

Using the translate transform to nudge elements

BONUS MATERIAL

Exercise B1: Responsive Images 167

Topics: Img srcset

Picture element

REFERENCE MATERIAL

Noble's Other Workbooks 177

Downloading the Class Files

Here's how to get the files you'll need for this training:

1. Navigate to the **Desktop**.
 2. Create a **new folder** called **Class Files** (this is where you'll put the files after they have been downloaded).
 3. Go to nobledesktop.com/download
 4. Enter the code **wd2-2407-08**
 5. If you haven't already, click **Start Download**.
 6. After the **.zip** file has finished downloading, be sure to unzip the file if it hasn't been done for you. You should end up with a **Advanced HTML CSS Class** folder.
 7. Drag the downloaded folder into the **Class Files** folder you just made. These are the files you will use while going through the workbook.
 8. If you still have the downloaded .zip file, you can delete that. That's it! Enjoy.
-

Before You Begin

If You've Done Other Noble Desktop Coding Books

If you've setup Visual Studio Code in other Noble Desktop workbooks, the only new thing you may want to do (it's optional though) is **Installing & Using the iOS Simulator (Mac Only)**. There's nothing new for Windows users to do.

Choosing a Code Editor to Work In

You probably already have a preferred code editor, such as [Visual Studio Code](#), [Sublime Text](#), etc. You can use whatever code editor you want, but we highly recommend Visual Studio Code.

Installing Visual Studio Code

Visual Studio Code is a great code free editor for Mac and Windows.

1. Visit code.visualstudio.com and download **Visual Studio Code**.
 2. To install the app:
 - Mac users: Drag the downloaded app into your **Applications** folder.
 - Windows users: Run the downloaded installer.
During installation check on the **Add "Open with Code"** ... options.
-

Setting Up Visual Studio Code

There are a few things we can do to make Visual Studio Code more efficient.

Setting Preferences


1. Launch **Visual Studio Code**.
2. In Visual Studio Code do the following:
 - Mac users: Go into the **Code** menu and choose **Preferences > Settings**.
 - Windows users: Go into the **File** menu and choose **Preferences > Settings**.
3. In the search field type: **wrap**
4. Find **Editor: Word Wrap** and change its setting from **off** to **on**

NOTE: This ensures you'll be able to see all the code without having to scroll to the right (because it will wrap long lines to fit to your screen size).

Before You Begin

Setting Up Open in Browser

Visual Studio Code does not include a quick way to preview HTML files in a web browser (which you'll do a lot). We can add that feature by installing an extension:

1. On the left side of the Visual Studio Code window, click on the **Extensions** icon .
2. In the search field type: **open in browser**
3. Under the **open in browser** click **Install**.

Defining a Keystroke for Wrapping Code

Visual Studio Code has a feature to wrap a selection with some code. This is very useful so we'll want to use it frequently. There's no keystroke for it by default, so let's define one:

1. In Visual Studio Code do the following:
 - Mac users: Go into the **Code** menu and choose **Preferences > Keyboard Shortcuts**.
 - Windows users: Go into the **File** menu and choose **Preferences > Keyboard Shortcuts**.
2. In the search field type: **wrap with**
3. Double-click on **Emmet: Wrap with Abbreviation** and then:
 - Mac users: Hold **Option** and press **W** then hit **Return**.
 - Windows users: Hold **Alt** and press **W** then hit **Enter**.
4. You're done, so close the **Keyboard Shortcuts** tab.

Installing & Using the iOS Simulator (Mac Only)

This free app lets you simulate an iPhone/iPad on a Mac, which is great for testing.

1. On the Mac, launch the **App Store** (it's found in the **Applications** folder).
2. Search for **Xcode** (which includes iOS Simulator) and install it. If you're on an older version of macOS you may need to upgrade your system to be able to run Xcode.
3. Once **Xcode** is installed **launch** it. (Xcode is found in the **Applications** folder.)
4. If asked to install anything, go ahead and do so. Once Xcode is running, go into the **Xcode** menu and choose **Open Developer Tool > Simulator**.
5. So that we don't have to launch Xcode each time we want to use iOS Simulator, **Control-click** (or **Right-click**) the **Simulator** icon in the Dock and in the menu choose **Options > Keep in Dock**.

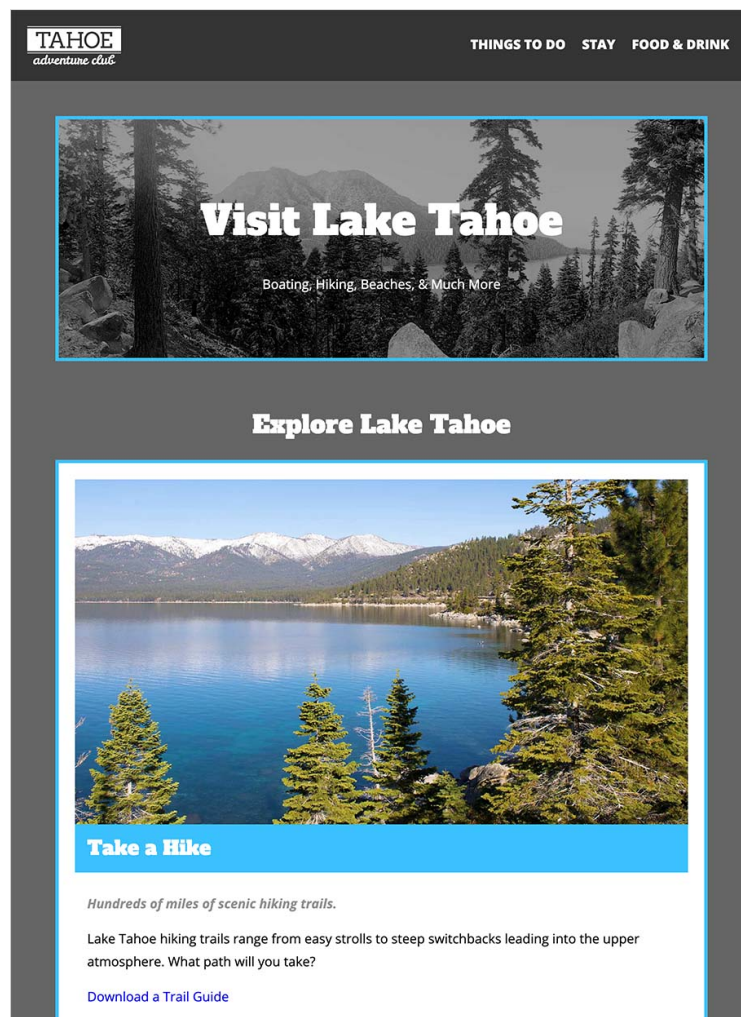
Before You Begin

6. How to preview a local webpage:

- Open the .html file in a web browser on your Mac (Safari, Chrome, etc.).
- Copy the URL (file:///Users...)
- In the Simulator, click into the URL bar, and then click **Paste and Go**.

TIP: **Option–Drag** to simulate a pinch to zoom.

Exercise Preview





Exercise Overview

This is the first in a series of exercises in which you'll create a webpage from scratch (the finished page is shown above). In this exercise you'll start coding the HTML and CSS. You'll learn how to use Emmet Shortcuts to make things easier and faster.

Creating a New HTML page with Emmet Shortcuts

1. Launch your code editor (**Visual Studio Code**, **Sublime Text**, etc.). If you are in a Noble Desktop class, launch **Visual Studio Code**.
2. We'll be working with the **Tahoe** folder located in **Desktop > Class Files > Advanced HTML CSS Class > Tahoe**. You should open that folder in your code editor if it allows you to (like Visual Studio Code does).

3. In this **Tahoe** folder, create a new **index.html** file. Here's how to do that in **Visual Studio Code**:
 - In the **Explorer** (files) panel on the left, hover over the **Tahoe** folder name and to the right click **New File** .
 - Type in **index.html** and hit **Return** (Mac) or **Enter** (Windows) to make the file.
4. With **index.html** now open, we need to type in all the default HTML tags such as **html**, **head**, **body**, etc. Luckily there's something called **Emmet** that can help us do this. **Emmet** shortcuts are a abbreviations you can use to write **HTML** and **CSS** with less typing. **Emmet** can be found in many code editors, including **Visual Studio Code**.

Type **!** and hit **Tab** to get all the **html** starter code!
5. In the **title** tag, replace **Document** with:
Things to do in Lake Tahoe - Tahoe Adventure Club
6. Save **index.html**
7. In this **Tahoe** folder, create a new **main.css** file in a **css** folder. Here's how to do that in **Visual Studio Code**:
 - In the **Explorer** (files) panel on the left, hover over the **Tahoe** folder name and to the right click **New File** .
 - Type in **css/main.css** and hit **Return** (Mac) or **Enter** (Windows) to make the file.

NOTE: By typing **css/main.css** it makes a **css** folder with a **main.css** file inside it!
8. We need to link our **HTML** page to this **css** file. Switch to **index.html**
9. Put your cursor at the end of the **title** tag and hit **Return** (Mac) or **Enter** (Windows) to create a new empty line.
10. Type **link** and hit **Tab** and **Emmet** will expand that to an entire link tag for **CSS**!
11. Your cursor will already be in the **href=""** so type **css/main.css** and you'll end up with this code:

`<link rel="stylesheet" href="css/main.css">`
12. Save **index.html**

Creating Structure Tags with Emmet

Let's learn more **Emmet** shortcuts to help us create the **HTML** structure for this page.

1. Put your cursor in the **body** tag.
2. Type **nav** and hit **Tab** to get an opening and closing **nav**:

`<nav></nav>`

3. Put your cursor in the **nav** tag (between the opening and closing tags) and hit **Return** (Mac) or **Enter** (Windows) to create a new empty line.
4. Type **img** and hit **Tab** to get an **img** tag like this:

```
<img src="" alt="">
```
5. In an empty line below the nav, type **main>p** (do not add spaces) and hit **Tab** to get:

```
<main>  
  <p></p>  
</main>
```
6. Let's see another way to create these 2 sections. Delete the **nav** and **main** (so you're back to an empty body tag).
7. Type **nav+main** (do not add spaces) and hit **Tab** to get:

```
<nav></nav>  
<main></main>
```
8. Put your cursor in the **nav** tag (between the opening and closing tags) and hit **Return** (Mac) or **Enter** (Windows) to put them on different lines.
9. Let's see one more Emmet shortcut. Type **ul>li*3>a** (no spaces) and hit **Tab** to get:

```
<ul>  
  <li><a href=""></a></li>  
  <li><a href=""></a></li>  
  <li><a href=""></a></li>  
</ul>
```

Amazing! Look at all that code we created with so little typing!

- Emmet uses **+** for sibling tags (a tag **after** another tag).
- Emmet uses **>** for nested tags (a tag **inside** another tag).
- Emmet uses ***** and a number to indicate how many tags we want.

Adding the Real Page Content

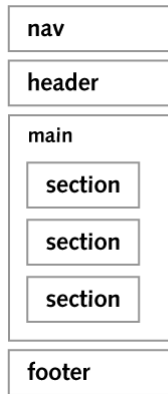
1. Delete all the tags inside of body so you're back to an empty body tag.
2. We've typed out all the page's content to save you time and hassling of repetitive copying/pasting.

From the **snippets** folder, open **homepage-content.html**

3. Hit **Cmd-A** (Mac) or **Ctrl-A** (Windows) to select all the code.
4. Hit **Cmd-C** (Mac) or **Ctrl-C** (Windows) to copy it.
5. Close the file.

6. You should be back in **index.html** and in the **body** tag, paste the new code.

Notice the content has 4 primary elements: **nav**, **header**, **main**, and **footer**. The **main** element contains 3 **section** elements. Here's a visualization of the structure:



7. Save **index.html**
8. Preview **index.html** in Chrome (because we'll be using its DevTools)
9. Notice the following:
 - There's no styling yet, so the bare content looks pretty rough.
 - When the window's width is smaller than the images, you'll have a horizontal scrollbar at the bottom of the page. The images do not scale down to fit into the window, so we'll need to fix that.
10. Leave **index.html** open in the browser as you work, so you can reload the page to see the code changes you'll make.

Fluid, Hi-Res Images

Why are the images so huge? We're using hi-res images suitable for display on hi-res screens, which have a higher pixel density (smaller, more tightly packed pixels) than low-res screens. Low-res images look a bit pixelated or blurry on hi-res screens. To make a hi-res image, the pixel width of the image file should be twice the width we code in HTML or CSS. Hi-res (**2x**) images occupy the same space as low-res (**1x**) images, so 2x images pack more pixels into the same space. More (smaller) pixels in a given space = hi-res!

1. Switch to **main.css** in your code editor.

2. To get rid of the horizontal scrollbar, we need to scale the images down to fit inside their parent container. Add the following new rule for all images:

```
img {  
    max-width: 100%;  
}
```

This **max-width** prevents the image from extending beyond the bounds of its parent element, ensures that the image does not scale larger than its native size, yet allows it to scale down to fit smaller screens.

3. Save **main.css**.
4. Switch back to Chrome and:
 - Reload **index.html**.
 - Resize the browser smaller to see that the images shrink to fit inside the window.
 - Resize the browser wider to see the long lines of text are hard to read.
5. Return to **main.css** in your code editor.
6. Let's give the page a dark gray background color so the sections stand out more. Above the **img** rule, add the following new rule:

```
body {  
    background: #555;  
    margin: 20px;  
}
```

TIP: Emmet also has CSS shortcuts. To get **margin: 20px**; you can type **m20** and hit **Tab**!

NOTE: Instead of **background-color**, we used the shorthand version **background**.

7. Let's give each section a white background so they stand out on the dark background. Below all the other rules, add this new rule:

```
section {  
    background: #fff;  
}
```

8. Save **main.css**, switch back to Chrome, and reload **index.html**.

The page should now be dark gray with 3 white areas.

Constraining the Width of Content

By default, block-level elements (such as header and section) have a width that's 100% of their parent element. This is why the text fills the available space. Let's set a fixed-pixel max-width as an upper limit on how much the text content can stretch.

1. Return to **main.css** in your code editor.
2. To ensure the content never stretches past a width of **800 pixels**. Above the **section** rule add this new rule:

```
header, main, footer {  
    max-width: 800px;  
}
```

NOTE: We'll be styling the **nav** differently than the other page content, so we're not styling that for now.

3. Save **main.css**, switch back to Chrome, and reload **index.html**.
4. Resize the browser window in and out to see that the layout expands until it reaches the max-width but doesn't budge after it reaches 800 pixels.

Centering the Content & Page Margins

The content would look nicer centered on the page. To center block elements (such as header and section) we use auto margins. Margin is space outside an element. We don't know how much space needs to be to the left or right, but the browser can **automatically** figure that out for us. The browser takes whatever available extra space there is in the parent container, divides it in two, and applies it equally on the left and right side of the element.

1. Switch back to **main.css**.
2. In the **header, main, footer** rule add margin as shown below:

```
header, main, footer {  
    max-width: 800px;  
    margin: auto;  
}
```

3. Save **main.css**, switch back to Chrome, and reload **index.html**.
 - Make sure the browser window is wide enough so you can see the column of content is now centered.
 - The header (**Visit Lake Tahoe** and the paragraph below it) doesn't stand out much, so let's make it more interesting.

Adding a Background Image to the Header

1. Switch back to **main.css** in your code editor.

2. Let's add a dark background image and make the text white. Above the **section** rule, add the following new rule:

```
header {  
    background-image: url(../img/masthead-darkened.jpg);  
    background-position: center;  
    color: #fff;  
    text-align: center;  
    padding-top: 50px;  
    padding-bottom: 50px;  
}
```

NOTE: **background-position** accepts one or two values. If you only specify one value, the other will automatically be **center**. So our value is interpreted as **center center**, which horizontally and vertically centers the photo.

3. Save **main.css** and reload **index.html** in Chrome.
 - The image is larger than the header, so we only see some of the center part of the photo. Below is what the entire photo looks like (the photo below wasn't darkened, so it's easier for you to see).
 - The page width is limited to 800px. We made the photo 1600px wide, so it can end up being displayed as a hi-res image. Let's scale it down to fit the container.



4. Switch back to **main.css** in your code editor.
5. Add the following bold code for background-size:

```
header {  
    background-image: url(../img/masthead-darkened.jpg);  
    background-position: center;  
    background-size: cover;  
    color: #fff;
```

6. Save **main.css** and reload **index.html** in Chrome. The background image should now be scaled down to show you as much as possible in the given space.

Using Shorthand for the Background Property

All the different property declarations for background-image (size, position, etc.) can be combined into a single line of code using shorthand.

1. Switch back to **main.css** in your code editor.
2. In the **header** rule, delete the **background-position** and **background-size** lines and any remaining whitespace.
3. We'll add those back in a moment, but to use the shorthand syntax we must first change **background-image** to **background**:

```
header {  
  background: url(../img/masthead-darkened.jpg);  
  color: #fff;
```

4. At the end of the background value, add the position and size back in by adding the code shown below in bold:

```
header {  
  background: url(../img/masthead-darkened.jpg) center / cover;  
  color: #fff;
```

NOTE: The order of values within the background property do not matter, except when using background-position and background-size together. For more info on this shorthand syntax visit sixrevisions.com/css/background-css-shorthand

5. Save **main.css** and reload **index.html** in Chrome. The header background photo should look the same, but with less code.

Styling the Footer

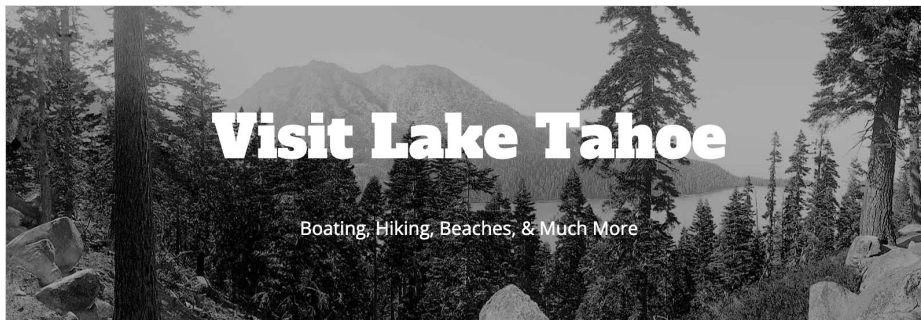
1. Scroll to the bottom of the page and notice that the single line of copyright text in the footer is too dark to read and we want it centered.
2. Switch back to **main.css** in your code editor.
3. Below the other rules, add the following new rule:

```
footer {  
  color: #ccc;  
  text-align: center;  
}
```

TIP: The Emmet shortcut to get **text-align: center;** is **tac** and hit **Tab**.

4. Save **main.css** and reload **index.html** in Chrome. The page is starting to take shape, but we'll continue developing it in the next exercise.
-

Exercise Preview



Exercise Overview

In this exercise you'll refine the typography of the page started in the previous exercise. You'll add custom webfonts from Google Fonts, examine font-weight, font-style, and learn about the benefits of unitless line-height.

Getting Started

1. If you completed the previous exercise, **index.html** should still be open from the **Tahoe** folder.

If you did not do the previous exercise, open the **Tahoe Ready for Typography** folder in your code editor and open **index.html**.
2. Preview **index.html** in a browser. Our design calls for some custom web fonts and better line spacing.
3. Leave **index.html** open in the browser as you work, so you can reload the page to see the code changes you'll make.

Loading Custom Web Fonts From Google Fonts

1. In a new browser tab, go to fonts.google.com
2. There are two fonts we want to use (**Alfa Slab One** and **Open Sans**). In the search field at the top of the page, search for **Alfa**
 - Click on **Alfa Slab One** in the search results.
 - Click on **Get font**.

3. Let's find a second font:

- In the navbar click **Fonts**.
- Search for **Open**
- Click on **Open Sans** in the search results.
- Click **Get font**.

4. At the top right of the page click the shopping bag icon.

5. You should see 2 fonts listed: **Alfa Slab One** and **Opens Sans**.

6. Click **Get embed code**

Notice **Open Sans** is labeled as **Variable**. This means we can change the weight to anything between the weight values listed (300 to 800 in this case), offering way more flexibility than a few premade weights like 300, 400, etc.

7. Copy the code for the links which load the fonts from Google's servers:

```
<link rel="preconnect" href="https://fonts.googleapis.com">
<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
<link href="https://fonts.googleapis.com/css2?
family=Alfa+Slab+One&family=Open+Sans:ital,wght@0,300..800;1,300..800&display=swap"
rel="stylesheet">
```

8. Keep this page open, but return to **index.html** in your code editor.

9. Paste the link above the link to main.css, as shown in bold below:

```
<title>Things to do in Lake Tahoe - Tahoe Adventure Club</title>
<link rel="preconnect" href="https://fonts.googleapis.com">
<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
<link href="https://fonts.googleapis.com/css2?
family=Alfa+Slab+One&family=Open+Sans:ital,wght@0,300..800;1,300..800&display=swap"
rel="stylesheet">
<link rel="stylesheet" href="css/main.css">
```

10. Save the file.

NOTE: Putting the link to Google Fonts before links to other CSS files will start the font download as soon as possible. To learn more about font loading and what the **display=swap** part of the link means, read css-tricks.com/font-display-masses

The rel="preconnect" lines of code tell the browser to connect to the domain where Google stores its fonts. This will speed things up for when the CSS file goes to download the font files.

Adding Fonts to the Page

Now that the fonts are loaded, we must apply them to the text. We want everything to be Open Sans (except for headings) so let's apply that to everything in the body.

1. Return to Google Fonts in the browser.
2. Copy the code for the **Open Sans** font:

```
font-family: "Open Sans", sans-serif;
```
3. Keep this page open, but return to your code editor.
4. Open **main.css** from the **css** folder.
5. Paste the following into the **body** rule (at the top) as shown below:

```
body {  
  background: #555;  
  margin: 20px;  
  font-family: "Open Sans", sans-serif;  
}
```

6. We want our 3 heading levels to be the **Alfa Slab One** font. Below the **img** rule, add the following new rule so we can style them all with one rule:

```
h1, h2, h3 {  
  
}
```

7. Return to Google Fonts in the browser.
8. Copy the code for the **Alfa Slab One** font:

```
font-family: "Alfa Slab One", serif;
```
9. Return to **main.css** in your code editor.
10. Paste the code into the rule for **h1, h2, h3** as shown below:

```
h1, h2, h3 {  
  font-family: "Alfa Slab One", serif;  
}
```

11. We don't want to fallback to a serif font. We'd rather fall back to Open Sans or any sans-serif font if our custom font doesn't load. Edit the font stack as follows:

```
font-family: "Alfa Slab One", "Open Sans", sans-serif;
```

12. Save **main.css**, switch back to the browser, and reload **index.html** to see the new fonts.

Styling the Headings & Font-Weight

1. Switch back to **main.css** in your code editor so we can style our headings.
2. Below the **h1, h2, h3** rule, add the following new rules:

```
h1 {  
    font-size: 50px;  
}  
h2 {  
    font-size: 30px;  
    color: white;  
    text-align: center;  
    margin-top: 60px;  
}  
h3 {  
    font-size: 24px;  
}
```

3. Save the file and reload the page in the browser to see:

- The headings are better sized and styled.
- Notice that the headings look extra thick (especially the **x** in **Explore**).

Alfa Slab One only comes in one weight (regular 400), but headings are bold by default (even if it's faked). We'll need to remove the bold.

4. Return to **main.css** in your code editor.
5. In the **h1, h2, h3** rule add the following code shown below in bold:

```
h1, h2, h3 {  
    font-family: "Alfa Slab One", "Open Sans", sans-serif;  
    font-weight: normal;  
}
```

6. Save **main.css**, switch back to the browser, reload **index.html**, and notice:

- The headings now look the correct thickness (look at the **x** in **Explore**).
- After the **Take a Hike** heading (which is an h3) there are some paragraphs, but the first paragraph is a brief tagline. We want that to stand out from the others. We want to do this for all 3 sections. The tagline paragraph always follows an h3, so we can style it with an adjacent sibling selector (or next-sibling selector).

7. Back in **main.css**, below the **h3** rule add the following new rule:

```
h3 + p {  
  font-weight: 700;  
  font-style: italic;  
  opacity: .5;  
}
```

NOTE: The **normal** keyword is the same as the numeric **400** weight. **700** is the same as **bold**. For more info, see tinyurl.com/moz-fw

Font-style accepts values of normal, italic, or oblique. If a given font family has an italic or oblique font (our font family does), the browser will use it, otherwise the browser will fake the slanted effect.

8. Save **main.css** and reload **index.html** in the browser.

- In each of the 3 white sections, the first paragraph after the heading should be bold, italic, and gray.
- It's nice that we could do this without creating a new class name.

Unitless Line-Height

1. Resize the browser window smaller so the **Visit Lake Tahoe** in the header breaks onto multiple lines. You'll have to open the DevTools to be able to make the content area small enough.
 - Notice the line spacing of the headings and paragraphs.
 - The paragraph line spacing is rather tight. Let's increase the space to make the text easier to read.
2. Return to **main.css** in your code editor.
3. In the **body** rule at the top, add line-height as shown in bold:

```
body {
```

CODE OMITTED TO SAVE SPACE

```
  line-height: 28px;  
}
```

4. Save the file and reload **index.html** in the browser. The paragraphs are better, but the headings (especially **Visit Lake Tahoe**) are now too tight.

What is happening? The line-height we applied to the body is inherited by all the elements inside it. By putting a fixed line height on the body, it sets that exact line height for all text on the page, including headings. A 28px line-height is way too small for the **Visit Lake Tahoe** h1 that has a font-size of 50px!

We could move the line-height from body to paragraphs, but then we'd have to separately declare the line height for each element (h1, h2, h3, etc.). Let's try a different option, using a percentage value instead of a fixed pixel value.

5. Return to **main.css** in your code editor.
6. Change the line-height value to a percentage, as follows:

```
body {  
    
  CODE OMITTED TO SAVE SPACE  
  line-height: 175%;  
}
```

NOTE: Why 175%? The line-height (28px) **divided** by the default font size (16px) equals **1.75**, which is **175%** when expressed as a percentage.

7. Save the file and reload **index.html** in the browser to see that nothing changed. We still have the same problem. Percentages are not helping us out in this regard. Why?

When using a specified value (even a percentage) the value is static and child elements will inherit the raw number value. What we really need to use is a **unitless** line height. When we write the value without specifying the unit of measurement, the value is dynamic!

8. Return to **main.css** in your code editor.
9. Change the line-height value as shown below. Do NOT to add a unit of measure!

```
body {  
    
  CODE OMITTED TO SAVE SPACE  
  line-height: 1.75;  
}
```

Setting the value to **1.75** is equivalent to setting the value to 175%, but with a crucial difference. The browser will multiply a unitless line-height with each element's unique font-size, rather than computing the number once and then applying the same value to all the elements.

10. Save the file and reload **index.html** in the browser and notice:

- The paragraphs look the same.
- The headings look better, but now they have a bit too much space. The bigger the text's font-size, the bigger the 1.75 line-height ends up being.

NOTE: Because unitless line-heights are more flexible, we will use them (instead of px values) from now on, and recommend you use them in your work.

11. Return to **main.css** in your code editor.

12. To reduce the line-height of our headings. In the **h1, h2, h3** rule, add the following bold code:

```
h1, h2, h3 {  
  font-family: "Alfa Slab One", sans-serif;  
  font-weight: normal;  
  line-height: 1.2;  
}
```

13. Save the file and reload **index.html** in the browser. That's looking better!

Styling the Footer

1. Scroll to the copyright line of text at the bottom of the page.

We want the copyright text to be thinner than this current normal weight. We loaded a **light 300** weight, so let's use that.

2. Return to **main.css** in your code editor.

3. In the **footer** rule, add font-weight as shown below in bold:

```
footer {  
  color: #ccc;  
  text-align: center;  
  font-weight: 300;  
}
```

4. Save **main.css**.

5. Switch back to the browser and reload **index.html** to see the text is now thinner.

How Are Variable Fonts Different Than Regular Fonts?

- Traditional fonts have a one font file for each style (regular/italic) and each weight. You're also limited because you can only choose from a specific list of premade weights (such as 300, 400, 500, etc).
 - Variable fonts take a feature (such as weight) and turn it into a customizable setting within a single font file. Not only is this more efficient, it's also more flexible as you can choose any weight in range!
 - While Google Fonts typically only offer weight as a variable option, do not limit your thinking to the standard options that you've seen before (bold, italic, extended, condensed, etc.). Font designers can make variable fonts do all sorts of cool things!
-

Styling the Navbar

Exercise Preview




Exercise Overview

In this exercise, you'll style the navbar for mobile devices.

Getting Started

1. If you completed the previous exercise, **index.html** should still be open from the **Tahoe** folder.

If you did not do the previous exercise, open the **Tahoe Ready for Navbar** folder in your code editor and open **index.html**.

2. Preview **index.html** in Chrome:
 - **Ctrl-click** (Mac) or **Right-click** (Windows) anywhere on the page and select **Inspect** to open Chrome's DevTools.
 - In the upper left of the DevTools panel click the **Toggle device toolbar** button  to enter device mode.
 - From the **Dimensions** menu in the toolbar above the page, select **iPhone SE**.
3. We're going to start by styling the mobile layout, and then work up to larger screens. This type of development is called "mobile first".

Leave the page and DevTools open in the browser and return to your code editor.

Overriding Default List Styles

We semantically coded our navigation links as a list, but don't want the bullets.

1. In your code editor, open **main.css** (from the **css** folder).

2. Above the **header**, **main**, **footer** rule, add the following new rules:

```
a {  
    color: #2985d1;  
    text-decoration: none;  
}  
nav ul {  
    list-style-type: none;  
    margin: 0;  
    padding: 0;  
}  
nav ul a {  
    color: #fff;  
}
```

3. Save the file, return to Chrome, and reload the page.

Now that we've removed the bullets and extra space, we want the list items to be in a horizontal row, rather than this vertical stack.

4. Return to **main.css** in your code editor.
5. Below the **nav ul** rule add the following new rule:

```
nav li {  
    display: inline;  
}
```

6. Save the file, return to Chrome, and reload the page.

The three links under the logo should now all be in a line.

7. Return to **main.css** in your code editor.
8. Let's center everything in the nav and add space below it. Above the rule for **nav ul** add the following:

```
nav {  
    text-align: center;  
    margin-bottom: 20px;  
}
```

9. Save the file, return to Chrome, and reload the page.

The logo and line of links should now be centered, but the links need more styling.

Styling the Navbar

- Back in your code editor, in the **nav ul a** rule, add the following new properties shown below in bold:

```
nav ul a {
  color: #fff;
  font-weight: 800;
  text-transform: uppercase;
  padding: 8px;
  display: inline-block;
}
```

NOTE: The **display: inline-block;** keeps the links inline, but makes padding work like it does on block elements. It also ensures an entire link would wrap down the next line if the screen is too small.

- Save the file, return to Chrome, and reload the page. That looks better!

Adding Hover & Focus States to the Nav Links


Let's add a hover to the navbar links. Mobile users won't see the hover effect, but desktop users can. We'll also use the same style for the focus state (when a user keyboard navigates to it).

- Back in your code editor, below the **nav ul a** rule add the following code:

```
nav ul a:hover,
nav ul a:focus {
  color: #97d0f2;
}
```

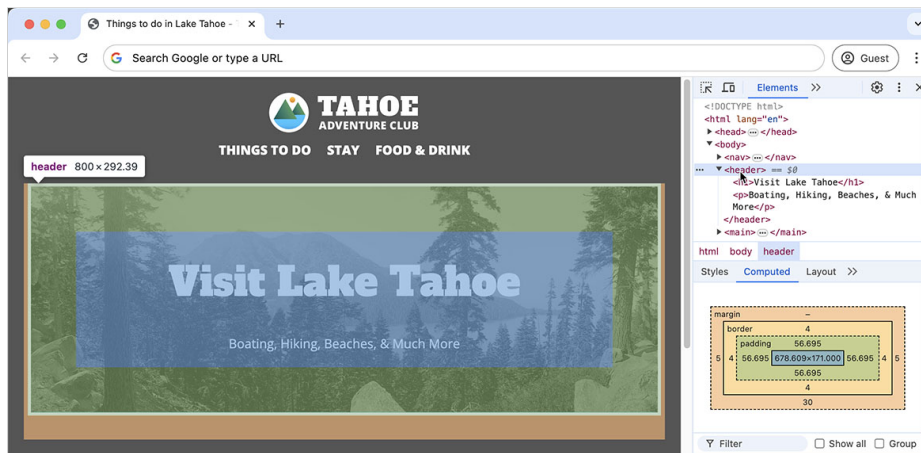
NOTE: Notice that the comma separated selectors are on two different lines? While we could have kept them on the same line, it's easier to read with them on two lines. CSS still treats it as one selector even though they are on different lines.

- Save **main.css** and reload **index.html** in the browser.

- In the upper left of the DevTools panel click the **Toggle device toolbar** button  to leave device mode.
 - Close DevTools.
 - Hover over the links to see they turn blue.
 - Because we used the same styling for the focus state, users that keyboard navigate the page will see the same effect when a link is selected via the keyboard.
-

CSS Box Model: Content-Box vs. Border-Box

Exercise Preview



Exercise Overview

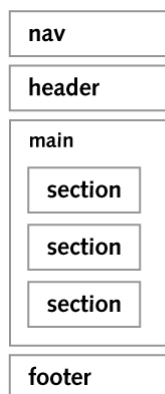
In this exercise you'll learn how the default box model treats widths, padding, and margins and how to switch to a newer (better) box model.

Styling the Sections (Borders, Margins, & Padding)

1. If you completed the previous exercise, **index.html** should still be open from the **Tahoe** folder.

If you did not do the previous exercise, open the **Tahoe Ready for Box Model** folder in your code editor and open **index.html**.

2. We want the **header** and the 3 **sections** to have the same border and spacing. As a reminder, here's a visualization of our page's structure:



3. Open **main.css** (from the **css** folder) in your code editor.

4. Below the **header, main, footer** rule, add this new rule:

```
header, section {  
  border: 4px solid #2fb3fe;  
  margin-bottom: 30px;  
}
```

5. Save **main.css**.

6. Preview **index.html** in Chrome (we'll be using its DevTools).

- The header and 3 sections below should all have a blue border now.
- We don't like how the content within each of the bordered areas touches the blue border, so let's add padding.

7. Back in **main.css**, add **padding** as shown below in bold:

```
header, section {  
  border: 4px solid #2fb3fe;  
  margin-bottom: 30px;  
  padding: 20px;  
}
```

8. Save **main.css**, switch back to Chrome, and reload **index.html**.

You should now have some space inside each blue bordered area. Padding is inside the border. Margin is outside the border.

Adding a Background Color to Headings

1. In the white background sections, let's make the headings (h3) look like they're attached to the images above them. Switch back to **main.css** in your code editor.
2. In the **h3** rule, add the following code shown in bold:

```
h3 {  
  font-size: 24px;  
  background: #2fb3fe;  
  color: #fff;  
}
```

3. Save **main.css**.

4. Switch back to Chrome and reload **index.html**. The blue background color shows that headings stretch to fill the entire available width because they are block-level elements.
5. The text inside these headings is way too cramped. Switch back to **main.css**.

CSS Box Model: Content-Box vs. Border-Box

6. Add padding on all 4 sides, by adding the following code shown in bold:

```
h3 {
  font-size: 24px;
  background: #2fb3fe;
  color: #fff;
  padding: 15px;
}
```

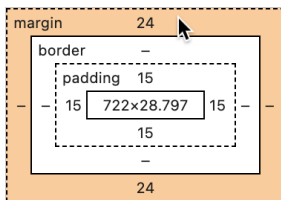
7. Save **main.css** and reload **index.html** in Chrome.

The spacing inside the blue heading backgrounds (padding) looks a lot better, but we have a lot of space above them. The heading is not flush with the image, as we want. This is most likely caused by margin.

Visualizing the Box Model in Chrome's DevTools

1. Let's use developer tools to see what's causing the issue. **Ctrl-click** (Mac) or **Right-click** (Windows) on the **Take a Hike** heading and choose **Inspect**.
2. In the DevTools, if you don't see the box model diagram shown below, click the **Computed** tab to see it.

As shown below, hover over **margin**. Then in the page notice the margin above and below the h3 in the page will appear as orange.



NOTE: A default amount of margin was applied by the browser (which is proportionate to the font size).

3. Let's remove the default margin on the top. Switch back to **main.css**.
4. Zero out the **margin-top** by adding the following code shown below in bold:

```
h3 {
  margin-top: 0;
}
```

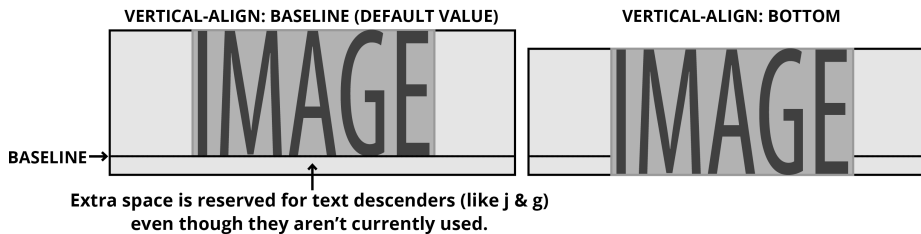
5. Save **main.css**, switch back to Chrome, and reload **index.html**. There's still a gap, even though the heading has no top margin. Perhaps the image has some margin?

6. **Ctrl-click** (Mac) or **Right-click** (Windows) on the image, choose **Inspect**, and look in the **Computed** tab to see that the image has no margin. What is causing the space below the image?

Fixing Spacing Issues Around Images

Images are rendered on a line of text as though they're part of a paragraph, even though they're not in a paragraph tag. By default, text (and images) are vertically aligned to the text baseline (where the bottom of each letter is positioned as you would when writing on lined paper).

Baseline makes sense for text, including lowercase characters like *j* and *g*. However, for images, which don't have descenders, the default vertical-align: baseline creates extra, unfilled space below them. A simple fix is to vertically align images to the **bottom**, rather than the **baseline**. The following diagram illustrates this:



1. Switch back to **main.css** in your code editor.
2. In the **img** rule, add the following bold code:


```
img {
  max-width: 100%;
  vertical-align: bottom;
}
```
3. Save **main.css**.
4. Switch back to Chrome and reload **index.html** to see that the extra space below the image is now gone, making the image and the heading look attached.
5. Keep the page open, but close Chrome's DevTools.

Paragraph Spacing

Let's align the paragraphs with the

text. We inset the headings with 15 pixels of padding, so we'll need the same amount on the paragraphs in each section.

1. Switch back to **main.css** in your code editor.

CSS Box Model: Content-Box vs. Border-Box

2. Below the **section** rule, add the following new rule:

```
section p {  
  margin: 15px;  
}
```

NOTE: We're not using padding because paragraphs have default margins (but no default padding), and we want to override the default margins.

3. Save **main.css** and reload **index.html** in Chrome.

The heading and paragraph text should now align.

Flexible Padding: Using Percentage Amounts

Currently the header and 3 blue bordered sections below all have 20px of padding. The header has plenty of padding on small screens, but we'd like more on large screens. While we could add media queries to increase the padding, let's try something easier: a flexible percentage amount instead of a fixed pixel amount.

1. Back in **main.css**, in the **header** rule (near the bottom), replace the top and bottom padding values with a single padding as shown below:

```
header {  
  background: url(../img/masthead-darkened.jpg) center / cover;  
  color: #fff;  
  text-align: center;  
  padding: 7%;  
}
```

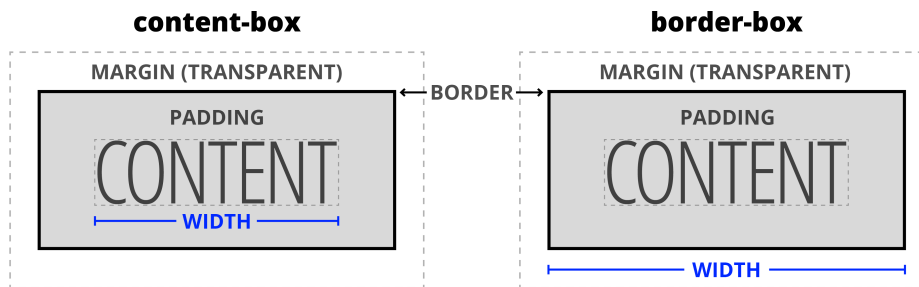
NOTE: This will take 7% of the width of the element and add that much padding to all 4 sides.

2. Save **main.css** and reload **index.html** in Chrome. Notice the following:

- Resize the browser and notice the header has less padding (space inside) when the window is narrow, and more padding when the window is wide.
- When the window is wide enough, the header is wider than the blue bordered section below it.

The header and content below all have an 800px max-width, so why can the header get wider than the sections? To understand, let's look at how the default box model (called **content-box**) works. CSS width is applied to the content. Padding and borders are then added, to end up with the final visual size for a box. That means a width of 800px plus 20px padding (on both sides) plus 4px borders (on both sides) ends up forming a box that is visually 848px ($800 + 20 \times 2 + 4 \times 2$).

Because padding is added to the max-width, the header ends up being wider than the sections below. Luckily there's a newer box model (called **border-box**) that works differently. As illustrated below, with **border-box** the CSS width is applied to the visual size of the box, including the borders and padding!



Switching to Border-Box

1. Still in Chrome, **Ctrl-click** (Mac) or **Right-click** (Windows) on the header's photo at the top of the page and choose **Inspect**.
2. Make sure the window is wide enough so you can see the header is wider than the section below.
3. Make sure the **header** element is selected in the **Elements** panel in the DevTools.
4. In the **Styles** panel of the DevTools, the top style should be **header**.

Go down the list of styles until you find the **header, main, footer** rule that we want to edit.

CSS Box Model: Content-Box vs. Border-Box

5. The last property in the **header, main, footer** rule should be **margin**. Click once on the **auto** value.

- Hit **Tab** to add a new property below that.
- Type **box-sizing** and then hit **Tab** to jump its value.
- Type **border-box** and hit **Tab** to apply it so you end up with the following:

```
header, main, footer {  
  max-width: 800px;  
  margin: ▶ auto;  
  box-sizing: border-box;  
}
```

6. In the page notice the following:
- The header got smaller, because the padding and border are now inside the 800px (instead of being added to it as it had been).
 - The header and sections are now the same width, which is what we want!
7. Now we must do this in our CSS file, so return to **main.css** in your code editor.
8. We could add the same code we just did in Chrome, but we really want to use this new box model for everything in the page. We can target all elements using the asterisk (*) selector.

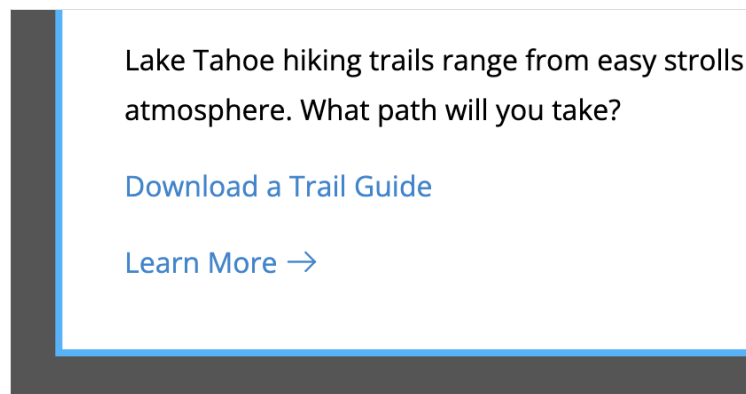
Above all the other rules, add this new rule:

```
* {  
  box-sizing: border-box;  
}
```

9. Save **main.css** and reload **index.html** in the browser.
- The header and content below are back to being the same width (which is what we want).
 - You should include this code in every site you create!
-

SVG Sizing (Width & Height) and Embedding SVG

Exercise Preview



Exercise Overview

In this exercise you'll learn more about SVG, such as how it's sized and that you can do more than just use SVG as a linked image. If you embed the SVG code into HTML, you gain some interesting new functionality.

Getting Started

1. If you completed the previous exercise, **index.html** should still be open from the **Tahoe** folder.

If you did not do the previous exercise, open the **Tahoe Ready for SVG** folder in your code editor and open **index.html**.

2. Preview **index.html** in Chrome.

We want to change the colored logo to a different white version.

3. Return to **index.html** in your code editor.
4. In the **nav**, change the image **src** by adding **-white** as shown below in bold:

```

```

5. Save **main.css** and reload **index.html** in Chrome.

Wow, this new logo is huge! Let's see why and how to fix it.

Sizing SVG

When exporting SVG from Adobe Illustrator, there's a **Responsive** option that controls the size. When **Responsive** is **not checked** it **includes** width and height code into the SVG file (as you saw with the first color logo that came in at the size we created it). When **Responsive** is **checked** it **does not include** width and height code (as you saw with the large white logo) so the SVG scales to fill whatever it's inside. That can be undesirable because we always have to code a size (via HTML or CSS), instead of it defaulting to the size it was created at.

Therefore, when exporting SVG files from Illustrator we typically recommend unchecking the **Responsive** option. For this logo we can edit the SVG code to add the width and height, instead of resaving it with Illustrator.

NOTE: Figma and Sketch code width and height into SVG files.

1. Let's look at the SVG code. In your code editor, open **tahoe-logo-white.svg** from the **img** folder.

Don't be overwhelmed by all the coordinates for the vector points. We're only going to be concerned with one small part of this file.

NOTE: SVG files are text files coded with XML (EXtensible Markup Language). XML and HTML are both markup languages, so the syntax is the same. You typically create SVG files using a graphics app, but they can be hand coded.

2. In the **svg** tag (the second tag) find the **viewBox="0 0 127.5 52.36"** part, which tells us the SVG is 127.5 pixels wide by 52.36 pixels tall.

NOTE: The viewBox numbers are coordinates:

- The top-left corner of the box starts **0 from the left** and **0 from the top**
- The bottom-right corner ends **127.5 px from the left** and **52.36 px from the top**.

3. We need to add width and height attributes, so add the folding **bold** code:

```
viewBox="0 0 127.5 52.36" width="127.5" height="52.36">
```

4. Save **tahoe-logo-white.svg** and close it.

5. Return to Chrome and reload **index.html**.

- The logo should now be smaller, which is the size it was created in Illustrator.
- If this was the size we wanted, we'd be done. We can override the original size though, so let's see how.

6. Return to your code editor.

7. Open **main.css** from the **css** folder.

SVG Sizing (Width & Height) and Embedding SVG

8. There are no other images in the nav so that will make it easier to target it with CSS. After the **nav** rule, add the following new rule:

```
nav img {
  width: 115px;
}
```

9. Save **main.css** and reload **index.html** in Chrome. Notice the logo is even smaller.

With SVG files you can set any width you want, smaller or larger, pixel or percentage, etc. without losing quality. That's the beauty of vectors!

Creating SVG Files

Any good web design app will let you export an SVG file. Here are things to know about a some ones:

- **Adobe Illustrator:** For detailed tips on saving SVG files in Illustrator visit tinyurl.com/ai-save-svg
- **Figma & Sketch:** Use the standard method of exporting.

Positioning the Arrow

1. Scroll down the page and in the **Take a Hike** section notice the **Learn More** link has a black arrow on the right.
 - It's too far down, needs some space between it and the text, and we don't like that it's black.
 - In an earlier exercise we aligned all images to the bottom to prevent extra space below them, so that's why the arrow is that far down. Let's fix the positioning and then change the color.

2. Switch back to **index.html** in your code editor.

3. Add a **button-secondary** class to all 3 **Learn More** links:

```
<p><a href="#" class="button-secondary">Learn More</a></p>
```

NOTE: Don't forget to do this to all 3 of these links!

4. In **main.css** below all the other rules, add the following new rule:

```
.button-secondary img {
  vertical-align: baseline;
  margin-left: 5px;
}
```

5. Save **main.css** and reload **index.html** in Chrome. Now the arrow is better positioned.

Embedding SVG (Instead of Linking)

While we could edit the SVG directly to change the color, if we use CSS we'll have greater flexibility, such as changing the color on hover! To be able to style SVG with CSS though, it can't be used as an `img` tag, the SVG code must be embedded into the HTML.

1. Return to your code editor.
2. Open **arrow.svg** from the **img** folder.
3. Look over the code and notice the following:
 - There are three elements that have an **ID** (**arrow**, **line**, and **tip**) which correspond to the layer/object name in the graphics app that created these.
 - The **arrow** contains two nested elements: **tip** and **line**.

Our webpage has three **Learn More** buttons, so we'll need to use this arrow three times. An **ID** can only be used once within a page, so we need to change the IDs to **classes** (which can be used multiple times).

4. Open the Find and Replace feature (your menu options may be named differently depending on your code editor). In Visual Studio Code, choose **Edit > Replace**.
5. Enter the following (the options may be named differently in your code editor):

Find: **id=**

Replace: **class=**

Proceed to **Replace All**, and a total of 3 replacements should be made.

6. Save this file (**arrow.svg**).
7. Select all code.
8. Copy it.
9. Close this file (**arrow.svg**).
10. Switch to **index.html** here in your code editor.
11. Around line 35 select the **img** tag for **arrow.svg**
 - Hit **Cmd-D** (Mac) or **Ctrl-D** (Windows) twice to select the other 2 matching image tags.
 - Paste the SVG code. It will paste it in all 3 places!
12. Save the file.

SVG Sizing (Width & Height) and Embedding SVG

- Return to Chrome and reload **index.html**.

In the **Take a Hike** section, after the **Learn More** link you should still see the black arrow, but it's lost the space between the text. That's because our CSS targets an **img** tag, not the **svg** tag we now have. Let's fix that.

- Return to **main.css** in your code editor.

- In the **.button-secondary img** selector, change **img** to **svg** as shown below:

```
.button-secondary svg {
```

- Save **main.css** and reload **index.html** in Chrome.

The arrow should once again have some space between it and the text.

Styling SVG Using CSS

- Switch back to **index.html** in your code editor.
- Look at the code for any of the three SVG arrows and notice:
 - The **line** and **tip** elements each have **stroke="#000"** which makes the black. SVG elements can have **fill**, **stroke**, and other settings. These are lines, so they have a stroke, but no fill.
 - Those two elements are nested inside a container **svg** element.
- We can use CSS to style SVG elements, so switch to **main.css**.
- We currently have a rule that targets the **svg** element, but that's not the element that has the stroke. We'll have to target all the elements inside the **svg**.

After the **.button-secondary svg** rule, add the following new rule:

```
.button-secondary svg * {
  stroke: red;
}
```

- The asterisk (*) selector targets all elements (in this case all elements within the **svg**, which are in a **button-secondary** class).
 - The **stroke** property will override the stroke attribute in the SVG. If you needed to change fill color, you'd use the **fill** property.
- Save the file and reload **index.html** in Chrome.
- The 3 arrows should now be red! It's good to see our CSS works, but let's use the same blue as the text links.
- Switch back to **main.css** in your code editor.

7. We could look up the hex color we used on the text links, but there's an easier and better way. As shown below, change **red** to **currentColor**:

```
.button-secondary svg * {  
  stroke: currentColor;  
}
```

NOTE: CSS values are typically hyphenated (like **current-color**). The capitalization style of **currentColor** comes from the original SVG specifications it was derived from. CSS is not case sensitive, so you write it as **currentColor** or **currentcolor**.

8. Save the file and reload **index.html** in Chrome to see:
 - The arrow should be the same color as the **Learn More** link!
 - The arrow is a bit too thick when compared to the text.
9. Switch back to **index.html** in your code editor.
10. In the SVG code for the **line** and **tip** elements, notice the **stroke-width="2"**
While we could change the attributes here, we can also change them via CSS!

11. Switch to **main.css**.
12. As shown below, add a new CSS property to override the SVG's attribute:

```
.button-secondary svg * {  
  stroke: currentColor;  
  stroke-width: 1.2;  
}
```

13. Save the file and reload **index.html** in Chrome to see the arrows are thinner, which between matches the thickness of the text.

Adding a Hover

Let's change the color of the **Learn More** link on hover and see what happens to the SVG arrow.

1. Switch back to **main.css** in your code editor.
2. Above the **.button-secondary svg** rule, add the following new rule:

```
.button-secondary:hover {  
  color: red;  
}
```

3. Save the file and reload **index.html** in Chrome.

SVG Sizing (Width & Height) and Embedding SVG

4. Hover over a **Learn More** link to see the color of the text and the SVG arrow both change! That's because the arrow still picks up on the current color!

Users that keyboard navigate a page (such as tabbing through elements), won't see this hover effect. It would be nice if they saw the same appearance when this link is selected (called focus), so it's often a best practice to include a **focus** style whenever you have a **hover** style.

5. Switch back to **main.css** in your code editor.
6. In the **.button-secondary:hover** selector add **.button-secondary:focus** and don't miss the comma after hover!

```
.button-secondary:hover,  
.button-secondary:focus {
```

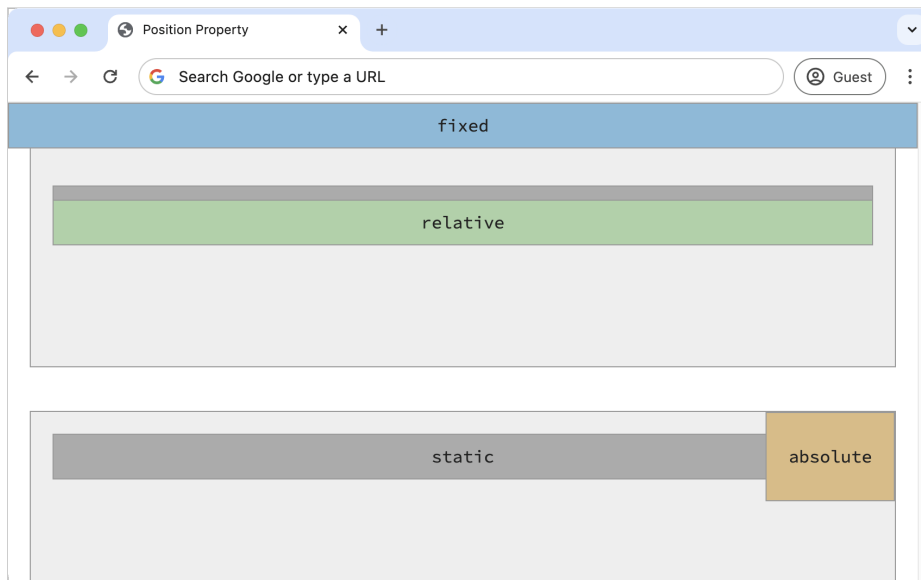
7. Save the file and reload **index.html** in Chrome.
8. Hit **Tab** to see the logo at the top becomes highlighted.
9. Keeping hitting **Tab** until the **Learn More** link is highlighted.

Now you should see it looks the same as when you hover. Nice!

Naming SVG Elements

- Many graphics apps turn object/layer names into IDs or classes in the SVG file they export. It's a good habit to name things before exporting SVG, and then look at the SVG code to make sure its clean.
- If you'll only be embedding SVG once on a page, you can keep the IDs (you don't have to change them to classes). Just be careful that the IDs do not conflict with other IDs you have used in HTML. To help ensure the IDs are unique, you could start each name with a unique prefix. For example, every element in a logo could start with **logo-** so the IDs would be **logo-something** and **logo-another-thing**

Exercise Preview



Exercise Overview

Browsers typically render page content in the order it's listed in the code. This means that the topmost markup will display first, followed by content listed later in the HTML. This is the normal document flow.

One way to take elements out of the normal document flow is using the **position** property, which you'll explore in this exercise. It opens up a world of "outside the box" possibilities.

Static Position & the Normal Document Flow

1. In your code editor, close any files you may have open.
2. For this exercise we'll be working with the **Position Property** folder located in **Desktop > Class Files > Advanced HTML CSS Class**. Open that folder in your code editor if it allows you to (like Visual Studio Code does).
3. Open **position.html** from the **Position Property** folder.
4. Preview the page in Chrome (we'll be using its DevTools).

The page is a playground to help you gain an understanding of the position property, in order to create interesting layouts.

5. Scroll down the page and note the following:

- There are 3 parent divs. In the CSS we gave them a light gray background, a border, and other basic styles.
- Inside each parent div, there are 2 divs: one colorful one and one darker gray div labeled **static**. Currently our CSS specifies only a different background color for each of the child divs. We haven't yet specified any positioning for the child divs.

The darker gray child divs labeled **static** are there to demonstrate the position property's default value. There is no need to declare this unless you are overriding a different value. It simply tells an HTML element to follow the normal document flow based on the order of the markup from top to bottom. All in all, it's a very reliable default. This makes it the perfect control for the experiments we'll perform on the more colorful divs.

6. The first value we'll test is **relative**. Scroll to the top of the page if you aren't already there. Leave the page open in Chrome so we can come back to it later.

Relative Position

1. Switch to your code editor.
2. Open **main.css** from the **css** folder.
3. In the **.relative-div** rule, declare a relative position:

```
.relative-div {  
  background: #aad2a6;  
  position: relative;  
}
```

4. Save **main.css**.
5. Switch back to Chrome and reload **position.html**. Hmm, nothing has changed.

Declaring a relative position by itself does nothing, but it enables a set of CSS properties that would otherwise have no effect. Specifying **top**, **right**, **bottom**, and/or **left** properties allow us to move any element that is not statically-positioned. The value we're using is named **relative** because these coordinates are determined **relative to** the element's natural position in the normal document flow.

6. Let's set a **top** coordinate and see how the div moves in accordance with it. Switch back to **main.css** in your code editor.

CSS Position Property

7. Let's try to move the div down by 75 pixels. To do that, add the following bold code to the **.relative-div**:

```
.relative-div {  
  background: #aad2a6;  
  position: relative;  
  top: 75px;  
}
```

8. Save **main.css** and reload **position.html** in Chrome. The Relative div has moved down 75 pixels. But why is it mostly covering the static div?

Relatively-positioned elements are unique in that they can move around the page visually but they remain anchored in the normal document flow. They “remember their home” because that’s where they get their base position from. Static elements are a little clueless in that they can’t “see” where a relative element has wandered off to. However, they do remember where the relatively-positioned element should be. This is why the static div remained in the same position as if the relative div never moved.

9. Let's take a closer look in Chrome's Developer Tools. **Ctrl-click** (Mac) or **Right-click** (Windows) on the green **relative** bar and choose **Inspect** to open the DevTools.

10. In the **Styles** tab of DevTools, find the **.relative-div** rule.

- Try checking **position: relative;** off and on to see that the top coordinate has no effect when it is checked off (which reverts to the default static position).
- Make sure the property declaration is checked on before continuing.

11. Click on the **top** property's value (**75px**) to highlight it.

- Use the **Up Arrow** on the keyboard to increase the value **1** pixel at a time. If you add the **Shift** key, you can modify the value **10** pixels at a time. Use the **Down Arrow** to decrease the value.
- Watch how the top coordinate is updated live in the browser window. If you push it far above or below its starting position, you'll see that it can pop right out of its containing element.

12. Let's see what it would look like if it moved over horizontally. Still in the **Styles** tab of the DevTools, click below the **top** property declaration to create a new empty line (so we can add more CSS).

13. Type **left** and hit **Tab**. Then type **100px**

- Use the **Up** and **Down Arrow** keys to modify the left position.

Positive vs. Negative Values

The top, right, bottom, and left properties can accept both positive and negative values. Positive values move an element **in the opposite direction** from the name of the property. For instance, we gave the top property a positive value, which will push the div downward. Likewise, a positive left position would move the element to the right.

Absolute Position

You can move a relatively-positioned element anywhere on the page, but because it leaves a gap in its native position in the normal document flow, it's best not to position these elements too far from "home". If you want an element to really move around, you are going to want to use **absolute** positioning instead.

1. Still in Chrome, scroll down to the next section of the page that shows the **absolute** label.
2. Switch back to **main.css** in your code editor.
3. In the **.absolute-div** rule, add the following bold code:

```
.absolute-div {  
  background: #dcb81;  
  position: absolute;  
}
```

4. Save **main.css**.
5. Switch back to Chrome and reload **position.html**.
 - The static div has moved up to where the absolute div used to be.
 - The absolute div width changed, collapsing to be hug the content inside it. Absolutely-positioned elements are **removed from the normal document flow** and other elements are no longer aware of it. This works well for elements we want to move anywhere in the document.
6. Switch back to **main.css**.

CSS Position Property

7. Absolutely-positioned block-level elements collapse but they are still blocks. This means you can still add width, margin, padding, etc. Let's try adding some **padding** as shown below in bold:

```
.absolute-div {
  background: #dcbb81;
  position: absolute;
  padding: 20px;
}
```

8. Save **main.css** and reload **position.html** in Chrome. The "box" has gotten bigger and looks a lot less cramped. But it still hasn't moved from its natural starting point.
9. Let's move the element down a tad. Switch back to **main.css** in your code editor.
10. Add the following bold code:

```
.absolute-div {
  background: #dcbb81;
  position: absolute;
  padding: 20px;
  top: 40px;
}
```

11. Save **main.css** and reload **position.html** in Chrome.

The div did position itself 40 pixels from the top, but from the top of the page! So how can we set which element this will be 40 pixels from the top of?

The Dynamic Duo: Relative Parent, Absolute Child

Absolute-positioned elements are removed from their original position in the normal document flow. The browser looks to the nearest parent element that has its position declared. If there is no parent element with a position declared (as is the case here), the absolute div is positioned in relation to the **<html>** element (the document itself). This is why our div is positioned from the top of the browser window.

Whenever you declare an **absolute** position, you should think "I absolutely need to determine where this element gets its coordinates".

1. Switch to **position.html** in your code editor so we can find a suitable parent element.
2. Locate the markup for the content we're working with:

```
<div class="parent">
  <div class="absolute-div">absolute</div>
  <div class="static-div">static</div>
</div>
```

The **.parent** element looks like a great choice to set a position property on.

3. Switch to **main.css**.
4. In the **.parent** rule, declare a **relative** position as shown below in bold:

```
.parent {
```

CODE OMITTED TO SAVE SPACE

```
    position: relative;
```

```
}
```

5. Save **main.css** and reload **position.html** in Chrome. Ah! Now the absolute div is 40 pixels from the top of its parent container.
6. Let's experiment some more. Switch back to **main.css** in your code editor.
7. Change the **.absolute-div**'s **top** coordinate and add **right** (as shown in bold below):

```
.absolute-div {  
    background: #dcbb81;  
    position: absolute;  
    padding: 20px;  
    top: 0;  
    right: 0;  
}
```

8. Save **main.css** and reload **position.html** in Chrome. Now the absolute div should be at the top right of parent.

Fixed Position

Let's look at one more kind of positioning... **fixed**. Unlike the other positions value, fixed elements only take orders from the **viewport** (the browser window itself).

By default, the content on a page moves when you scroll. However, the viewport does not budge when you scroll through a page. This means whenever you declare a fixed position, that element will stay put no matter what happens on the page itself.

1. Still in Chrome, scroll down to the bottom-most section of the page, **fixed**.
2. Switch back to **main.css** in your code editor.
3. To see this in action, find the **.fixed-div** rule and add the following code:

```
.fixed-div {  
    background: #84bada;  
    position: fixed;  
}
```

4. Save **main.css** and reload **position.html** in Chrome. The static div has moved up to fit where the fixed div once was. Clearly the fixed div has been removed from the document flow. But where did it go?

CSS Position Property

5. If you have a tall enough display and can see the fixed div, notice it remains in its original position, but does not scroll with the other page content. If your display is too short, you won't be able to scroll down to see it.

We don't want it to remain in its original position in the page, so we need to specify the distance from the edges of the viewport.

6. Switch back to **main.css** in your code editor.
7. Position the **.fixed-div** to the very top of the browser window by adding the following bold code:

```
.fixed-div {  
  background: #84bada;  
  position: fixed;  
  top: 0;  
}
```

8. Save **main.css** and reload **position.html** in Chrome.
 - It's now sitting flush with the top of the browser window!
 - Scroll through the page to see that the fixed div does not scroll with the other page content.
 - The fixed div has also collapsed down to the hug its content. Let's expand the element so it stretches across the entire browser window, mimicking a common feature on modern websites: a navbar that stays put as you scroll.

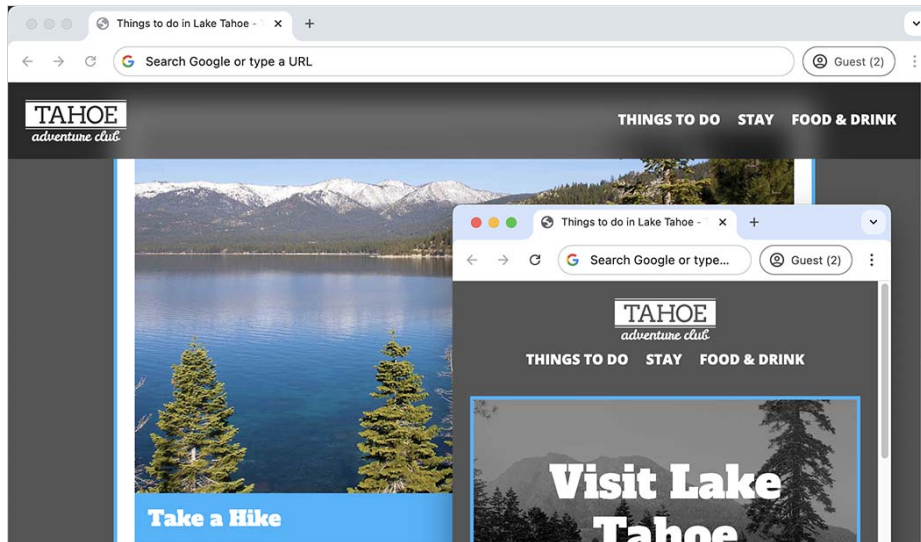
9. Back in **main.css**, add the following bold coordinates:

```
.fixed-div {  
  background: #84bada;  
  position: fixed;  
  top: 0;  
  left: 0;  
  right: 0;  
}
```

10. Save **main.css** and reload **position.html** in Chrome. Awesome, our header stretched out to fill the width of the screen.
-

Fixed Position Navbar, RGBA, & Backdrop Filters

Exercise Preview



Exercise Overview

In this exercise, you'll create a fixed position navbar. We don't want to waste valuable screen space on small screens, so we'll only fix the navbar on larger screens. We'll use media queries to style the navbar appropriately for different screen sizes.

Getting Started

1. If you completed the last **Tahoe** exercise, reopen **index.html** from the **Tahoe** folder.

If you did not do the last **Tahoe** exercise, open the **Tahoe Ready for Fixed Navbar** folder in your code editor and open **index.html**.

2. Preview **index.html** in a browser.

Because we've built this page with a mobile-first approach, the navbar on small screens is how we want it. We want to change the layout for larger screens though.

Altering the Navbar Appearance on Larger Screens

1. Switch back to your code editor and open **main.css** (in the **css** folder).

2. Let's start by adding a dark background. At the bottom, below all the other rules add the following:

```
@media (min-width: 600px) {  
  nav {  
    background: #000;  
  }  
}
```

3. Save the file and reload the page in your browser.

When the page is 600px or wider, the nav should now have a black background. Let's put the logo on the left and the nav links on the far right.

4. Switch back to **main.css** in your code editor.

5. Still working in the media query, add the following bold code:

```
nav {  
  background: #000;  
  display: flex;  
  align-items: center;  
}
```

6. Save the file and reload the page in your browser.

- The logo and nav links should now be on a single line.
- We need some padding so the content doesn't touch the edge of the black.

7. Switch back to **main.css** and add the following bold code:


```
nav {  
  background: #000;  
  display: flex;  
  align-items: center;  
  padding: 20px;  
}
```

8. Save the file and reload the page in your browser.

Now we just need to move the links to the far right.

Fixed Position Navbar, RGBA, & Backdrop Filters

- Switch back to **main.css** and in the **min-width: 600px** media query, add the following new rule:

```
@media (min-width: 600px) {
  nav {
    
  }
  nav ul {
    margin-left: auto;
  }
}
```

- Save the file and reload the page in your browser.
 - Resize the page to see the mobile nav is still centered with no background, and on larger screens we now have this new layout with a background.

Making the Navbar Fixed Position on Larger Screens

We want the navbar to be fixed to the top of the browser window so it remains visible even as the user scrolls down. This can waste valuable screen space on small screens, so we'll only do this on wider screens.

- Switch back to **main.css** and in the **min-width: 600px** media query add the following:

```
nav {
  background: #000;
  display: flex;
  align-items: center;
  padding: 20px;
  position: fixed;
}
```

- Save the file and reload the page in your browser.
 - Scroll down the page to see that the fixed header stays at the top of the page but we have some issues to solve.
 - The fixed header has collapsed down hug the content, so it no longer stretches across the page like we want.
 - The **Visit Lake Tahoe** section that was previously below the header has moved up as far as it can, because the navbar was removed from the normal document flow.

3. We'll solve the width issue by adding some positioning information. Switch back to **main.css** and in the **min-width: 600px** media query add the following:

```
nav {  
  CODE OMITTED TO SAVE SPACE  
  position: fixed;  
  top: 0;  
  left: 0;  
  right: 0;  
}
```

4. Save the file and reload the page in your browser.
- The navbar should now full stretch across the top of the page.
 - Let's make the background color partially transparent so we can see the content scroll behind it.
5. Switch back to **main.css** and in the **min-width: 600px** media query change the background color:

```
nav {  
  background: rgba(0,0,0, .5);  
  CODE OMITTED TO SAVE SPACE  
}
```

6. Save the file and reload the page in your browser.
- Scroll down and notice how you can see the content through the semi-transparent navbar.
 - When the page first loads, the **Visit Lake Tahoe** section (at the top of the page) gets covered up by the navbar. To prevent that, let's add some top margin to the body, which will push that content farther down the page.
7. Back in **main.css**, add the following bold code at the beginning of the media query:

```
@media (min-width: 600px) {  
  body {  
    margin-top: 130px;  
  }  
  nav {
```

Fixed Position Navbar, RGBA, & Backdrop Filters

8. Save the file and reload the page in your browser.
 - You'll probably need to scroll up to see the change. That's because it kept the previous scroll position which won't happen when loading the page normally.
 - The **Visit Lake Tahoe** section now starts below the navbar.
 - Resize the window in until you see the narrow mobile layout. Scroll down the page to see that the header does not stay fixed on narrow screens. Great!

Blurring & Desaturating the Navbar's Background

When content scrolls behind the navbar it makes the navbar text harder to read. We can improve this by blurring the content behind the navbar.

1. Back in **main.css**, in the **min-width: 600px** media query add the following code to the **nav** rule:

```
nav {
  -webkit-backdrop-filter: blur(15px);
  backdrop-filter: blur(15px);
  background: rgba(0,0,0, .5);
```

CODE OMITTED TO SAVE SPACE

```
}
```

2. Save the file and reload the page in your browser.
 - Scroll down to see the background is blurred behind the navbar. Nice!
 - We like this, but having colors behind the navbar (even though they're blurred) might still be too distracting. There's another filter that can change the saturation. Let's desaturate the color completely to make it black and white.
3. Back in **main.css**, add the following code shown in bold:

```
nav {
  -webkit-backdrop-filter: blur(15px) saturate(0%);
  backdrop-filter: blur(15px) saturate(0%);
```

CODE OMITTED TO SAVE SPACE

```
}
```

4. Save the file and reload the page in your browser.

Scroll down to see the background is not only blurred, but it's also converted to black and white. Amazing!

Transparent Hex Colors

Hex colors originally could not be transparent, so RGBA was created.

RGBA (Red, Green, Blue, Alpha) has the same color spectrum as hex values. Each of the Red, Green, and Blue values uses a number between 0 and 255.

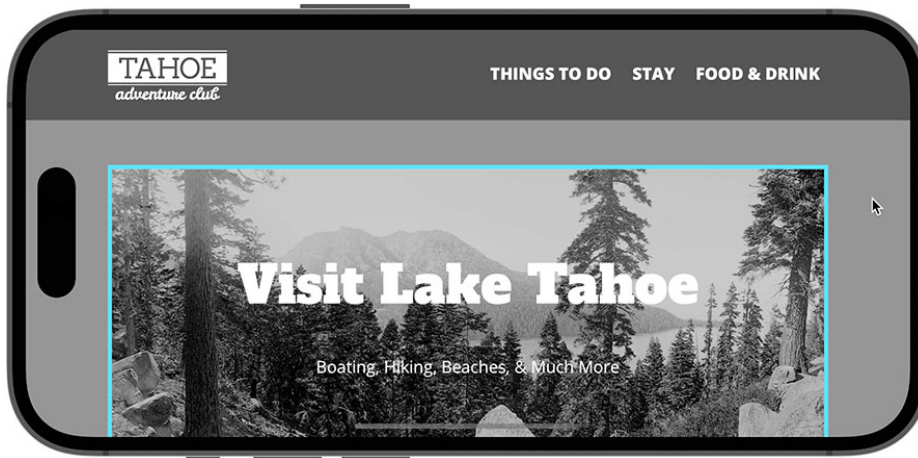
- When all values are 0, the color is black.
- When all values are 255, the color is white.
- **A** stands for **Alpha**, which specifies how transparent the color is. Alpha values range from 0 (fully transparent) to 1 (fully opaque). A decimal value is something in between (.5 is 50%).

Hex colors were later updated to support alpha (using 4 and 8 digit values instead of the original 3 and 6), but converting 0–100% alpha into hex is not as easy. So when specifying alpha we prefer RGBA. Here's an example:

- Black: #000000 or #000
- Black with 60% transparency: #00000099 or #0009

To learn more about hex opacity visit drawne.com/hex-opacity

Exercise Preview



Exercise Overview

In this exercise you'll make several improvements specifically for mobile devices.

Getting Started

1. If you completed the previous exercise, **index.html** should still be open from the **Tahoe** folder.

If you did not do the previous exercise, open the **Tahoe Ready for Mobile Improvements** folder in your code editor and open **index.html**.
2. Open **index.html** and preview it in Chrome.

Preventing Mobile Text Size Adjustment

Some mobile browsers enlarge text they think is too small (if they think it doesn't break the layout). We don't want browsers arbitrarily overriding some font sizes, so let's disable that. We do this on every website we make.

1. Go into the **snippets** folder and open **text-size-adjust.css**.
2. Hit **Cmd-A** (Mac) or **Ctrl-A** (Windows) to select all the code.
3. Hit **Cmd-C** (Mac) or **Ctrl-C** (Windows) to copy it.
4. Close the file.

5. Open **main.css** (in the **css** folder) and above the **body** rule paste the new code:


```
html {
  -moz-text-size-adjust: 100%;
  -webkit-text-size-adjust: 100%;
  text-size-adjust: 100%;
}
```

6. Save the file.


We don't need to preview this change. Even though we shouldn't see anything change, this code should be added to all websites to ensure we have complete control over font-size and browser's won't do something we don't tell them to.

Preventing a Fixed Navbar on Short Screens

1. Preview **index.html** in Chrome.

- **Ctrl-click** (Mac) or **Right-click** (Windows) anywhere on the page and select **Inspect** to open Chrome's DevTools.
- In the upper left of the DevTools panel click the **Toggle device toolbar** button  to enter device mode.

2. From the **Dimensions** menu in the toolbar above the page, select **iPhone SE**.

- To the right of that same toolbar, click the **Rotate**  button to simulate holding the phone horizontally.
- Scroll down and notice the navbar remains on screen.

Having a fixed navbar takes up a lot of vertical space on a short screen. Let's only make it so the navbar is only fixed on taller screens.

3. Switch back to **main.css** in your code editor.
4. In the **min-width: 600px** media query, change the **nav** position from **fixed** to **absolute**.

```
nav {
  CODE OMITTED TO SAVE SPACE
  position: absolute;
  CODE OMITTED TO SAVE SPACE
}
```

5. Save the file and reload the page in Chrome. Scroll up/down to see the navbar now scrolls with the page content.


Improvements for Mobile Devices

6. While most media queries focus on the screen **width**, they can look at screen **height**.

Back in **main.css** add this new media query at the bottom below all the other code. Pay close attention to **width** and **height** in this code!

```
@media (min-width: 600px) and (min-height: 750px) {
  nav {
    position: fixed;
  }
}
```

7. Save the file and reload the page in Chrome.

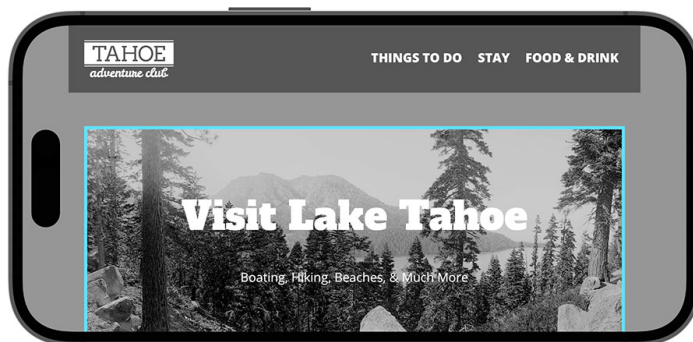
In the DevTools click the **Toggle device toolbar** button  to leave device mode.

Close the DevTools and resize the window's height to see:

- The navbar scrolls with the page content when the window is short.
- The navbar does not scroll (it's fixed) when the window is tall.

Dealing with the iPhone's Dynamic Island (or Notch)

1. As shown below, currently the iPhone's dynamic island (or notch) creates extra empty space on the left and right of the page, so it won't cover the content.



While this works, we sometimes want parts of the page to cover the entire width.

2. Chrome's DevTools do not simulate the iPhone's dynamic island (or notch), so the only way is Mac users can install Xcode (free) and use the Simulator app. Instructions for doing that were in the **Before You Begin** section at the beginning of this book.

If you want to see this page in the Simulator app:

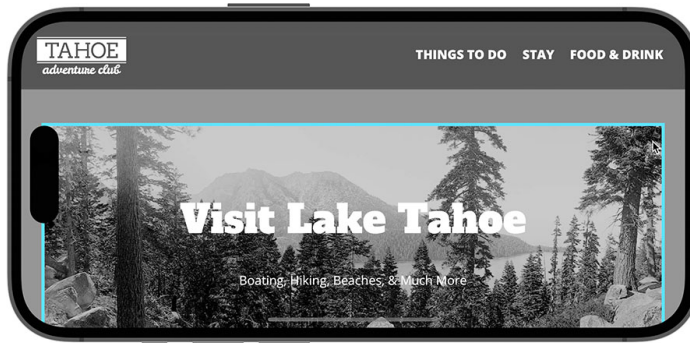
- In Chrome copy the URL (file:///Users...)
- In the Simulator, click into the URL bar, and then click **Paste and Go**.
- There's a rotate button in the titlebar of Simulator's window.

If you're on Windows or can't install Xcode, you can follow along with this exercise and we'll show you screenshots of what the code does.

3. Back in your code editor, go into the **snippets** folder and open **iphone-dynamic-island.txt**.
4. Copy the **viewport-fit=cover** code.
5. Switch to **index.html** and paste it into the viewport meta tag as follows (be sure to add the comma before it):

```
<meta name="viewport" content="width=device-width, initial-scale=1.0, viewport-fit=cover">
```

6. Save the file. If you could reload the page in the Simulator this is what you'd see:



- Now the content covers the entire screen and can go behind the island/notch. That's fine for backgrounds, but not content.
 - Next we'll have to add some padding back in (wherever appropriate) to push the content back into the page so it's not under the island/notch.
7. In your code editor, switch back to **iphone-dynamic-island.txt** (in the **snippets** folder).
 - Copy all the lines of the **padding** code up to and including the ending semicolon ;
 8. Switch to **main.css** and in the **body** rule, paste the CSS so you get:

```
body {
```

CODE OMITTED TO SAVE SPACE

```
padding:
```

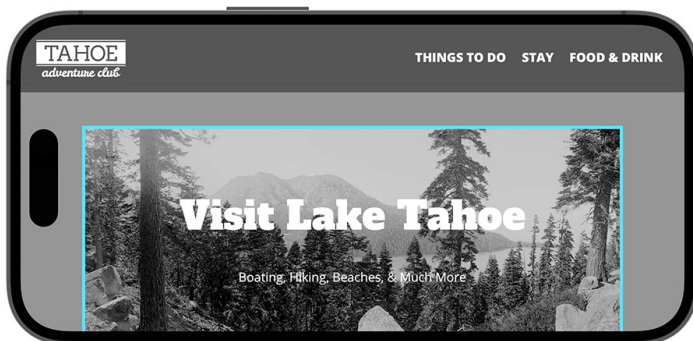
```
env(safe-area-inset-top)
env(safe-area-inset-right)
env(safe-area-inset-bottom)
env(safe-area-inset-left)
```

```
;
```

```
}
```

Improvements for Mobile Devices

9. If you could reload the page in the Simulator this is what you'd see:



- The main page content has the correct padding again, but the nav needs padding inside its background color.

10. In the **min-width: 600px** media query, find the **nav** rule and:

- Delete the **padding: 20px;** that is already there.
- Then paste the new padding code again so you get:

```
nav {
```

CODE OMITTED TO SAVE SPACE

```
padding:
```

```
  env(safe-area-inset-top)
  env(safe-area-inset-right)
  env(safe-area-inset-bottom)
  env(safe-area-inset-left)
```

```
;
```

CODE OMITTED TO SAVE SPACE

```
}
```

11. We need to add back the 20 pixels of padding (which we just removed) to this inset padding. Start by replacing the **top** and **bottom** values with **20px** as shown below:

```
padding:
```

```
  20px
```

```
  env(safe-area-inset-right)
```

```
  20px
```

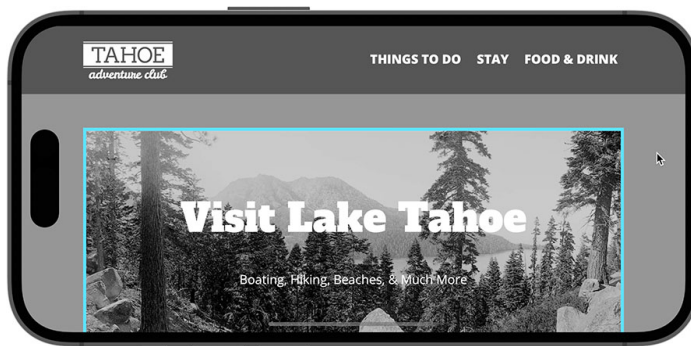
```
  env(safe-area-inset-left)
```

```
;
```

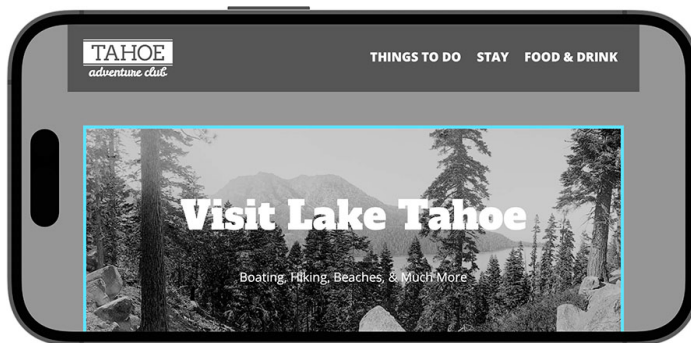
12. We need to add 20px to the left and right inset padding. To do math in CSS we have to use `calc()`. Wrap the `calc()` function around the `env()` and make sure you add the parenthesis at the end!

```
padding:
  20px
  calc( 20px + env(safe-area-inset-right) )
  20px
  calc( 20px + env(safe-area-inset-left) )
;
```

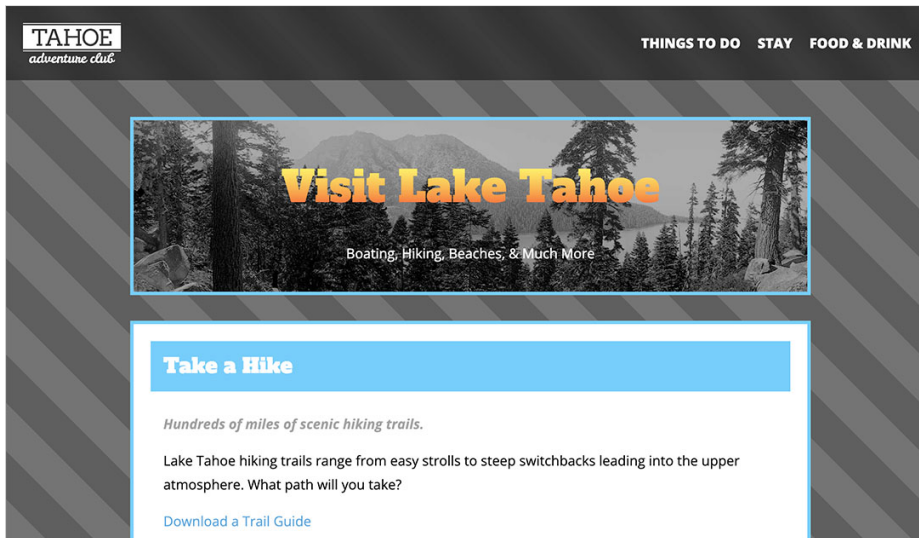
13. Save the file.
14. If you could reload the page in the Simulator this is what the final result looks like:



Compared to how we started:



Exercise Preview



Exercise Overview

In this exercise, you'll learn about CSS gradients. They can be used for smooth color transitions, or to create patterns such as stripes!

Getting Started

1. In your code editor, close any files you may have open.
2. We'll be working with the **Tahoe Gradients** folder located in **Desktop > Class Files > Advanced HTML CSS Class**. Open that folder in your code editor if it allows you to (like Visual Studio Code does).
3. Open **index.html** from the **Tahoe Gradients** folder.
4. Preview **index.html** in a browser.

This is the same page as we've been working with in previous exercises, but we removed some content to make it easier to see the background area that we'll be working on.

5. Leave the page open in the browser so we can come back to it and reload as we make changes.

Creating a CSS Background Gradient

CSS gradients are a function which creates an **image** of a gradient. The CSS gradient image has no set dimensions. Its size will match the size of the element it applies to.

1. Switch back to your code editor and open **main.css** from the **css** folder.
2. Let's have some fun with learning the syntax for CSS gradients. Add the following new background-image property declaration:

```
body {  
  background: #555;  
  background-image: radial-gradient(red, orange, purple);  
  
  CODE OMITTED TO SAVE SPACE  
}
```

NOTE: We are keeping solid gray background color because later we'll be making the gradient partially transparent so it shows through to the background color.

3. Save and preview **index.html** in a browser. Woah, trippy! Let's try a linear, rather than radial gradient.

4. Return to **main.css**.

5. Edit the gradient style as follows:

```
background-image: linear-gradient(red, orange, purple);
```

6. Save the file and reload the browser. The gradient fills the page and goes from top to bottom by default. Let's change the gradient to go from left to right.

7. Return to **main.css** and edit the gradient as follows:

```
background-image: linear-gradient(to right, red, orange, purple);
```

8. Save the file and reload the browser. Cool. Let's try to put this on an angle.

9. Return to **main.css** and edit the gradient as follows:

```
background-image: linear-gradient(45deg, red, orange, purple);
```

10. Save the file and reload the browser. Fancy that. And you can use any angle you like. What about specifying the size of each gradient color to create less of a rainbow gradation and more of a stripe effect? We can do that.

11. Return to **main.css**.

Creating a Striped Pattern

1. Let's start with red, and continue with red until 40px. Edit the gradient as follows:

```
background-image: linear-gradient(45deg, red, red 40px, orange, purple);
```

2. Next, set orange to start at 40px (exactly where the red ends) and continue to 80px:

```
background-image: linear-gradient(45deg, red, red 40px, orange 40px, orange 80px, purple);
```

Background Gradients & Text Gradients

3. Then set purple to start at 80px (exactly where the orange ends) and continue to 120px:

```
background-image: linear-gradient(45deg, red, red 40px, orange 40px, orange 80px, purple 80px, purple 120px);
```

4. Save the file and reload the browser. If needed, scroll to the bottom of the page to see the small red and orange stripes (the rest of the page should be purple). We want this to repeat as a pattern, not just appear as a single tiny stripe effect in the bottom-left corner.

5. Return to **main.css** and add **repeating** as shown below:

```
background-image: repeating-linear-gradient(45deg, red, red 40px, orange 40px, orange 80px, purple 80px, purple 120px);
```

6. Save the file and reload the browser. Yowza. Now that we know how to make stripes, let's tone this down a bit. Let's create a series of gray stripes by using the gray background of the page and alternating it with stripes that are a semi-transparent overlay.

7. Return to **main.css** and replace the color red with **transparent**, as follows:

```
background-image: repeating-linear-gradient(45deg, transparent, transparent 40px, orange 40px, orange 80px, purple 80px, purple 120px);
```

8. Save the file and reload the browser. You can see through to the gray background in these transparent areas. Great! We only want two colors for the pattern, dark gray and a not-so-dark gray. Let's get rid of the purple stripe to pare things down.

9. Return to **main.css** and **delete** the purple values as follows:

```
background-image: repeating-linear-gradient(45deg, transparent, transparent 40px, orange 40px, orange 80px);
```

10. Save the file and reload the browser.

This could work for a construction company... but we're getting closer. Now let's change the orange stripe to a partially transparent black stripe (which will darken the background gray).

11. Return to **main.css** and replace **orange** with the following RGBA values:

```
background-image: repeating-linear-gradient(45deg, transparent, transparent 40px, rgba(0,0,0, .2) 40px, rgba(0,0,0, .2) 80px);
```

12. Save the file and reload the browser.

- Now we have some nice looking stripes.
- If your screen is tall enough to show some space below the content, at some window widths you may notice a zig zag in the stripes just below the content where the pattern does not align. That's because the background does not fill the entire screen, it ends at the bottom of the content and then repeats.

13. Return to **main.css** and add the following code shown in bold:

```
body {  
  background: #555;  
  background-image: repeating-linear-gradient(45deg, transparent, transparent  
40px, rgba(0,0,0, .2) 40px, rgba(0,0,0, .2) 80px);  
  background-attachment: fixed;  
    
  CODE OMITTED TO SAVE SPACE  
}
```

14. Save the file, reload the browser, and:

- Notice the background gradient no longer repeats, because it's now relative to the viewport rather than the page's content.
- Make the window short enough so you can scroll.
- Scroll around to see the content moves over the background (which remains in a fixed position).

Text Gradients

CSS gradients are only for backgrounds, not text fill color. So to fill text with a gradient requires a little trickier CSS.

1. Return to **main.css** and in the **h1** rule add this CSS:

```
h1 {  
  font-size: 50px;  
  background: linear-gradient(yellow, red);  
}
```

2. Save the file, reload the browser, and you'll see the gradient behind the text.

We only want to see it in the text area, so we need to "clip" it to only show in the text area.

Background Gradients & Text Gradients

3. Return to **main.css** and in the **h1** rule add this CSS:

```
h1 {  
  font-size: 50px;  
  background: linear-gradient(yellow, red);  
  -webkit-background-clip: text;  
  background-clip: text;  
}
```

4. Save the file, reload the browser, and you'll see the gradient has disappeared.

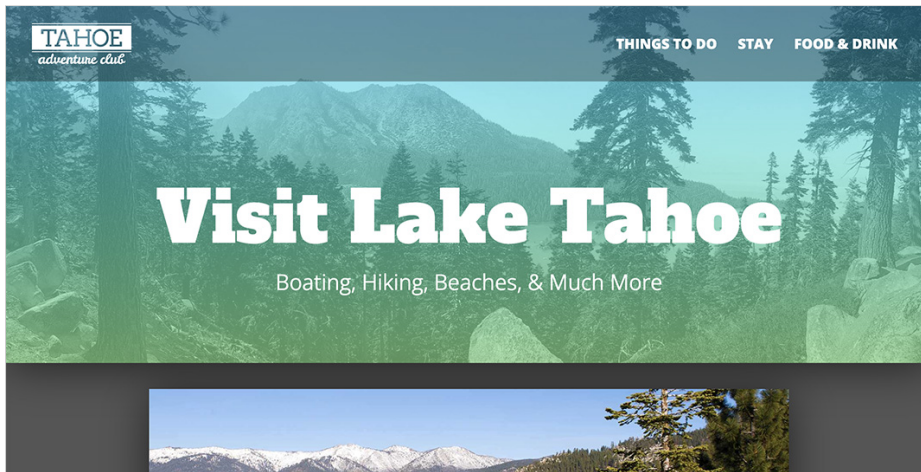
The gradient is hidden behind the white text, so the last step is to make the text color transparent so we can see through to the background gradient.

5. Return to **main.css** and in the **h1** rule add this CSS:

```
h1 {  
  font-size: 50px;  
  background: linear-gradient(yellow, red);  
  -webkit-background-clip: text;  
  background-clip: text;  
  color: transparent;  
}
```

6. Save the file, reload the browser, and you'll finally see the text gradient. Fire!
-

Exercise Preview



Exercise Overview

In this exercise, you'll learn how to add multiple backgrounds to a single element. You'll size type relative to the screen size (so type gets larger on larger screens).

Multiple Backgrounds

We are not limited to only a single background! We can use a comma separated list of backgrounds.

1. In your code editor, close any files you may have open.
2. We'll be working with the **Tahoe New Design** folder in **Desktop > Class Files > Advanced HTML CSS Class**. Open that folder in your code editor if it allows you to (like Visual Studio Code does).
3. Open **index.html** from the **Tahoe New Design** folder.
4. Preview **index.html** in a browser.

This is the same content we've been working with, but we made some design changes. Let's add a pattern over the header's background photo.

NOTE: We used some new concepts to create this design (such as drop shadows) which we'll cover later in this book.

5. Switch back to your code editor and open **main.css** from the **css** folder.
6. Find the **header** rule.

TIP: To quickly go to a style in Visual Studio Code, hit **Cmd-Shift-O** (Mac) or **Cmd-Shift-O** (Windows), start typing **header** and when the style in the list is selected hit **Return** (Mac) or **Enter** (Windows) to go to it.

7. Add the following code shown in bold. Don't miss the comma at the end!

```
header {
  background: url(../img/pattern.svg),
              url(../img/masthead.jpg) right top / cover;
  color: #fff;
}
```

NOTE: The order of backgrounds matter. The first will be on top, the second will be under that, the third would be behind that, and so on.

8. Save the file and reload the browser.

- You should see a blue diamond patten over the black and white photo.
- We know this doesn't look good, but we wanted this example to make it easy for you to see how the pattern is on top (because it's the first background image in the list), and the photo is behind that (because it's second in the list). Now let's make something that looks better!

Coloring a Background Image By Overlaying a Gradient

Because CSS gradients are background images, you can combine a background image and a background gradient on a single element.

1. Return to **main.css** and change the pattern to a gradient, using the following code shown in bold:

```
header {
  background: linear-gradient(rgba(0,201,255, 1), rgba(146,254,157, 1)),
              url(../img/masthead.jpg) right top / cover;
  color: #fff;
}
```

2. Save the file and reload the browser.

You should only see the gradient, because we made the alpha setting in our RGBA value 1, which means 100% opaque. Let's make it transparent so we can see through to the photo behind.

3. Return to **main.css** and change the two alpha values from **1** to **.6** as show in bold:

```
background: linear-gradient(rgba(0,201,255, .6), rgba(146,254,157, .6)),
              url(../img/masthead.jpg) right top;
}
```

4. Save and preview **index.html** in a browser.

Sweet, now you can see through the gradient to see the underlying background image. What a cool way to color an image!

Multiple Backgrounds, Viewport Sizing (vw), & Clamp

Sizing Type to the Viewport

1. Resize the window from a wide desktop size to a narrow mobile view, noticing the heading font size remains a fixed size.

On small screens the type is a bit big, and on large screens the type is a bit small. Instead of using a fixed pixel size, we can switch to viewport units which are relative to the size of the window/screen.

2. Return to your code editor.
3. In the **h1** rule, edit the font-size value as shown below in bold:

```
h1 {
  font-size: 11vw;
  margin: 10px 0;
```

CODE OMITTED TO SAVE SPACE

```
}
```

NOTE: How does this work? 11vw is like 11% of the viewport width. On a 320px screen, $320 \times .11 = 35.2\text{px}$ type. On a 1200px screen, $1200 \times .11 = 132\text{px}$ type. So the type will be larger on larger screens!

4. Save the file.
5. Switch back to the browser and reload. Resize the window from narrow to wide and notice how the font size changes in relation to the viewport size. It looks good at a mobile phone size, but huge at a desktop size!

When the window is larger than about 600px, the heading starts dominating the page too much. Let's reduce the size on larger screens.

6. Return to your code editor.
7. Towards the bottom of the file, inside the **min-width: 600px** media query add a new rule for the **h1** as shown below in bold:

```
@media (min-width: 600px) {
  h1 {
    font-size: 7.5vw;
  }
  main {
```

8. Save the file.
9. Switch back to the browser and reload.
 - Resize the window from mobile through desktop size to check out the type as the size changes.
 - The type size is good on small and medium screens, but is too big on wide screens.

Using Clamp to Set a Min & Max Font Size

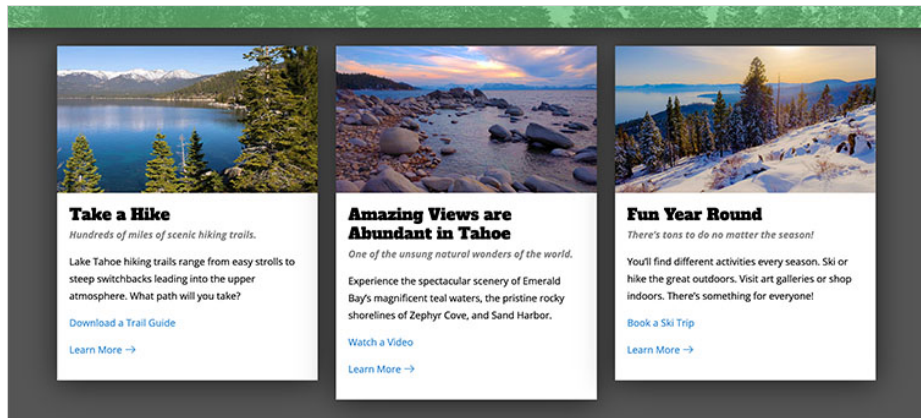
1. Return to your code editor.
2. Edit the font size of the **h1** rule you just made as shown below in bold:

```
@media (min-width: 600px) {  
  h1 {  
    font-size: clamp(50px, 7.5vw, 90px);  
  }  
  main {
```

NOTE: Clamp has 3 font sizes: minimum, preferred, and maximum

3. Save the file.
 4. Switch back to the browser and reload. Resize the window from mobile through desktop size and it should now look good from small to big!
-

Exercise Preview



Exercise Overview

In this exercise, you'll create columns using Flexbox and see some vertical alignment options.

Getting Started

1. If you completed the previous exercise, **index.html** should still be open from the **Tahoe New Design** folder.

If you did not do the previous exercise, open the **Tahoe New Design Ready for Columns** folder in your code editor and open **index.html**.

2. Preview **index.html** in Chrome.

Below the header photo are 3 sections. On wide screens let's make these into columns so users don't have to scroll as much to see the content.

3. Leave the page open so we can come back and reload as we make changes.

Creating a 3-Column Layout Using Flexbox

1. Switch back to your code editor and open **main.css** from the **css** folder.

2. Create a new media query for larger screens by adding the following code at the very bottom below all the other code:

```
@media (min-width: 1100px) {  
  main {  
    max-width: 1400px;  
    display: flex;  
  }  
}
```

3. Save the file and reload the page in Chrome.
 - On wider screens, the main content area should be wider with 3 columns.
 - There's too much space between and around the columns.

4. Back in **main.css**, remove the space by adding the following new rule:

```
@media (min-width: 1100px) {  
  main {  
  
    CODE OMITTED TO SAVE SPACE  
  
  }  
  section {  
    margin: 0;  
  }  
}
```

5. Save the file and reload the page in Chrome.

Nice! While this looks cool with the overlapping shadowed sections, the design calls for 30px of space between the columns, so let's add that.

6. Back in **main.css**, add the correct amount of gap between the columns:

```
@media (min-width: 1100px) {  
  main {  
    max-width: 1400px;  
    display: flex;  
    gap: 30px;  
  }  
  section {  
    margin: 0;  
  }  
}
```


Creating Columns with Flexbox

7. Save the file and reload the page in Chrome.

- The space between the columns is now good.
- Resize the window so you can see how when it gets narrower, the left and right columns touch the edge of the window.
- We want to add the same 30px of space on the outside (on the left and right).

8. Back in **main.css**, add horizontal padding to create space on the left and right:

```
@media (min-width: 1100px) {  
  main {  
    max-width: 1400px;  
    display: flex;  
    gap: 30px;  
    padding: 0 30px;  
  }  
  section {  
    margin: 0;  
  }  
}
```

9. Save the file and reload the page in Chrome.

- This is looking pretty good.
- In the **Take a Hike** section, click **Learn More** to go to the hiking page.
- On the hiking page, notice the column widths are not equal. The browser tries to chose widths based on how much content each column has. We don't want that, so we'll have to set a width to make them all equal.

10. Back in **main.css**, add a width:

```
@media (min-width: 1100px) {  
  main {  
    CODE OMITTED TO SAVE SPACE  
  }  
  section {  
    margin: 0;  
    width: 33.33%;  
  }  
}
```


11. Save the file and reload the page in Chrome.

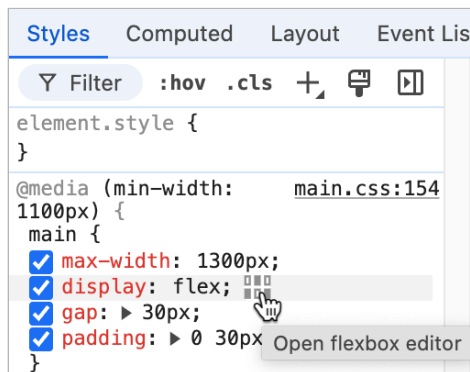
- Now all 3 columns are the same width.
- Click the logo at the top left of the page to return to the homepage (index.html)
- Let's see how Flexbox gives us control the vertical alignment of the columns.

Vertical Alignment with Flexbox

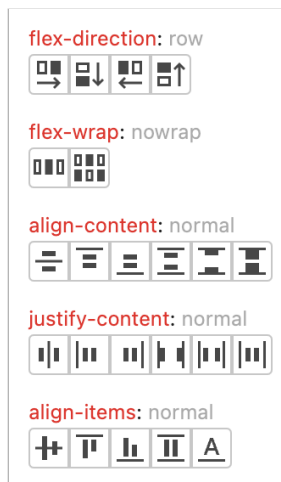
1. **Ctrl-click** (Mac) or **Right-click** (Windows) on a photo in one of the 3 sections and select **Inspect** to open Chrome's DevTools.
2. Make sure the window is wide enough so you see the 3 columns.
3. In the DevTools **Elements** panel you'll see the HTML code.

Select the **main** tag.

4. In the **Styles** panel, in the **main** rule click the Flexbox editor button  shown here:



5. You should now see this pop-up panel:



Creating Columns with Flexbox

6. Under **align-items** click each icon and watch how the vertical alignment of the 3 columns change:

- center
- flex-start (which is like top)
- flex-end (which is like bottom)
- stretch (this is the default alignment)
- baseline (which isn't appropriate for this layout)

We like the top alignment (flex-start), so let's add that to our code.

7. Back in **main.css**, add the following code shown below in bold:

```
@media (min-width: 1100px) {  
  main {  
    max-width: 1400px;  
    display: flex;  
    align-items: flex-start;  
    gap: 30px;  
    padding: 0 30px;  
  }  
}
```

8. Save the file and reload the page in Chrome.

The 3 columns should be aligned at the top, each with their own individual height.

9. In the **Take a Hike** section, click **Learn More** to go to the hiking page.

Notice the columns on this page take on the same alignment. Nice!

Exercise Preview

to steep switchbacks leading into the upper atmosphere. What path will you take?

[Download a Trail Guide](#)

[Learn More »](#)

Exercise Overview

Usually content is added using HTML, but in this exercise you'll learn how it can be helpful to use CSS to add content before or after an element.

Getting Started

1. If you completed the previous exercise, **index.html** should still be open from the **Tahoe New Design** folder.

If you did not do the previous exercise, open the **Tahoe New Design Ready for Pseudo-Elements** folder in your code editor and open **index.html**.

2. Preview **index.html** in Chrome.

For the **Learn More** links at the bottom of each section, we want to remove the SVG arrow and switch to a different arrow.

3. Switch back to your code editor.

Removing the SVG Arrows

1. Around line 36 we want to delete the entire SVG tag and all its contents. Instead of dragging a selection, let's see another way (which has a keystroke).

Place your cursor in the **svg** part of `<svg class="arrow">`

2. Hit **Cmd-Shift-Control-Right Arrow** (Mac) or **Shift-Alt-Right Arrow** (Windows) 3 times until the entire SVG tag up to the `` is selected.

NOTE: This is the keystroke for **Selection > Expand Selection**.

3. There are 2 more of these tags to select and delete:

- Hit **Cmd-D** (Mac) or **Ctrl-D** (Windows) 2 times so all 3 instances of the SVG code are selected.
- Hit **Delete** (Mac) or **Backspace** (Windows) to delete all the SVG tags at once!

4. All 3 **Learn More** links should now look like this:

```
<p><a href="#" class="button-secondary">Learn More</a></p>
```

- Except for the first which is linked to hikes.html
- Notice the **Learn More** links have a **button-secondary** class we can use to style them.

5. Save the file.

Pseudo-Elements & the Content Property

1. Open **main.css** (from the **css** folder).

2. Find these 2 rules (which we no longer need) and delete them:

- `button-secondary svg`
- `.button-secondary svg *`

3. Above the **footer** rule add the following new rule

```
.button-secondary::before {  
  
}
```

The `::before` creates a **pseudo-element** that will render something into the anchor tag before everything else that's already in it. It's typically used to add cosmetic content to an element, as we'd like to do here.

4. Specify the content to add by adding the code shown below in bold. Pay close attention to spaces, and to type » use **Opt-Shift-** (Mac) or hold down **Alt** while typing code **0187** on the numeric keypad (Windows).

```
.button-secondary::before {  
  content: "» ";  
}
```

5. Save the file and reload the page in Chrome.

- Notice the **Learn More** links now have the special » character added at the beginning of the **Learn More** links.

6. Return to your code editor.

Pseudo-Elements & the Content Property

7. Change **before** to **after**:

```
.button-secondary::after {
  content: "» ";
}
```

8. Move the space from before the » character to after it:

```
.button-secondary::after {
  content: " »";
}
```

9. Save the file and reload the page in Chrome.

Now the » character should be at the end of the **Learn More** links.

Seeing Pseudo-Elements in Chrome's DevTools

1. **Ctrl-click** (Mac) or **Right-click** (Windows) on a **Learn More** link's » character and choose **Inspect**.
2. In the DevTools **Elements** panel you'll see the HTML code.
 - Expand the `` element to see the `::after` inside.
 - Click on the `::after` and you'll see the style that creates it listed in the **Styles** panel.

Updating the Box-Sizing Rule

1. Switch back to **main.css**
2. In the `*` rule at the top, we set all elements (`*`) to use the better box-sizing.

Currently this rule would not affect content added with `::before` or `::after` though. So as a best practice, let's also apply it in those cases. In the selector, add `::before` or `::after` as shown below:

```
*, *::before, *::after {
  box-sizing: border-box;
}
```

3. Save the file. This doesn't change anything currently in our page, but is a best practice in case we need it later.

Single vs. Double Colons (Pseudo-Element vs. Pseudo-Class)

Some pseudo-elements (such as `:before` and `:after`) can be written with one or two colons (`:after` or `::after`). Why? Initially they were written with one colon, but the W3C wanted to distinguish pseudo-elements from pseudo-classes (such as `:hover`, `:first-of-type`, and `:last-child`) so they changed pseudo-elements to a double colon syntax.

Because the single colon syntax had already been used, it's acceptable for backward compatibility but is not allowed for new pseudo-elements.

What's the difference between a pseudo-class and pseudo-element?

Pseudo-Class:

- Styles an element when it's in a certain state.
- Preceded by one colon.
- Examples are `:hover`, `:first-of-type`, `:last-child`, `:not`, `:checked`

Pseudo-Element:

- Styles a specific part of an element.
- Preceded by two colons.
- Examples are `::after`, `::before`, `::first-letter`, `::first-line`

We've only covered some pseudo-classes and pseudo-elements. You can learn more at tinyurl.com/pseudo-ce

Exercise Preview

enjoy the great outdoors. visit art galleries or shop indoors. There's something for everyone!

[Book a Ski Trip](#) ↗

[Learn More »](#)

Exercise Overview

In this exercise, you'll learn how to use attribute selectors to add icons so users can know what kind link they will be clicking.

Getting Started

1. If you completed the previous exercise, reopen **index.html** from the **Tahoe New Design** folder.

If you did not do the previous exercise, open the **Tahoe New Design Ready for Attribute Selectors** folder in your code editor and open **index.html**.

2. Preview **index.html** in Chrome.

The links at the bottom of each column go to a different type of destination:

- Click **Download a Trail Guide** to open a PDF in a new tab. (This is a single page dummy PDF file.)
- Click **Watch a Video** to go to a youtube.com video in a new tab.
- Click **Book a Ski Trip** to go to a website in a new tab.

Let's add icons to tell users what kind of links these are.

3. Leave the page open in the browser.

Looking at the Links

1. Return to **index.html** in your code editor.

2. Near the end of each **section** tag, find the link above **Learn More** and note the **href** for each:

- href="downloads/trail-guide.pdf"
- href="https://www.youtube.com/watch?v=4ZLZh3ZWdGI"
- href="https://www.skilaketahoe.com"

Adding the Link Icons

We'll be adding the icon as a background image on the link. We've provided the icons in the **img** folder. To make things simpler, the icons are designed to be displayed at the exact same size: 16px by 16px.

Let's start out writing some very general rules and get more specific as we progress.

1. Open **main.css** from the **css** folder.
2. Let's write a general rule for anchor tags to give them a background image. Above the **footer** rule, add the following new rule:

```
a {  
    background: url(../img/icon-pdf.png);  
}
```

NOTE: We already have a rule for anchor tags, but we'll be getting more specific with this in a moment so we're making a new rule.

3. Save the file, then reload the page in your browser.
4. The background image is showing up behind the links, but it's repeating by default. Let's change that.
5. Return to **main.css** in your code editor.
6. Add the bold code shown:

```
a {  
    background: url(../img/icon-pdf.png) no-repeat;  
}
```

7. Save the file, then reload the page in your browser.
8. The icon is no longer repeating behind the links. It only appears once.
9. Let's move the icon to the right of each link. Go to **main.css** and add the bold code:

```
a {  
    background: url(../img/icon-pdf.png) no-repeat right center;  
}
```

10. Save the file, then reload the page in your browser.

Attribute Selectors

11. We want to display the icons at 16 x 16px. Go to **main.css** and add:

```
a {
  background: url(../img/icon-pdf.png) no-repeat right center / 16px 16px;
}
```

12. Save the file and reload the page in the browser.

So the size looks good, but the icon is hard to see behind the text. This is because we're putting it behind the anchor tag which is an inline element. Inline elements are only as wide as they need to be by default. In order to make space for the icon to the right of the links, we need to give the anchor tag some padding on the right.

13. Go to **main.css** and add:

```
a {
  background: url(../img/icon-pdf.png) no-repeat right center / 16px 16px;
  padding-right: 24px;
}
```

14. Save the file and reload the page in the browser. Looks great!

Targeting with Attribute Selectors

Now that we've gotten the icon styled correctly, we need to apply the appropriate icon to each type of link. What's the best way to do this? We could use a class, but that requires manually adding extra code to every link. A better way is to use attribute selectors.

Attributes are settings we add to an element's HTML opening tag (for example href, alt, and target). We want the PDF icon to only appear next to the link for the trail guide PDF, so we can target the link's **href** attribute.

1. Switch back to **main.css**.
2. Add **[href="downloads/trail-guide.pdf"]** to the **a** selector as shown below:

```
a[href="downloads/trail-guide.pdf"] {
  background: url(../img/icon-pdf.png) no-repeat right center / 16px 16px;
  padding-right: 24px;
}
```

3. Save the file and reload the page in the browser. Now the PDF icon only shows next to the **Download a Trail Guide** link.

This code isn't reusable because it only applies to this specific link. We'd like to use this icon anytime we link to a PDF, so there's a better way. We can target **href** attributes that end with **.pdf**

4. Go to **main.css** and edit the selector as follows:

```
a[href$=".pdf"] {  
    background: url(../img/icon-pdf.png) no-repeat right center / 16px 16px;  
    padding-right: 24px;  
}
```

When using the = sign, the target must match **exactly** what is between the quotes.

Using **\$=** targets anything that **ends with** what is within the quotes. So with the above code, we are targeting any link with an **href** attribute that ends with **.pdf**

5. Save the file and reload the page in the browser. Everything looks the same but now our code is more versatile because it works for all PDF links.

Adding the External Link Icon

Next we need to target external links, such as **Book a Ski Trip**. For this we can target links with an **href** that starts with **http**.

1. Return to your code editor.
2. Copy the rule we just wrote and paste a duplicate directly below:

```
a[href$=".pdf"] {  
    background: url(../img/icon-pdf.png) no-repeat right center / 16px 16px;  
    padding-right: 24px;  
}  
a[href$=".pdf"] {  
    background: url(../img/icon-pdf.png) no-repeat right center / 16px 16px;  
    padding-right: 24px;  
}
```

3. Edit the copy as shown in bold:

```
a[href$=".pdf"] {  
    background: url(../img/icon-pdf.png) no-repeat right center / 16px 16px;  
    padding-right: 24px;  
}  
a[href^="http"] {  
    background: url(../img/icon-external-link.svg) no-repeat right center /  
16px 16px;  
    padding-right: 24px;  
}
```

Using **^=** targets anything that **begins with** what is in the quotes. So the above code targets any link with an **href** that starts with **http** (and give them a background image of **icon-external-link.svg**).

Attribute Selectors

4. Save the file and reload the page in the browser.

Notice the **Book a Ski Trip** link has an external links icon next to it.

Adding the YouTube Icon

1. Notice that the external link icon appears next to the **Watch a Video** link as well because that is also a link that starts with **http**. It's true that this is an external link but we'd like it to have a YouTube icon instead.
2. Go to **main.css** in your code editor.
3. Again, copy the rule we previously wrote and paste a duplicate directly below:

```
a[href^="http"] {
    background: url(../img/icon-external-link.svg) no-repeat right center /
16px 16px;
    padding-right: 24px;
}
a[href^="http"] {
    background: url(../img/icon-external-link.svg) no-repeat right center /
16px 16px;
    padding-right: 24px;
}
```

4. We want to give all YouTube links a YouTube icon, so we can target anything that has a string that contains "youtube.com". Edit the code as shown in bold:

```
a[href^="http"] {
    background: url(../img/icon-external-link.svg) no-repeat right center /
16px 16px;
    padding-right: 24px;
}
a[href*="youtube.com"] {
    background: url(../img/icon-youtube.svg) no-repeat right center / 16px 16px;
    padding-right: 24px;
}
```

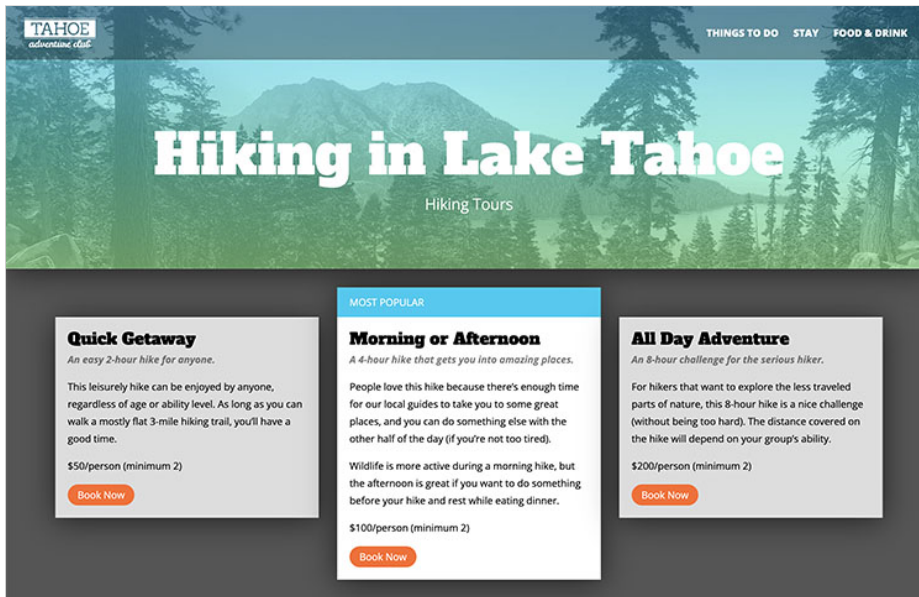
Using ***** targets anything that contains the content within the quotes. The ***** is a wildcard symbol that means the target content can appear anywhere, surrounded by any other characters.

5. Save the file and reload the page in the browser. Check out the YouTube icon next to the **Watch a Video** link!

NOTE: You may be wondering why the external link icon isn't appearing next to the YouTube link anymore, even though the link starts with http. Both the **http** and **youtube** selectors have the same amount of specificity, so the later rule (youtube) overrides the earlier rule (http).

Relational Selectors

Exercise Preview



Exercise Overview

In this exercise, you'll learn about CSS selectors that enable you to target elements based on the relationships between elements.

Getting Started

1. We'll be working with the **Tahoe Relational Selectors** folder in **Desktop > Class Files > Advanced HTML CSS Class**. Open that folder in your code editor if it allows you to (like Visual Studio Code does).
2. Open **index.html** from the **Tahoe Relational Selectors** folder.
3. Preview **index.html** in Chrome.
 - This is the same content we've been working with, but we made some minor changes to the links at the bottom of the 3 sections.
 - Now that we've put both links on the same line, it's hard to see that there are two different links. Let's make the first link (Learn More) stand out.
4. Leave the page open in Chrome.

Using first-child & last-child

1. Return to **index.html** in your code editor.

2. At the bottom of any of the 3 **section** tags, notice the last paragraph has a **links** class and contains the 2 links we just saw in the browser.
3. Switch to **main.css** (in the **css** folder).
4. Below the **section .text-wrapper** rule, add the following new rule:

```
section .links a:first-child {  
    color: #05b333;  
    margin-right: 15px;  
}
```

5. Save the file, reload the page in Chrome, and:
 - Notice the **Learn More** links are now green and have a bit more space between the link to their right.
 - Notice there's extra white space at the bottom of the 3 columns (more than on the sides). That's because the links are in a paragraph which has bottom margin. Let's remove that extra space.
6. Return to your code editor.
7. Below the **section .text-wrapper** rule, add the following new rule:

```
section .text-wrapper :last-child {  
    margin-bottom: 0;  
}
```

NOTE: A few steps ago we attached `:first-child` to the link `<a>` tag (`a:first-child`) to find links that were a first child.

The selector we're writing in this step is not attached to any tag. Using `:last-child` by itself means it will target anything that is a last child of `.text-wrapper`. In other words, it acts like a wildcard.

8. Save the file and reload the page in Chrome. That removed the extra space at the bottom of the 3 columns so the spacing around the text looks more even.
9. In Chrome, at the bottom of the first column find the **Download a Trail Guide** link.
10. **Ctrl-click** (Mac) or **Right-click** (Windows) on the **Download a Trail Guide** link and choose **Inspect**.

Relational Selectors

11. In the DevTools **Styles** panel you should see the **section .text-wrapper :last-child** rule is also applying to this element, even though we don't want it to!

That's because it too is a last child inside the **.links** paragraph (which is in **.text-wrapper**). We have to be very careful with selectors such as last-child!

There are multiple ways we could write a better selector that would not include this link. We could target **.links** but what if some columns wouldn't have links? We still want to use a relational selector to remove the extra space from below anything that might be at the bottom of this column. Continue on to learn the solution.

12. In the DevTools **Elements** panel (where you see the HTML), notice that the **links** paragraph is inside a **text-wrapper** div. We only want to target immediate children of the **text-wrapper** div.

Direct Child/Descendant Selectors

The direct descendant selector (child combinator) lets us target only immediate children, rather than all descendants (which includes grandchildren, great-grandchildren, etc.).

1. Return to your code editor.
2. In the rule you just wrote, add the **>** character as shown below:

```
section .text-wrapper > :last-child {
  margin-bottom: 0;
}
```

NOTE: Think of the **>** as a pointing arrow rather than a "greater than" symbol.

3. Save the file and reload the page in Chrome. In the **Styles** panel you should see that the **Download a Trail Guide** link no longer has the **:last-child** rule applied.

Using first-of-type

1. Still in Chrome, **Ctrl-click** (Mac) or **Right-click** (Windows) on any of the column headings (such as **Take a Hike**) and choose **Inspect**.
2. Press your **Delete** key to remove it.
3. Notice the first paragraph (which used to be just below the heading) lost its bold italic styling.
4. Reload the page.
5. Return to your code editor.

6. Find the **h2 + p** rule.

While this rule currently works, what if a column might not have a heading? There's another way we can target this element. It's the first paragraph in each section.

7. Change the **h2 + p** selector name to **section p:first-of-type** as shown below in bold:

```
section .text-wrapper p:first-of-type {  
    font-weight: 700;  
    font-style: italic;  
    opacity: .4;  
    margin-top: 3px;  
}
```

8. Save the file, and reload the page in Chrome. The first paragraphs should still have their bold italic formatting.
9. Still in Chrome, **Ctrl-click** (Mac) or **Right-click** (Windows) on any of the column headings (such as **Take a Hike**) and choose **Inspect**.
10. Press your **Delete** key to remove it.
11. Notice that this time, the first paragraph (which used to be just below the heading) has kept its bold italic styling.

As you can see, there are many ways to target something with CSS. While relational selectors can be useful, they require that those relationships will not change. If you can't be sure about the relationship, you should use a class instead.

12. Reload the page.

Using nth-child

1. Return to your code editor.
2. Let's switch to another page in this site. In your code editor open **hikes.html**.
3. Find the **main** tag and notice this page has a **price-list** class so we'll be able to target content in this page, without affecting the other page.
4. Preview **hikes.html** in Chrome.

If we wanted to style the second column, how could we target it? It's not the first or last-child or first or last-of-type. The nth-child selector allows us to specify the number of the child we want.

5. Return to **main.css** in your code editor.

Relational Selectors

6. Before we target the second column, let's test the `nth-child` selector. In the **min-width: 1100px** media query, below the **main** rule add the following new rule:

```
main.price-list :nth-child(2) {  
    background: red;  
    border: 5px solid black;  
}
```

Within the parentheses of `nth-child()` we put the number of the child we want to target. We declared a background color and border so we'll be able to easily see which elements have been selected.

7. Save the file and reload the page in Chrome.
- Wow, there are multiple red boxes! The second child of every element in the main part of the page got a red background with black border.
 - We have to be careful with the `nth-child` selector. As we saw with the `last-child` selector, if we're not specific enough, `nth-child` can apply to children, grandchildren, etc. It's often a best practice to combine it with a direct descendant selector (which uses the `>` notation) to avoid applying it to unwanted elements.
8. Return to **main.css** in your code editor.
9. Let's target only the second element which is a direct descendant (child) of the main price-list. Add `>` to the selector name as follows:

```
main.price-list > :nth-child(2) {  
    background: red;  
    border: 5px solid black;  
}
```

10. Save the file and reload the page in Chrome.
- Now only the second column is red with a black border around it.
11. Return to **main.css** in your code editor.
12. Let's check out another cool feature of `nth-child`. It can target multiple children, such as **odd** or **even** numbered children. Change the **2** to **odd** as shown below:
- ```
main.price-list > :nth-child(odd) {
 background: red;
 border: 5px solid black;
}
```
13. Save the file and reload the page in Chrome. Now the first and third columns are red with a black border.
14. Return to **main.css** in your code editor.

15. Now we can make the styling appropriate for this page. Delete the border, change the background color, and add some top margin so you end up with as follows:

```
main.price-list > :nth-child(odd) {
 background: #ddd;
 margin-top: 48px;
}
```

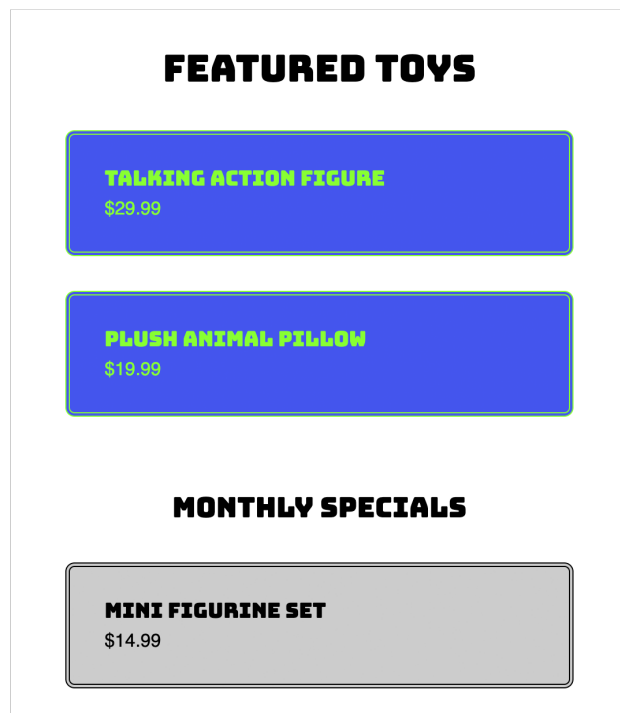
16. Save the file and reload the page in Chrome.

The first and third columns should have more space above them and have a light gray background which helps the middle column to visually stand out more.

---

# CSS Variables (Custom Properties)

## Exercise Preview



## Exercise Overview

In this exercise you'll learn about CSS variables (also called custom properties). Variables are found in many programming languages (such as JavaScript), and are also in CSS.

We often reuse something (a font, a size, or an amount of padding) many times throughout a CSS file. Changing those values everywhere can be tedious. Variables let us store a value in one place and refer to that by name. When we update the variable's original value, all places are updated. That's powerful!

---

## Getting Started

1. Open the file `variables.html` which is located in **Desktop > Class Files > Advanced HTML CSS Class**.
2. Preview `variables.html` in Chrome.

For this exercise we're going to work in a basic page so it will be easier to focus on the code required for this technique.

3. Leave the page open in Chrome.

---

## Defining a CSS Variable

1. Switch back to **variables.html** in your code editor.

We embedded the CSS in this file for simplicity, but everything would work the same if it's in a linked CSS file.

2. Let's create a variable for the main font that we want to use (a Google font named **Coiny** which we've already linked the page to). This will allow us to easily change the font later. At the start of the **style** tag, add the following code shown in bold:

```
<style>
 :root {
 --font-primary: Coiny, sans-serif;
 }
 body {
```

What is **:root**? Mozilla's MDN web docs say "The **:root** CSS pseudo-class matches the root element of a tree representing the document. In HTML, **:root** represents the **<html>** element and is identical to the selector **html**, except that its specificity is higher." [tinyurl.com/css-var-root](https://tinyurl.com/css-var-root)

We can declare a variable (custom property) on any element, but if no value is set on an element, the value of its parent/ancestor is used. Declaring a variable in **:root** makes it available to all elements (because **:root** is the ancestor of all elements).

When creating variables you choose the name, so **font-primary** is a name we created, not something special given to us by CSS.

---

## Using the Variable

1. From now on we can refer to the Coiny font using our variable. In the **h1**, **h2** rule, replace the font name with a reference to our variable, as shown below in bold:

```
h1, h2 {
 font-family: var(--font-primary);
 font-weight: normal;
```

2. Do the same thing again in the **.product-name** rule:

```
.product-name {
 font-family: var(--font-primary);
 font-size: 18px;
}
```

3. Save the file, then reload the page in Chrome. If your code is correct, nothing should change, but now it will be easier to update that font. Let's do that next!

# CSS Variables (Custom Properties)

## Updating the Variable

1. Return to your code editor.
2. Change **Coiny** to **Bungee** (another Google font we've loaded) as shown below:

```
:root {
 --font-primary: Bungee, sans-serif;
}
```

3. Save the file, then reload the page in Chrome. The font for the headings and product names should have both changed.

On such a basic page this might not seem as impressive, but on a large site (where the font is referenced many times) this makes updates much easier and faster.

## Doing More with Variables

Let's create another variable and we'll see more things we can do with them.

1. Return to your code editor.
2. Throughout our design we want to consistently use the same amount of space (or increments of that amount). Create a variable for that, as shown below in bold:

```
:root {
 --font-primary: Coiny, sans-serif;
 --spacing-primary: 15px;
}
```

3. In the **.card** rule replace the margin's pixel value with a reference to the variable we just created:

```
.card {
 CODE OMITTED TO SAVE SPACE
 margin: var(--spacing-primary);
 CODE OMITTED TO SAVE SPACE
}
```

4. In the previous exercise we learned how `calc()` can perform math in CSS. We can combine that with variables! For **h2** tags we want to use double the amount of standard spacing, so we can multiply our standard space variable by 2.

In the **h2** rule, replace the pixel value with **calc()** as shown below in bold:

```
h2 {
 margin-top: calc();
}
```

5. Inside `calc()` add the following code shown in bold (pay close attention to spaces):

```
h2 {
 margin-top: calc(var(--spacing-primary) * 2);
}
```

6. Save the file, then reload the page in Chrome and notice:

- Everything should still look the same if your code is correct.
- The padding inside each card (colored box) visually looks correct, but to achieve that we had to make the vertical padding (top and bottom) 5 pixels less than the horizontal padding (left and right), which compensated for some space added by the text's line height. In our current CSS, we did the math ourselves. Now that we're using variables, we'll want to use CSS `calc()` to do it.

7. Return to your code editor.

8. In the `.card` rule notice the padding amount is 10px (vertical) and 15px (horizontal).

9. In the `.card` rule, replace the padding's 15px with a reference to our variable:

```
.card {
 CODE OMITTED TO SAVE SPACE
 padding: 10px var(--spacing-primary);
 CODE OMITTED TO SAVE SPACE
}
```

10. Replace the padding's 10px with `calc()` as shown below:

```
.card {
 CODE OMITTED TO SAVE SPACE
 padding: calc() var(--spacing-primary);
 CODE OMITTED TO SAVE SPACE
}
```

11. Inside `calc()` add the code shown below in bold (pay close attention to spaces):

```
.card {
 CODE OMITTED TO SAVE SPACE
 padding: calc(var(--spacing-primary) - 5px) var(--spacing-primary);
 CODE OMITTED TO SAVE SPACE
}
```

12. Save the file, then reload the page in Chrome. The page should still look the same if your code is correct, but we're now ready to see how powerful this variable is.



## CSS Variables (Custom Properties)

13. **Ctrl-click** (Mac) or **Right-click** (Windows) anywhere on the page and select **Inspect** to open Chrome's DevTools.
14. In DevTools' **Element** panel, select `<html lang="en">`.
15. In **Styles** panel you should see the `:root` selector and our 2 variables.
16. Next to `--spacing-primary` click on the **15px** value to select it.
17. Press the **Up Arrow** key multiple times to increase the value and watch how the page updates!  
  
The 15px we were using is a bit small, somewhere around 30px seems better.
18. Return to your code editor.
19. As shown below in bold, change the `--spacing-primary` value to **30px**:

```
:root {
 --font-primary: Coiny, sans-serif;
 --spacing-primary: 30px;
}
```

---

### The Power of Inheritance

In HTML we nest elements inside of others. We often refer to the containing elements are parents/ancestors, and the nested elements as children/descendants. In CSS, descendants inherit settings from their ancestors. For example setting a font on body gets inherited to headings and paragraphs within the body.

CSS variables also work with inheritance. Let's see how we can redefine the value stored in a variable for one a specific part of a page.

1. In the `:root` rule, add a new variable as shown below in bold:

```
:root {
 --font-primary: Bungee, sans-serif;
 --spacing-primary: 30px;
 --card-background-color: #45e;
}
```

2. In the `.card` rule, update the background-color's value to use our new variable, by changing the code shown below in bold:

```
.card {
 background-color: var(--card-background-color);
 CODE OMITTED TO SAVE SPACE
}
```

3. Define one more variable, as shown below in bold:

```
:root {
 --font-primary: Bungee, sans-serif;
 --spacing-primary: 30px;
 --card-background-color: #45e;
 --card-text-color: white;
}
```

4. In the `.card` rule, replace the color's value with a reference to our new variable:

```
.card {
 background-color: var(--card-background-color);
 color: var(--card-text-color);

 CODE OMITTED TO SAVE SPACE
}
```

5. Let's also use the color in the border, so replace the border's **white** with a reference to our variable:

```
.card {
 background-color: var(--card-background-color);
 color: var(--card-text-color);
 border: 4px double var(--card-text-color);

 CODE OMITTED TO SAVE SPACE
}
```

6. Save the file, then reload the page in Chrome. If your code is correct, the color of the card should still look the same (white text on a blue background).
7. Return to your code editor.
8. Change the value of `--card-text-color` as shown below in bold:

```
:root {
 --font-primary: Bungee, sans-serif;
 --spacing-primary: 30px;
 --card-background-color: #45e;
 --card-text-color: #8f3;
}
```

9. Save the file, then reload the page in Chrome.
- Notice the text and border color changed from white to green.
  - Now that we have these variables, let's make the two products under **Monthly Specials** look different.
10. Return to your code editor.

## CSS Variables (Custom Properties)

11. In the HTML, notice:

- There are 4 links which share the same **card** class.
- The **card** links are grouped into two divs: **featured** and **specials**.
- The links we want to change the color of are in the **specials** div.

12. In the CSS, below the **:root** rule, add the following new rule:

```
.specials {
 --card-background-color: #ccc;
 --card-text-color: black;
}
```

NOTE: If no value is set for a custom property (variable) on an element, the value of its parent/ancestor is used. In the examples so far, the parent/ancestor was the **:root** element. By redeclaring variables on **.special**, the cards within **.special** will use these new values (which override the **:root** values).

13. Save the file, then reload the page in Chrome.

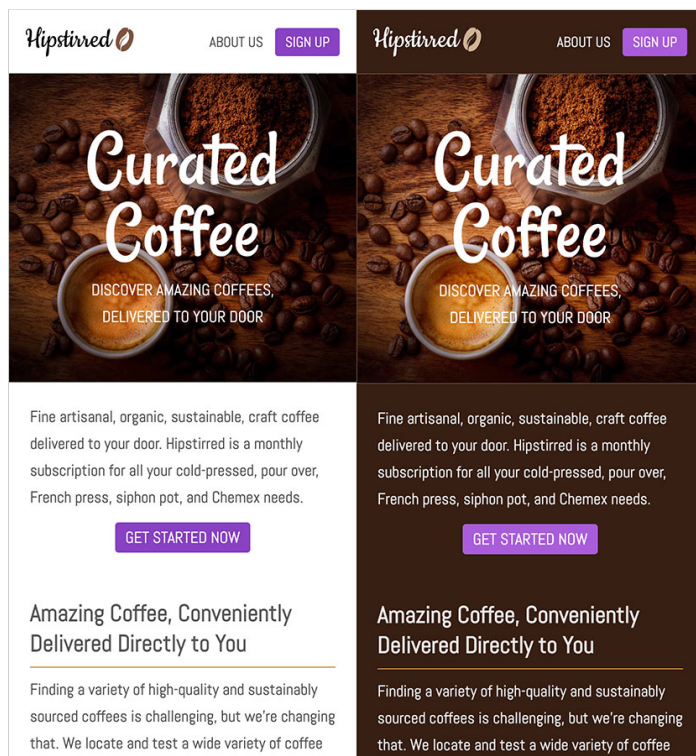
The bottom 2 product cards should now have a different text and background color than the top 2 cards (black text on a gray background).

---



# Light & Dark Modes using CSS

## Exercise Preview



## Exercise Overview

In this exercise you'll learn how to use CSS variables to make a webpage change its appearance based on a user's preference for light or dark mode.

## Getting Started

1. In your code editor, close any files you may have open.
2. We'll be working with the **Light Dark Mode** folder located in **Desktop > Class Files > Advanced HTML CSS Class**. Open that folder in your code editor if it allows you to (like Visual Studio Code does).
3. Open **index.html** from the **Light Dark Mode** folder.
4. Preview **index.html** in Chrome.

Operating systems such as iOS, Android, macOS, and Windows have light and dark modes that users can choose between.

This page has been designed for light mode. We'll keep this design, but we'd also like users who prefer dark mode to be able to see an alternate design that matches their preference (of a darker background with light text).

5. Leave the page open in Chrome.

---

## Detecting Dark Mode & Changing Colors

1. Back in your code editor, open **main.css** from the **css** folder.
2. In the **:root** style near the top, notice we've already created CSS variables (which we've used appropriately in other styles).

Using variables is crucial, because we can redefine what they represent and all the places that refer to them will all update!


3. Below the **:root** style, add the following new rule:

```
@media (prefers-color-scheme: dark) {
 :root {
 --text-color-primary: #ddd;
 --background-color-primary: #26150d;
 --background-color-subtle: #4c3125;
 }
}
```

4. Save the file and reload the page in Chrome.

If your computer is set to dark mode, you'll automatically see the change!

If your computer is set to light mode you could change the mode, but instead you can use Chrome's DevTools to simulate dark mode as follows:

- **Ctrl-click** (Mac) or **Right-click** (Windows) anywhere on the page and select **Inspect** to open Chrome's DevTools.
- At the top right of the **Styles** panel, click the paint brush  and from the menu choose **prefers-color-scheme: dark**

5. Notice the following:

- Not everything needs to change its appearance. For example the text over the photo and the purple button.
- Just changing the colors gets the page in pretty good shape, but some graphics need to also change:
  - the black logo at the top left can barely be seen
  - the social media icons at the bottom right are too hard to see
- There can be other changes we'd like to make. For example, we think the photo now blends in too much with the background, so we'd like to add a subtle border to it.

## Light & Dark Modes using CSS

- Back in **main.css** below the **.hero** rule, add the following new rule:

```
@media (prefers-color-scheme: dark) {
 .hero {
 border: 1px solid var(--background-color-subtle);
 }
}
```

NOTE: While we could have put this rule in the dark mode media query we already made, sometimes it's nice to keep all the styles related to an element (in this case the hero) together so they can be found easily.

- Save the file and reload the page in Chrome.

Notice the subtle border around the photo (which is only visible in dark mode).

---

### Changing Images with the Picture Element

Instead of using an **img** tag, we can use a **picture** tag which allows us to dynamically change between different images!

- In your code editor, switch to **index.html**
- In the **header** wrap a picture tag around the current image:

```

 <picture>

 </picture>

```

TIP: If you're using Visual Studio Code (and set up the wrap keystroke using the instructions in the **Before You Begin** section near the start of this book), you can quickly wrap a tag as follows:

- Select the entire **img** tag.
- Hit **Option-W** (Mac) or **Alt-W** (Windows).
- Type **picture** and hit **Return** (Mac) or **Enter** (Windows).

3. Inside the picture tag, add a new **source** for the image that will be used for dark mode:

```
<picture>
 <source srcset="img/logo-light.svg" media="(prefers-color-scheme: dark)">

</picture>
```

Notice how we're using the same media query from CSS, but here in HTML! If the **source** tag's media query does not apply, the **img** tag will be used as a fallback. Therefore the **img** tag will be for light mode.

4. Save the file and reload the page in Chrome.

The logo text should now be white so you can see it on this dark background.

---

## Styling SVG for Dark Mode

Unlike the logo which is multiple colors, the social media icons in the footer are a single color. If we convert the linked **img** tags to embedded (inline) SVG we could use **currentColor** to style them as we did in a previous exercise.

1. In your code editor, go into the **img** folder and in **social-icons** open **facebook.svg**
  - Select all the code and copy it.
  - Close the file.
2. Switch to **index.html** in your code editor.
3. In the **footer** find the **facebook** link and delete the **img** tag and paste the SVG code so it looks like:

```

 <svg width="24" height="24" viewBox="0 0 24 24" fill="none" xmlns="http://
www.w3.org/2000/svg">
```

CODE OMITTED TO SAVE SPACE

```
</svg>

```

4. Repeat this process for the other 2 icons (X and Instagram).
5. Save **index.html** and reload the page in Chrome.

You should still see the icons. If you noticed they moved up a little, don't worry we'll fix that soon. Let's change their color first.

6. Switch to **main.css** in your code editor.



## Light & Dark Modes using CSS

7. Below the **.social a** rule, add this new rule:

```
.social svg * {
 fill: currentColor
}
```

8. Save the file and reload the page in Chrome.

Notice they are the color of links. Let's style these links to be black in light mode, and white in dark mode.


9. Back in **main.css** edit the existing **.social a** rule by adding a color for light mode:

```
.social a {
 color: #000;
 text-decoration: none;
 margin-left: 10px;
 opacity: .6;
}
```

10. Below the **.social a** rule, add this new in a media query:

```
@media (prefers-color-scheme: dark) {
 .social a {
 color: #fff;
 }
}
```

11. Save the file and reload the page in Chrome.

- The icons are now a good color and the page should look great in dark mode!
- Make sure light mode still looks good by either switching your computer to light mode or in Chrome's DevTools at the top right of the **Styles** panel, click the paint brush  and choose **prefers-color-scheme: light**

12. When we switched the social icons from **img** tags, to **svg** tags they lost the vertical alignment on **img** tags so they moved up slightly (so they're no longer vertically aligned). Let's fix that.

Back in **main.css** edit the **img** selector by adding **svg**:

```
img, svg {
 max-width: 100%;
 vertical-align: bottom;
}
```

13. Save the file and reload the page in Chrome.

The social icons should now be properly vertically centered.

### Optional Bonus: "Alt" Text for Embedded SVGs

Inline SVGs don't have an **alt** attribute like **img** tags do. To tell screen reader say to someone who is vision impaired or blind, we can add a **title** tag.

1. Switch back to **index.html** in your code editor.
2. In the **footer** find the **facebook** link and add a **title** inside the **svg** tag as shown:

```

 <svg width="24" height="24" viewBox="0 0 24 24" fill="none" xmlns="http://
www.w3.org/2000/svg">
 <title>Facebook</title>
 <path d="M22.6566 0H1.32392C0.593267
```

3. If you want, you could repeat this process with the other 2 social media icons.
4. Save the file and reload the page in Chrome.

The icons should still look the same but if you hover over the Facebook icon and pause, a small tooltip will appear that says **Facebook**.

#### Another Way to Code Light & Dark Modes

---

There is a newer approach that uses less code, but as of this writing we don't think the browser support is good enough to use it yet.

Learn more about the technique at [web.dev/articles/light-dark](https://web.dev/articles/light-dark)

See current browser support: [caniuse.com/mdn-css\\_types\\_color\\_light-dark](https://caniuse.com/mdn-css_types_color_light-dark)

---

## Exercise Preview

SIGN UP FOR MORE INFO

First and Last Name
Email
I am available

Choose one ▾

I want to volunteer for

- ☐ Affordable Housing
- ☐ Animals

Anything else you'd like to add?

Sign Me Up

## Exercise Overview

In this exercise, you'll learn how to create a form with various types of user inputs.

## Getting Started

1. In your code editor, close all open files to avoid confusion.
2. For this exercise we'll be working with the **Helping Hands** folder located in **Desktop > Class Files > Advanced HTML CSS Class**. Open that folder in your code editor if it allows you to (like Visual Studio Code does).
3. Open **volunteer.html** from the **Helping Hands** folder.
4. Preview **volunteer.html** in a browser.

We'll create a form in the left column under the **Sign Up for More Info** heading.

5. Leave the page open in the browser.

## Creating a Form & Text Inputs with Labels

1. Back in **volunteer.html** find the **Sign Up for More Info** heading on line 28.

2. In the empty line below that, type **form** and hit **Tab** to use Emmet's shortcut to expand to:

```
<form action=""></form>
```

- For user inputs (form elements) to work, they must all be inside of a **form** tag.
  - The form **action** will need to point to something on the back-end to process the form when it gets submitted. That requires a back-end language such as back-end JavaScript, PHP, Ruby on Rails, etc. Those must be run through a web server and not merely viewed in a web browser. So while we can't make that part of a form work, we can code the HTML for what the user sees (and later style it with CSS).
3. In the **form** create a new line. On that new line type **input** and hit **Tab** to create:

```
<form action="">
 <input type="text">
</form>
```

4. Save and reload the page in your browser.
  - Click in the empty field and notice you can type text.
  - The user will need something to tell them what to enter here.
5. Back in your code editor, on a new line above the **input**, type **label** and hit **Tab** to create:

```
<form action="">
 <label for=""></label>
 <input type="text">
</form>
```

6. In the label:

```
<label for="">First and Last Name</label>
```

7. Save and reload the page in your browser.

Click on **First and Last Name** and notice that nothing happens. You'll see why we're trying this is just a moment.

8. Back in your code editor, on the **input** add the following code shown in bold:

```
<input type="text" id="full-name">
```

9. Point the **label** to the name you just added to the input:

```
<label for="full-name">First and Last Name</label>
```

# Creating Forms with HTML

10. Save and reload the page in your browser.
  - Click on **First and Last Name** again and notice this time it puts the cursor into the text field!
  - This proves they are connected, which is important for accessibility.
11. When the form is submitted, it needs a name to put for the data the user entered. While this doesn't change anything visibly on the front-end, it's important for the back-end to function. Back in your code editor, add a **name** as shown below:

```
<input type="text" name="full-name" id="full-name">
```

---

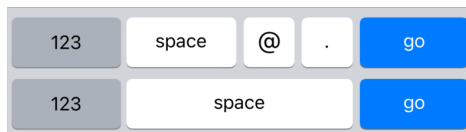
## Email Input

1. Let's add an **email** field. On a new line type **input:email** and hit **Tab** to create:

```
<form action="">
 <label for="full-name">First and Last Name</label>
 <input type="text" name="full-name" id="full-name">

 <input type="email" name="" id="">
</form>
```

NOTE: How does the **email** type differ from **text**? One way is that mobile users get different keyboards. In the screenshot below, notice the top keyboard has an @ and period to make typing emails faster, versus the regular text keyboard below that.



2. Add in the **name** and **id**:
 

```
<input type="email" name="email" id="email">
```
3. On a new line above, type **label** and hit **Tab** to create:
 

```
<form action="">
 <label for="full-name">First and Last Name</label>
 <input type="text" name="full-name" id="full-name">

 <label for=""></label>
 <input type="email" name="email" id="email">
</form>
```
4. Point the label to the input's email id, and add text for the user to see:
 

```
<label for="email">Email</label>
```

5. Save and reload the page in your browser.

This form doesn't look good because we haven't styled it, but we'll do that in a later exercise. For now let's focus on functionality.

---

## Submit Button

All forms need a way to submit the information.

1. Back in your code editor, on a new line at the end of the form type **input:submit** and hit **Tab** to create:

```
<input type="email" name="email" id="email">
```

```
<input type="submit" value="">
</form>
```

2. Add some text for the value:

```
<input type="submit" value="Sign Me Up">
```

3. Save and reload the page in your browser.

- In the **email** field type **me**
- Click the **Sign Me Up** button and you should get a message about the email not being valid. Each browser varies on what it says and how that looks.
- In the **email** field type **me@my.com**
- Click **Sign Me Up** to see the page reloads and the email becomes blank.

This proves the form was submitted. We left the **form** tag's **action** blank, so nothing happens with the data. It will be up to a back-end programmer to take that data, do something with it (such as store it in a database) and decide what happens next. For example they could create a thank you page, or an error page if something is wrong.

---

## Textarea

Text inputs are a single line. For multiple lines we use a **textarea**.

1. Back in your code editor, on a new line above the submit type **textarea** and hit **Tab** to create:

```
<textarea name="" id=""></textarea>
```

```
<input type="submit" value="">
</form>
```

## Creating Forms with HTML

2. Give it a **name** and **id**:

```
<textarea name="comments" id="comments"></textarea>
```

3. On a new line above that, type **label** and hit **Tab** to create:

```
<label for=""></label>
<textarea name="comments" id="comments"></textarea>
```

4. Point the label to the id you just gave the textarea, and add some text for users to see:

```
<label for="comments">Anything else you'd like to add?</label>
```

5. Save and reload the page in your browser.

- You should see the a multi-line text area (currently only a couple lines).
- We'll control the size with CSS, which we'll do in a later exercise.

---

### Fieldset, Legend, & Radio Buttons

We group related form elements (such a multiple radio buttons) with a fieldset. Instead of a label (which we used for inputs), fieldset uses a legend.

1. Between email and comments, on a new line type **fieldset** and hit **Tab** to create:

```
<input type="email" name="email" id="email">

<fieldset></fieldset>

<label for="comments">Anything else you'd like to add?</label>
```

2. Create a new line inside the fieldset and type **legend** and hit **Tab** to create:

```
<fieldset>
 <legend></legend>
</fieldset>
```

3. In the legend add some text:

```
<legend>I want to volunteer for</legend>
```

4. Save and reload the page in your browser.

- The fieldset looks like a bordered box with the title **I want to volunteer for**
- We can remove that border via CSS (which we'll do in a later exercise).
- Let's add some content into this area.

5. Back in your code editor, in a new line below **legend** type **ul>li\*2** and hit **Tab** to create:

```
<fieldset>
 <legend>I want to volunteer for</legend>

</fieldset>
```

6. In the first **li**, create a new line, type **input:radio** and hit **Tab** to create:

```


 <input type="radio" name="" id="">


```

7. Add an **id**:

```

 <input type="radio" name="" id="housing">

```

8. Add a **label** below it:

```

 <input type="radio" name="" id="housing">
 <label for="housing">Affordable Housing</label>

```

9. Copy and paste the input and label into the second **li**:

```


 <input type="radio" name="" id="housing">
 <label for="housing">Affordable Housing</label>

 <input type="radio" name="" id="housing">
 <label for="housing">Affordable Housing</label>


```



# Creating Forms with HTML

10. Change the **id**, **for**, and **text** as highlighted below in bold:

```

 <input type="radio" name="" id="housing">
 <label for="housing">Affordable Housing</label>

 <input type="radio" name="" id="animals">
 <label for="animals">Animals</label>

```

11. Save the file and preview in a browser.

- Ignore the bullet points from the default list styling. We'll remove those later.
- Check the radio buttons and notice how you can choose both options.
- Radio buttons are supposed only allow users to choose one option. That's how they are different from checkboxes which allow users to select multiple options. Let's see how to make these radio buttons work correctly.

12. Back in your code editor, give both inputs the same name:

```

 <input type="radio" name="interests" id="housing">
 <label for="housing">Affordable Housing</label>

 <input type="radio" name="interests" id="animals">
 <label for="animals">Animals</label>

```

13. Save and reload the page in your browser.

- Try clicking both options and notice how you can only select one. That's how this is supposed to work!
- There's only one issue remaining. How will the back-end know what value to associate with the one option the user selected? We need to add a value attribute that's unique to each radio button.

14. Back in your code editor, add the **values** shown below:

```

 <input type="radio" name="interests" id="housing" value="housing">
 <label for="housing">Affordable Housing</label>

 <input type="radio" name="interests" id="animals" value="animals">
 <label for="animals">Animals</label>

```

NOTE: You don't see these in a browser, but the back-end developer will need them.

---

## Checkbox vs Radio Button

What if we want people to be able to choose multiple options? We'd use checkboxes instead of radio buttons.

1. Change **radio** to **checkbox** as shown below:

```

 <input type="checkbox" name="interests" id="housing" value="housing">
 <label for="housing">Affordable Housing</label>

 <input type="checkbox" name="interests" id="animals" value="animals">
 <label for="animals">Animals</label>

```

2. Save and reload the page in your browser.

Check on both options and notice you're not limited to just one. That's correct for how checkboxes are supposed to work.

---

## Form Menus with the Select Tag

A more compact way of allowing users to select only one option from a list of many choices is to use a menu.

1. Back in your code editor, between email and fieldset, on a new line type **select** and hit **Tab** to create:

```
<input type="email" name="email" id="email">

<select name="" id=""></select>

<fieldset>
```

## Creating Forms with HTML

2. Give it a **name** and **id**:

```
<select name="time" id="time"></select>
```

3. Above it add a **label** with an appropriate name and text:

```
<label for="time">I am available</label>
<select name="time" id="time"></select>
```

4. Now we must provide a list of options to display in the menu. Inside the **select** create a new line, type **option\*2** and hit **Tab** to create:

```
<select name="time" id="time">
 <option value=""></option>
 <option value=""></option>
</select>
```

5. Give it appropriate values and text to show the user:

```
<option value="days">Days</option>
<option value="nights">Nights</option>
```

6. Save and reload the page in your browser.

Click on the **Days** menu and choose **Nights** to see the menu works.

---

### Optional Bonus: Adding a Non-Selectable Menu Option

1. We can add a title to the menu that users can't select as an option. Back in your code editor add this:

```
<option selected disabled>Choose one</option>
<option value="days">Days</option>
<option value="nights">Nights</option>
```

2. Save and reload the page in your browser.

Click on the menu and notice the new option can be seen, but not chosen. Nice!

---



## Styling Forms (with Attribute Selectors)

### Exercise Preview

**SIGN UP FOR MORE INFO**

First and Last Name  
John White

Email  
jw@styling.com

I am available  
Days

### Exercise Overview

In this exercise, you'll learn to style the form we created in the previous exercise.

### Getting Started

1. If you completed the previous exercise, **volunteer.html** should still be open from the **Helping Hands** folder.

If you did not do the previous exercise, open the **Helping Hands Ready for Form Styling** folder in your code editor and open **volunteer.html**.

2. Preview **volunteer.html** in Chrome. The form works, but needs layout and better styling to match our page.
3. Leave the page open in the browser.
4. In your code editor, open **main.css** from the **css** folder.
5. Below the **.form-heading** rule, add this new rule:

```
form {
 border: solid 1px #ddd;
 border-radius: 0 0 8px 8px;
 padding: 25px;
}
```

6. Save and reload the page in Chrome.

The form should have rounded border with padding inside.

---

## Styling the Labels & Inputs

Let's put the text labels and text fields onto their own lines.

1. Back in your code editor, below the **form** rule, add a rule for **labels**:

```
form > label {
 display: block;
 margin: 20px 0 5px;
}
```

By using a direct descendant selector, we avoid styling the checkbox labels in the nested fieldset. We want to keep those labels next to their checkboxes.

The shorthand above sets a 20px top margin, 0 left/right, and 5px on bottom.

2. Save the file and reload the page in Chrome.

Notice the text labels are now on their own line, with the text fields below them. Now let's improve the text fields.

3. Back in your code editor, below the **form > label** rule, add a new rule for text inputs:

```
input[type="text"] {
 display: block;
 width: 100%;
}
```

NOTE: By default, inputs are set to display as inline. We're changing them to display as block to ensure they are on their own line and that we'll be able to set size, padding, etc. like we do on block elements.

4. Save the file, reload the page in Chrome, and notice:

- The **First and Last Name** text field is now the full width of the column.
- The **Email** text field is not the full width, because its type is email, not text like our CSS rule targeted.
- The menu is unstyled and we'll want it to make the other form elements.
- The textarea below **Anything else you'd like to add?** is not on its own line because we didn't target textareas.

## Styling Forms (with Attribute Selectors)

5. Back in your code editor, edit the input selector as shown in bold, making sure you don't miss the commas!

```
input[type="text"],
input[type="email"],
textarea,
select {
 display: block;
 width: 100%;
}
```

6. Save and reload the page in Chrome.

The email field, menu, and textarea should all be 100% wide and on their own lines.

7. Back in your code editor, add more styling to make them look better:

```
input[type="text"],
input[type="email"],
textarea,
select {
 display: block;
 width: 100%;
 font-size: 16px;
 background: #eee;
 border: none;
 border-radius: 8px;
 padding: 10px;
 margin-bottom: 20px;
}
```

8. Save and reload the page in Chrome:

- Notice the text fields should have a gray background, no border, and rounded corners which better matches the rest of the page's design.
- The menu is too wide, so let's make it the width of its contents.

9. Back in your code editor, below the rule you just edited, add this new rule:

```
select {
 width: auto;
}
```

10. Save the file, reload the page in Chrome, and:

- Notice the menu's width has shrunk to hug its contents. Much better.
- We can't move the default menu arrow. That would require hiding the default arrow and adding a custom arrow. We'll keep with the default arrow for this page.
- There's too much space above **First and Last Name**, so let's fix that.

11. Back in your code editor, below the **form > label** rule, add this new rule:

```
form > label:first-of-type {
 margin-top: 0;
}
```

12. Save the file, reload the page in Chrome, and:

- Notice the space above **First and Last Name** should look better.
- Try to resize the textarea below **Anything else you'd like to add?** and notice that when you adjust the width, it can get wider than the column or become too small. It would be better if you could only adjust the height, not the width.

13. Back in your code editor, below the **select** rule, add the following new rule:

```
textarea {
 min-height: 120px;
 resize: vertical;
}
```

14. Save the file, reload the page in Chrome, and:

- Notice the textarea is initially a bit taller.
- Resize the textarea and notice that you can only change the height, not the width.
- Click into a text field and it may get a blue outline. How this looks differs across browser. If you don't like the way this looks, we can get rid of it.

15. Back in your code editor, below the **textarea** rule, add the following new rule:

```
input[type="text"],
input[type="email"],
input[type="submit"],
textarea,
select {
 outline: none;
}
```

16. Save the file and reload the page in Chrome.

Click into any of the fields and notice no more blue outline.

---

## Changing Background Color on Focus

With no blue outline, it's hard to know which field your cursor is in, so we should make some other visual indication of which field the cursor is in.



## Styling Forms (with Attribute Selectors)

1. Back in your code editor, above the **aside section** rule, add the following new rule:

```
input:focus,
textarea:focus,
select:focus {
 background: #e1efff;
 color: #02387a;
}
```

NOTE: This rule targets the inputs and textarea when they are focused on (selected by a mouse or keyboard).

2. Save the file, reload the page in Chrome, and:
  - Click into any of the text fields and type something to see their color change! When you click out of a field, it goes back to the original color. Neat effect!
  - The fieldset has a border and some spacing by default, but let's remove that.

---

### Styling the Fieldset

1. Back in your code editor, above the **aside section** rule, add the following new rule:

```
fieldset {
 border: 0;
 margin: 0;
 padding: 0;
}
```

2. Save the file and reload the page in the browser.
  - The border around the list of checkboxes is gone.
  - Let's remove the bullets next to the checkboxes.
3. Our form contains a list (**ul** tag) with all the checkboxes. Back in your code editor, below the **fieldset** rule, add the following new rule:

```
fieldset ul {
 list-style-type: none;
 margin: 0;
 padding: 0;
}
```

4. Save and reload the page in the browser.

The form is looking good, except for the **Sign Me Up** button.

---

### Styling the “Sign Me Up” Button

1. The **Sign Me Up** button is an input with **type="submit"** which we can use to target it. Back in your code editor, below the **fieldset ul** rule, add the following new rule:

```
input[type="submit"] {
 font-size: 20px;
 padding: 10px 20px;
 font-family: var(--font-secondary);
 font-weight: 600;
 text-transform: uppercase;
 background: #6bb359;
 color: #fff;
 border: none;
 border-radius: 8px;
}
```

NOTE: font-secondary is a variable we created (above in the :root rule).

2. Save the file and reload the page in the browser. Much improved! Let's make the button change color on hover.
3. Back in your code editor, below the **input[type="submit"]** rule, add the following new rule:

```
input[type="submit"]:hover, input[type="submit"]:focus {
 background: #449d44;
}
```

4. Save the file, reload the page in Chrome, and:  
  
Hover over the **Sign Me Up** button to see it darken.

#### Minimum Type Size for Mobile Optimized Forms

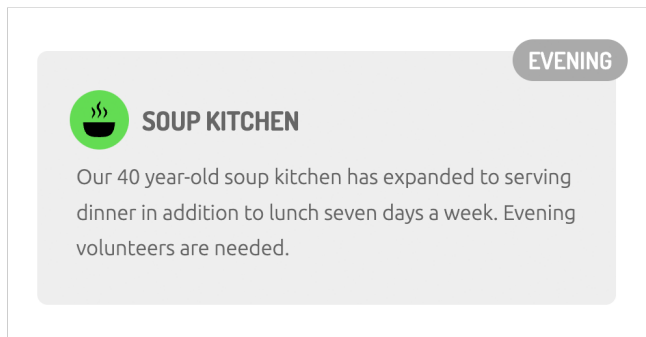
If the font size of a text field is less than 16px, iOS will zoom in on the text field when you click on it. That can be annoying because you have to zoom back out when done typing. Style text fields as 16px or bigger to avoid this.

#### Safari & Menu Styling

Form elements can be challenging to style as browsers vary in their default appearance as well as how CSS styling works. By default Safari does not accept the menu styling we did in this exercise. You could apply **-webkit-appearance: none;** to the **select** tag to get it to work, but that would also remove the arrow (which indicates it's a menu) so you'd have to add a custom arrow graphic.

## Relative Sizes: Em and Rem

### Exercise Preview



### Exercise Overview


In this exercise, you will learn the difference between a fixed size (such as pixels) and relative sizes (such as ems and rems).

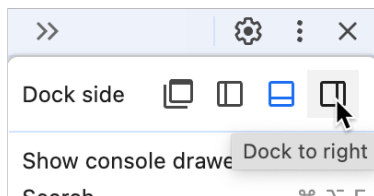
---

### Em Units

1. If you completed the previous exercise, **volunteer.html** should still be open from the **Helping Hands** folder.

If you did not do the previous exercise, open the **Helping Hands Ready for Em vs Rem** folder in your code editor and open **volunteer.html**.

2. Preview **volunteer.html** in Chrome (we'll be using its DevTools).
3. **Ctrl-click** (Mac) or **Right-click** (Windows) on the **Sign Me Up** button and choose **Inspect**.
4. If the DevTools are not docked to the right side of the browser window, at the top right of the DevTools panel click the  button and choose **Dock to right** as shown below:



5. In the **Styles** panel, the top style should be `input[type="submit"]`

- In that style, click on the **20px** value for **font-size**.
- Use your **Up** and **Down Arrow** keys to make the text bigger and smaller.

As you change the font-size, notice the green padding area around the type remains roughly the same size. It does grow a little bit, because the line height changes with the font-size, which affects the spacing around the type.

If we truly wanted to proportionally scale the button up or down, we'd need to adjust the padding amounts as well.

6. Leave the page open in Chrome.

### About Em Units

One problem with using pixel sizes is when you decide to change the size of something (such as a sidebar or navbar) which contains many individually sized elements, you have to manually change each element's pixel size. This can be tedious, but luckily we can use relative sizes, such as ems.

In a webpage 1em is equal to the current font-size. So if an element is in the body tag, then 1em is equal to the root font-size (browsers default to 16px unless you've changed it with CSS). If you have an article for which you've set a 20px font-size, inside the article 1em equals 20px.

While ems are based on font-size, you can use ems for other values such as padding, margin, etc. as you'll see next.

7. Return to your code editor.
8. Open **main.css** from the **css** folder.
9. Find the `input[type="submit"]` rule and change the padding to ems as shown below:

```
input[type="submit"] {
 font-size: 20px;
 padding: .5em 1em;
}
```

CODE OMITTED TO SAVE SPACE

This element's font-size is 20px, therefore 1em currently equals 20px. The padding was 10px 20px, so:

- 10px is half of 20px, so that equals 0.5em.
- 20px is the same as the font-size, so that equals 1em.

## Relative Sizes: Em and Rem

10. Save and reload the page in Chrome.

The **Sign Me Up** button's padding should still look good, but let's see how the padding will now automatically adjust when we adjust the font-size.

The **Sign Me Up** button should still be selected with the `input[type="submit"]` at the top of the DevTools **Styles** panel.

If it's not, **Ctrl-click** (Mac) or **Right-click** (Windows) on the **Sign Me Up** button and choose **Inspect**.

11. Click on the **20px** value for **font-size**.

- Use your **Up** and **Down Arrow** keys to make the text bigger and smaller.
- This time, notice the green padding area adjusts proportionally as you change the font-size!

12. In the DevTools, click on the **Computed** tab, click on it.

- Here you can see all the properties listed in alphabetical order. Instead of scrolling through them to find padding, click in the **Filter** field and type **padding**
- This tells you how the padding's em units have been translated into pixels.
- Hit the **Esc** key to remove the filter.

13. Switch back to the **Styles** tab.

---

### Using Em Units for Font-Size

Let's switch to using ems for the font-size of the headings and paragraphs in aside (right column on desktop).

1. Back in your code editor, find the **h3** rule and change font-size to **1.25em**:

```
h3 {
 color: currentColor;
 font-size: 1.25em;
 margin: 0 0 7px;
}
```

2. Scroll down, and change the font-size of the **aside section p** rule:

```
aside section p {
 font-size: 1em;
 margin: 0;
}
```

3. Let's set a font-size on the **section** tags which contain the h3 and p tags we just updated. Find the **aside section** rule and add a font-size as shown below:

```
aside section {
 font-size: 14px;
```

CODE OMITTED TO SAVE SPACE

```
}
```

Within the aside's section tags, 1em will now equal 14px.

4. Find the **.opportunity-icon** rule and change the height:

```
.opportunity-icon {
 height: 2.25em;
```

CODE OMITTED TO SAVE SPACE

```
}
```

NOTE: This way the icon's height will be based on the font-size of the text it's in.

5. Save and reload the page in Chrome.
6. **Ctrl-click** (Mac) or **Right-click** (Windows) on the **Volunteer Opportunities** heading (at the top of the right column, or below the form if you're only seeing one column) and choose **Inspect**.
7. In the DevTools **Elements** panel, just below the selected **h2** click on any of the 3 **section** tags to select it.
8. In the **Styles** panel, the top style should be **aside section**
- Click on the **14px** value for **font-size**.
  - Hit your **Up Arrow** key a few times to make the size bigger and notice:
    - The headings and paragraphs in all 3 gray sections get bigger because they are using an **em** font-size.
    - The time badges (Evening, Day, and Flexible) do not change size because they are using a **px** font-size.
    - The colored icons get bigger because their height is using an **em** value.

One benefit of relative font sizes (like em) is making one size change can affect multiple elements.

## Relative Sizes: Em and Rem

### Rem Units

This page is a fairly basic example, but imagine using ems throughout multiple nested elements with different font-sizes (each changing what 1em is).

Unfortunately, this can get complex very quickly. The challenge of ems is the value of 1em is always changing, so matching a size across elements can be a huge pain. Ems are not the only relative unit. We can also use **rem**, which stands for **root em**.

- **Em** is relative to the current font-size, which changes throughout a page.
- **Rem** is relative to a page's root font-size, which is consistent throughout a page.

In this page we have not yet set default a font-size, so **1rem** will refer to the default **16px** set by the browser (or whatever the user's preference is if they changed the default size).

1. Back in your code editor, in **aside section p** add an "r" to change "em" to "rem":

```
aside section p {
 font-size: 1rem;
 margin: 0;
}
```

2. In the **h3** rule, add an "r" to change "em" to "rem":

```
h3 {
 color: currentColor;
 font-size: 1.25rem;
 margin: 0 0 7px;
}
```

3. Let's define a root font-size. We can set this on the **html** or **:root** element. At top of the file, add this font-size to the **:root** rule:

```
:root {
 font-size: 1rem;
 --font-primary: Ubuntu, sans-serif;
 --font-secondary: Dosis, sans-serif;
}
```

This will be used on all screen sizes, but next we'll set a larger root size for desktops.

You might wonder why we're not using pixels here. Keep in mind that users can change their browser's default text size. If we used pixels we're be overriding their preference, which is disrespectful. By using a relative size (such as rem, em, or %) we are creating a size relative to the user's preference (which by default is 16px unless they have changed it). In this case, 1rem would normally be 16px.

4. Towards the bottom of the file, in the media query for **min-width: 930px** add this new rule:

```
@media (min-width: 930px) {
 :root {
 font-size: 1.25rem;
 }
}
```

NOTE: Assuming the browser's default type size of 16px, this would set our base font size to 20px (1.25 x 16 = 20px).

5. Save and reload the page in Chrome.


Resize the browser window to see the size of the heading and paragraph text in the aside is bigger on desktop and smaller on mobile.

---

## Rems & A User's Browser Preference

While sizing text with pixels is easy (especially when coding a design that someone created in an app such as Figma), one of the most important reasons to use rem (and never to use pixels for font-size) is accessibility.

Example: a vision impaired user needs larger text, so they change their browser's preference. Pixel-based font sizes ignore that setting, while rem and em font sizes will honor and be based on that preference. Let's see this in action!

1. At the top right of the Chrome window, click on the  icon and from the menu choose **Settings**.
2. Drag the **Settings** tab out of the current window and arrange the windows so you can see it and our webpage at the same time.
3. In the **Settings** window, in the search field type **size**
  - From the **Font size** menu choose **Small**
  - Notice in our webpage how text using rem sizes changes, but pixel-sized text (like the **Become a Volunteer** heading) does not!
  - Set the **Font size** back to the default **Medium (Recommended)**

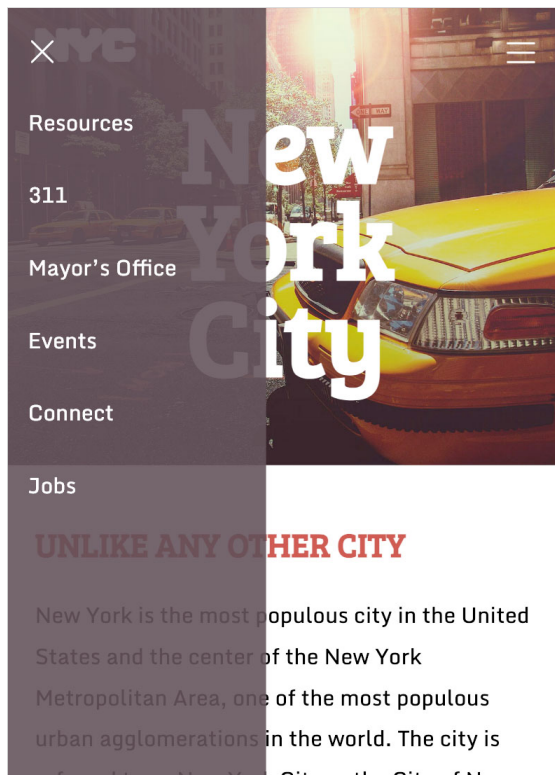
TIP: If you're coding a design that was created in Figma, Figma's Dev Mode has an option to convert pixels into rem!

---



# Off-Screen Side Nav Using Only CSS

## Exercise Preview



## Exercise Overview

In this exercise you'll learn how to create an off-screen navigation menu that is hidden from view until the user clicks a button to show it. You'll create this functionality purely with CSS. No JavaScript is required!

## Styling the Nav

1. We'll be switching to a new folder of prepared files for this exercise. In your code editor, close all open files to avoid confusion.
2. For this exercise we'll be working with the **Off-Screen Nav** folder located in **Desktop > Class Files > Advanced HTML CSS Class**. Open that folder in your code editor if it allows you to (like Visual Studio Code does).
3. Open **index.html** from the **Off-Screen Nav** folder.
4. Preview **index.html** in a browser.

The navigation is already coded with some styling done. Leave the page open in your browser.

5. We want to position the mobile menu first. In your code editor, open **main.css** from the **css** folder.

6. At the bottom below all the other rules, add the following:

```
@media (max-width: 700px) {
 nav {
 position: fixed;
 }
}
```

7. Save and reload your browser.

Resize the window to a mobile phone size, and you should see that the nav is now sitting on top of the page, as we wanted. Let's make it a bit wider and the full height of the screen.

8. Back in your code editor, add this width and height to the **nav** rule you just made:

```
nav {
 position: fixed;
 width: 12em;
 height: 100%;
}
```

9. Save and reload your browser.

- The nav looks pretty close to what we want on mobile.
- Resize the window very short, so some of the nav items get cut off.
- Try to scroll to see the missing nav items, and you'll see that you can't! This could easily happen on phones held horizontally, so let's fix that.

10. Back in your code editor, add the following property to the **nav** rule as shown below in bold:

```
nav {
 position: fixed;
 width: 12em;
 height: 100%;
 overflow-y: auto;
}
```

11. Save and reload your browser.

- Now you should be able to scroll through the nav items that are cut off.
- Resize the browser to the large desktop view (the nav will jump back to the top of the page). Let's style the nav for this size.

# Off-Screen Side Nav Using Only CSS

- Back in your code editor, at the bottom of the styles (after the current media query) add this new media query:

```
@media (min-width: 701px) {
 nav ul {
 display: inline-flex;
 }
}
```

NOTE: Why inline-flex and not flex? Inline elements follow their parent element's text alignment, which is how we'll make the nav move to the right.

- Save and reload the page in the browser.

The nav is now a single line across the top of the page. That's getting close, but we want the links aligned on the right.

- Back in your code editor, in the min-width: **701px** media query, add the following code shown below in bold:

```
@media (min-width: 701px) {
 nav {
 text-align: right;
 padding-right: .6em;
 }
 nav ul {
 display: inline-flex;
 }
}
```

- Save and reload the page in the browser.

Resize the browser window to see that the nav should look good in both screen sizes.

---

## Adding a Clickable Element to Open & Close the Nav

How does a user open or close the menu? We could use a link, but that would require JavaScript to trigger the change. One way we can accomplish this (with just CSS) is by using a checkbox. A checkbox has a **:checked** pseudo-class selector that allows us to style an element based on whether the element is checked or not (which will translate to our menu being open or closed)!

- Let's add the checkbox. In your code editor open **index.html**.
- Directly below the opening **body** tag, add the following input tag as shown below:

```
<body>
 <input type="checkbox" id="nav-checkbox">

```

3. Save the file.
4. Go to **main.css**.
5. At the bottom of the styles not in a media query, below the **.pull-quote** rule, add this new rule:

```
#nav-checkbox {
 position: absolute;
 left: 50%;
}
```

6. Save and reload the page in the browser.
  - You should see a small checkbox in the middle of the page at the top. We'll hide it later, but for now we want the checkbox easy to find and see.
  - Next we need to make the checkbox hide the menu.
7. Back in your code editor, in the **max-width: 700px** media query, below the **nav** rule, add this new rule:

```
#nav-checkbox:checked ~ nav {
 display: none;
}
```

NOTE: Siblings are two elements that share a common parent, such as our checkbox and nav. The sibling selector we are using targets the nav, but only if it comes after a checked checkbox with an ID of nav-checkbox. The order of our HTML is really important for this to work! In the HTML, the checkbox must be before the nav (and in the same parent element as the nav).

8. Save and reload the page in the browser.
  - Make sure the window is smaller than 700px.
  - Check the checkbox, noticing how the nav disappears when the box is checked.

---

## Moving the Nav Off-Screen

Rather than having the checkbox hide the menu, we want the checked state to show the menu. Additionally, we don't want to rely on **display: none** to hide the nav. We want it to slide onto the screen. Let's move the nav off-screen, so checking the box can bring it back onto the screen.

1. In your code editor, return to **main.css**.

## Off-Screen Side Nav Using Only CSS

2. In the **max-width: 700px** media query, add the following properties to the **nav** rule, as shown below in bold:

```
nav {
 position: fixed;
 width: 12em;
 left: -12em;
 height: 100%;
 overflow-y: auto;
 z-index: 100;
}
```

NOTE: This pulls the nav off the screen with a negative **left** position. The **left** position value must be equal to the **width** of the element to ensure that the whole element is hidden off-screen. We also set a **z-index** to ensure that the nav is always on top.

3. Find the **#nav-checkbox:checked ~ nav** rule and edit the code as shown in bold, so it will make the nav appear:

```
#nav-checkbox:checked ~ nav {
 left: 0;
}
```

4. Save and reload the page in the browser.

Toggle the checkbox to see that now the nav appears or disappears when the box is checked or unchecked!

The functionality works, but it could use some finesse. We can apply a CSS transition to slide the nav in from the side, rather than have it abruptly appear on the page.

5. Back in your code editor, at the bottom of the **nav** rule, add the following transition:

```
nav {
 position: fixed;
 width: 12em;
 left: -12em;
 height: 100%;
 overflow-y: auto;
 z-index: 100;
 transition: all .2s ease;
}
```

6. Save and reload the page in the browser.

Toggle the checkbox to see that the nav now slides in nicely from the side. Cool!

## Adding a Menu Button

We need something better for our users to click than a checkbox. We can use a form **label** with an image in it. In a standard form, labels are used to describe a form element, such as an email field. When you click a label, it focuses on the form element it targets, or in the case of a checkbox, it checks it!

1. Switch to **index.html** in your code editor.
2. Just under the opening **body** tag, add the code as shown below in bold:

```
<input type="checkbox" id="nav-checkbox">
<label for="nav-checkbox" class="menu-button">

</label>
```

3. Notice that the **for** attribute in the label is the same as the checkbox's **id**. This builds the connection and allows us to use the label to control the checkbox.
4. Save the file.
5. Switch to **main.css**.
6. In the **max-width: 700px** media query, after the **#nav-checkbox:checked ~ nav** rule, add the following new rule:

```
.menu-button {
 position: absolute;
 top: 10px;
 right: 15px;
 color: #fff;
 padding: 7px;
}
```

7. Save and reload the page in the browser.
8. In the top right corner, click the menu icon (3 lines):
  - Notice that the checkbox changes to the checked state, even though we're just clicking on the label. Cool!
  - Notice that when you hover over the label the cursor does not change to the hand icon that's common for links. We should change the cursor to better indicate to users that they can click on the menu icon.
9. Return to **main.css** in your code editor.

# Off-Screen Side Nav Using Only CSS

10. At the bottom of the **.menu-button** rule, add the new code as shown below in bold:

```
.menu-button {
```

CODE OMITTED TO SAVE SPACE

```
 cursor: pointer;
}
```

11. Save and reload the page in the browser.

- Hover over menu icon to see the proper hand cursor.
- Resize the window larger than 700px. We need to hide the menu icon here on desktop view.

12. Back in your code editor, at the start of the min-width: **701px** media query, add the following new rule:

```
.menu-button {
 display: none;
}
```

13. Save and reload the page in the browser.

- The menu icon should no longer appear in the desktop layout. The checkbox is still showing, but we'll hide that later.
- Resize the browser window to the mobile size.

---

## Adding a Close Button

Once the menu is open, we want users to be able to clearly know how to close the menu again. Let's add a close button inside the menu.

1. Switch to **index.html** in your code editor.
2. At the top of the **nav** add the code shown below in bold:

```
<nav>
 <label for="nav-checkbox" class="close-button">

</label>

```

Note again that we are giving this label the same **for** value as the menu icon. We can have multiple labels for the same checkbox, allowing us to create two different buttons that both control a single checkbox!

3. Switch to **main.css** so that we can style the close button.

4. In the **max-width: 700px** media query, after the **.menu-button** rule, add the following new rule:

```
.close-button img {
 padding: 17px;
 margin-top: 7px;
 cursor: pointer;
}
```

5. We also need to hide the close button on desktop (because the nav is always showing there). In the **min-width: 701px** media query, add a second selector to the **.menu-button** rule as follows. (Don't miss the comma!)

```
.menu-button, .close-button img {
 display: none;
}
```

6. Save and reload the page in the browser.
  - Click the menu icon to open the nav, then click the close button to close it.
  - Wow, it already works! That's because it's another label that triggers the same checkbox. The functionality of the checkbox was already working, this is just a second way to trigger it.

---

## Hiding the Checkbox with CSS

Our off-screen navigation works beautifully, with each label doing the correct function. All that's left is to hide the checkbox, because we don't need to see it any longer.

1. Back in your code editor, in the general styles, find the **#nav-checkbox** rule.
  - Delete the **left: 50%** line of code.
2. Also make the following edits shows in bold:

```
#nav-checkbox {
 position: fixed;
 clip-path: circle(0);
}
```

- While **display: none** may have seemed more intuitive it hides content from screen readers. The **clip-path** property is more accessible because it keeps content "visible" to screen readers.
- Changing the position value to **fixed** prevents some browsers, like Chrome, from jumping back up to the top of the page when the checkbox is toggled.



# Off-Screen Side Nav Using Only CSS

3. Save and reload the page in the browser.

Test out the completed menu!

---

## Optional Bonus: Creating an Overlay

While it's good that users have a close button, it would also be nice if they could click anywhere outside the menu to close it. Just as we did with the menu and close buttons, we can use another label to create a clickable element that controls the checkbox.

1. Switch to **index.html** in your code editor.
2. Just under the opening **body** tag, add the code shown below in bold:

```
<label for="nav-checkbox" class="menu-button">

</label>
<label for="nav-checkbox" class="overlay"></label>
```

3. Let's make the overlay visible with some styling. Switch to **main.css**.
4. In the **max-width: 700px** media query, after the **.close-button img** rule, add the following new rule:

```
.overlay {
 position: fixed;
 height: 100%;
 width: 100%;
 background: rgba(0,0,0, .5);
}
```

5. Save and reload the page in the browser.
  - Notice the slightly transparent black background over most of the page.
  - Click anywhere on the dark overlay and notice that the menu opens! That's because the overlay is a label **for** the **nav-checkbox** element, so it functions the same as the other buttons.
  - We only want the overlay to be available when the menu is open, and invisible at all other times.
6. Back in your code editor, find the **overlay** rule and add **display none**:

```
.overlay {
 display: none;
 CODE OMITTED TO SAVE SPACE
}
```

7. Below the **overlay** rule, add the following new rule:

```
#nav-checkbox:checked ~ .overlay {
 display: block;
}
```

8. Save and reload the page in the browser.

- Notice that the overlay is hidden.
- Click the menu icon to see that the nav appears, as well as the semi-transparent black background behind it.
- Click anywhere on the semi-transparent dark background to close the menu.
- You could keep the dark overlay if you like it, but we want to hide the overlay (while keeping the functionality).

9. Back in your code editor, in the **.overlay** rule, delete the **background** property.

10. Save and reload the page in the browser.

- Open the menu, then click anywhere outside the menu and it will still close.

---

### Optional Bonus: Flipping the Nav to the Right Side

With a small tweak, we can make the nav come in from the right.

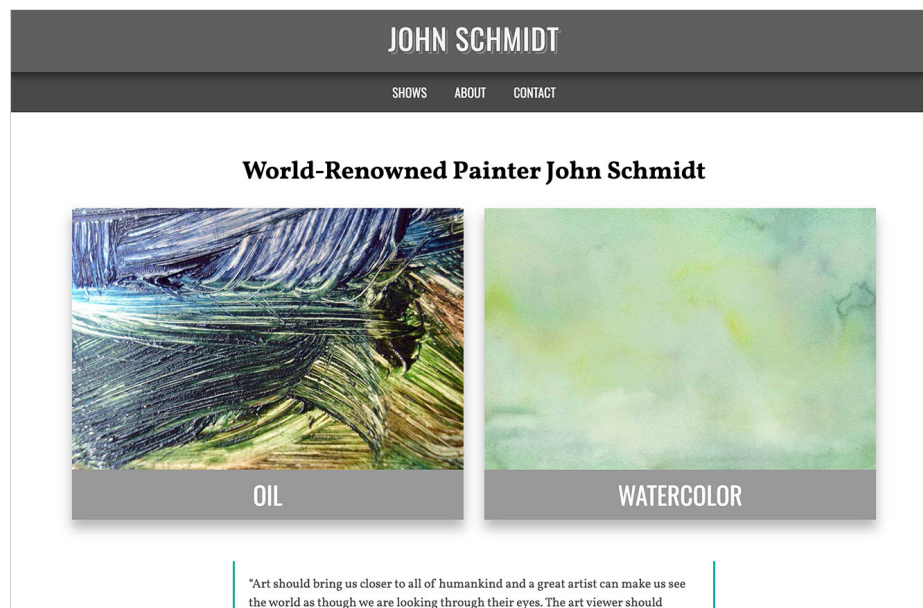
1. Back in your code editor, in the **max-width: 700px** media query, change **left** to **right** in both of these two rules (changes are shown in bold). Don't miss both places!

```
nav {
 position: fixed;
 width: 12em;
 right: -12em;
 CODE OMITTED TO SAVE SPACE
}
#nav-checkbox:checked ~ nav {
 right: 0;
}
```

2. Save and reload the page in the browser.

- Test out the nav and see that it comes in from the right side now.

## Exercise Preview



## Exercise Overview

In this exercise, you'll use box-shadow and text-shadow to add visual depth. You'll also learn how to adjust an element's front to back stacking order using relative position with z-index.

## Getting Started

1. We'll be switching to a new folder of prepared files for this exercise. In your code editor, close all open files to avoid confusion.
2. For this exercise we'll be working with the **Artist Website** folder located in **Desktop > Class Files > Advanced HTML CSS Class**. Open that folder in your code editor if it allows you to (like Visual Studio Code does).
3. Open **index.html** from the **Artist Website** folder.
4. Preview **index.html** in Chrome (we'll be using its DevTools).
  - Behold the mostly finished portfolio homepage of artist John Schmidt. Let's add some drop shadows.
  - The first thing we want to do is add a box-shadow to the 4 large featured art images to make them stand out more. Hover over them to see the pointer (hand) cursor. This tells us that each image is wrapped in a link. (Each link is also wrapped in a **div** tag with a class of **category**.)

5. Leave the page open in Chrome.

---

## Using the CSS Box-Shadow Property

The **box-shadow** property adds a drop shadow behind an element.

1. Back in your code editor, open **main.css** from the **css** folder.
2. In the **.category a** rule, add the bold code below:

```
.category a {
 box-shadow: 2px 6px 4px #000;
 margin: 30px 0;
 display: block;
 text-decoration: none;
}
```

The 4 values are horizontal (x) offset, vertical (y) offset, blur amount, & color.


3. Save and reload the page in Chrome.

Wow, that shadow is very harsh. Let's open the DevTools so we can tweak it.

4. **Ctrl-click** (Mac) or **Right-click** (Windows) on any of the large featured art images and select **Inspect**.
5. An **img** tag should be selected. In the **Elements** panel of DevTools, click the line above the selected image **<a href="#">**
6. In the **Styles** panel:

- In the **.category a** rule, click on the box-shadow's first px value (horizontal offset) and press your **Down Arrow** key until it becomes **-5px**
- While we're at it, change the second value (vertical offset) to **-6px**.

The shadow is now going in the opposite direction. Positive values place the shadow at the bottom right. Negative values place the shadow at the top left.

- Click the  icon by the box-shadow values and experiment with the blur and other settings.

## Box-Shadow, Text-Shadow, & Z-Index

7. Back in your code editor, modify the box-shadow values as shown below in bold:

```
.category a {
 box-shadow: 0 12px 18px rgba(0,0,0, .3);
 margin: 30px 0;
 display: block;
 text-decoration: none;
}
```

Remember that the alpha transparency takes a value between 0 (fully opaque) and 1 (fully transparent). This code gives us a 30% transparent black.

8. Save and reload the page in Chrome.

- Nice. We like this soft, subtle shadow.
- At the top of the page, notice the gray band with **John Schmidt**.

We want to add a shadow to this section so it looks like it's casting a shadow on the darker gray navigation below it. This top section (above the nav) is a div with a class of **logo-wrapper**.

9. Back in your code editor, add a shadow to the existing **.logo-wrapper** rule:

```
.logo-wrapper {
 background: #444;
 box-shadow: 0 2px 15px rgba(0,0,0, .8);
 CODE OMITTED TO SAVE SPACE
}
```

TIP: If your code editor has Emmet installed (like Visual Studio Code does), you can type **bxsh** and hit **Tab**. You will see: **box-shadow: inset hoff voff blur #000**; Make sure to delete **inset**, because we don't want an inset shadow here. Then tab into each of the placeholders and add the desired values.

10. Save and reload the page in Chrome.

Oh no! We expected to see a shadow cast onto the navbar, but we can't see it because it's hidden behind the navbar.

---

### Changing an Element's Default Stack Order

HTML elements have a default stacking order that determines which objects are in front of others. This is an object's position along the z-axis, which represents back to front depth.

1. To see the structure of the markup, switch to **index.html** in your code editor.

2. Locate the **header** tag to see that the `<div class="logo-wrapper">` and `<nav>` are siblings within their parent **header** tag.

We can change the kind of positioning to change the stacking order.

3. Switch to **main.css**.
4. Give the **.logo-wrapper** a relative position as shown below in bold:

```
.logo-wrapper {
 background: #444;
 box-shadow: 0 2px 15px rgba(0,0,0, .8);
 position: relative;
}
```

CODE OMITTED TO SAVE SPACE

NOTE: We don't want to use fixed positioning because we want this content to scroll with the document. Absolute positioning won't work either because we want to keep it in the normal document flow.

5. Save and reload the page in Chrome. Yes, our shadow is showing!

But why does this work? **Relative** position acts differently than the default **static** position. One of the ways it differs is how things stack!

NOTE: If we have multiple elements that we needed to manage (and assuming they all have a position other than static), we could use **z-index** to further control their front to back order. You'll see an example of z-index next.

---

## Inset Shadows & Z-Index

1. Still in Chrome, **Ctrl-click** (Mac) or **Right-click** (Windows) on any of the large featured art images and select **Inspect**.
  - In the **Elements** panel of the DevTools an **img** tag should be selected.
  - On the line above the selected image click on `<a href="#">`
2. In the **Styles** panel:
  - In the **.category** a rule, click on the **box-shadow's** value.
  - Change the **rgba()** to **#000** so you get **0 12px 18px #000** to make the shadow more pronounced.
  - Add **inset** at the beginning so you end up with **inset 0 12px 18px #000**
  - The shadow mostly disappears because the image in the link is covering over it, but you can see it a little bit of shadow in the gray band of text below the photo. Let's send the image backward in the stacking order.

## Box-Shadow, Text-Shadow, & Z-Index

3. In the **Elements** panel of the DevTools, click on the **img** tag inside the selected link `<a href="#">`
4. In the **Styles** panel there should be an **img, svg** rule.
  - Click in the empty space to the right of the **vertical-align: bottom;** property to create a new blank line so we can type in a new property.
  - Type **z-index: -1;** which should complete that property and put you on another line, ready to type another property.
5. We still don't see the shadow because z-index can't work with the default static position. Add **position: relative;** and the inset shadow should appear!
  - Positive numbers pull an element front. The higher the number, the farther front it moves. So **z-index: 20;** would be in front of **z-index: 10;**
  - Negative numbers push an element back.
6. This was just a test so we're not going to change the code in our code editor.

---

### Adding Drop Shadows to Text with CSS Text-Shadow

Let's give the **John Schmidt** text at the top of the page a shadow.

1. Switch back to **main.css** in your code editor.
2. The text we want to style is the anchor tag inside the **.logo-wrapper**. Below the **.logo-wrapper** rule, add this new rule:

```
.logo-wrapper a {
 text-shadow: 4px 4px 0 #666;
}
```

3. Save and reload the page in Chrome.

The **John Schmidt** text should now have a sharp gray shadow.

---

### Layering Text-Shadows for a Detached Outline Effect

The sharp shadow looks pretty cool, but we want to make it look like a stylized outline that is detached from the text by a few pixels. To get this result, we'll layer 2 text-shadows on top of each other. We want to create a shadow like the one we just created in the DevTools, but partially covered by a shadow that is indistinguishable from the logo-wrapper's background.

1. Switch back to **main.css** in your code editor.

2. In order for our overlay to look convincingly invisible, we need to set its color to the logo-wrapper's background color.

In the rule for **.logo-wrapper**, notice the background property is **#444** so that's what we'll use.

3. Shadow effects also have a stacking order. The first shadow is layered atop the second one, which overlays the third, and so on. Because we want a masking effect, we need to add the shadow that's the same color as the heading's background **before** the one we already have.

Add the following bold code to the text-shadow and do not miss the comma!

```
.logo-wrapper a {
 text-shadow: 2px 2px 0 #444, 4px 4px 0 #666;
}
```

The first shadow is smaller than the second one. This will create the illusion of 2 pixels of space between the text and the shadow that looks like a floating outline.

4. Save and reload the page in Chrome.
  - That's one stylin' shadow!
  - Hover over **John Schmidt** text. It would be interesting if we made the shadow move on hover.
5. Back in your code editor, copy the entire **.logo-wrapper a** rule.
6. Paste the code directly below and make changes highlighted below in bold (adding **:hover** and changing the shadow values):

```
.logo-wrapper a {
 text-shadow: 2px 2px 0 #444, 4px 4px 0 #666;
}
.logo-wrapper a:hover {
 text-shadow: -2px 2px 0 #444, -4px 4px 0 #666;
}
```

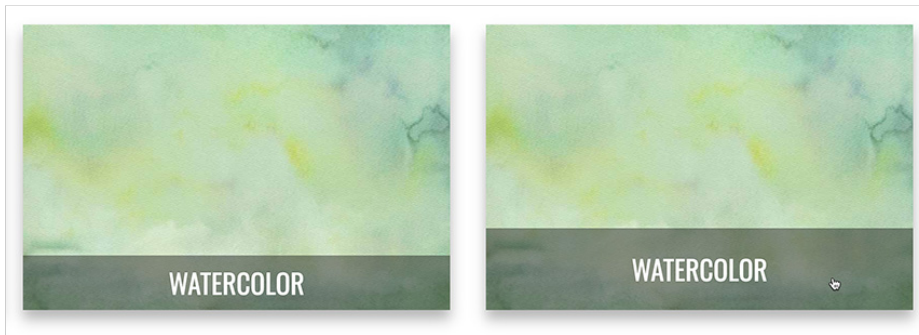
7. Save and reload the page in Chrome.

Hover over the text to see the shadow shift. Artsy!

---



## Exercise Preview



## Exercise Overview

In this exercise, you'll learn to make hovers animate smoothly by adding transitions. Transitions animate properties (such as color and size) much like "tweening" does, making the browser animate the in between states over a given time.

---

## Getting Started

1. If you completed the previous exercise, **index.html** should still be open from the **Artist Website** folder.

If you did not do the previous exercise, open the **Artist Website Ready for Transitions** folder in your code editor and open **index.html**.

2. Preview **index.html** in Chrome.

For the 4 photos, we want to move the text labels up and onto the bottom of the photo. We'll then animate those text labels on hover, so we're going to use absolute positioning to position them.

3. Leave the page open in the browser.

---

## Positioning the Text Over the Photo

1. Back in your code editor, open **main.css** from the **css** folder.

2. The anchor tag inside category div contains both the image and the overlay, so this is the closest parent element. In the **.category a** rule, declare a relative position as shown in bold:

```
.category a {
```

CODE OMITTED TO SAVE SPACE

```
 position: relative;
}
```

NOTE: Remember that anchors are natively inline elements, which make poor relative parents due to the limited space these elements take up. To make this work, note that we already set the display property to block (inline-block would also work).

3. In the **.category .description** rule, set a position as follows:

```
.category .description {
```

CODE OMITTED TO SAVE SPACE

```
 position: absolute;
}
```

4. Save and reload the page in Chrome.

The description has collapsed and it needs to know where to go. Let's give it positioning coordinates.

5. Back in your code editor, to make the overlay stretch to fill the entire anchor tag add the following bold coordinates to **.category .description**

```
.category .description {
```

CODE OMITTED TO SAVE SPACE

```
 position: absolute;
 bottom: 0;
 left: 0;
 right: 0;
}
```

6. Save and reload the page in Chrome.

- That looks nice! The overlay is at the bottom of its container, stretching to fill the width of its container.
- Let's make the description taller when we hover over the link (which contains both the image and description).

## CSS Transitions

7. Back in your code editor, below `.category .description` add this new rule:

```
.category a:hover .description {
 line-height: 6rem;
}
```

NOTE: It may be easier to read a CSS selector like this from right to left. We are targeting the `.description` div, but only when it is inside an `a` (anchor) tag we're **hovering** over (but only for anchors in the `.category` div).

8. Save and reload the page in Chrome:

- Hover over any of the images to see their description gets taller.
- It would be smoother if we animated the change in height. We can do that by adding a transition.

---

### Adding a Transition: Animating the Description Overlays

A CSS transition needs to know 2 things: what properties to transition and how much time the change should take. It may seem counterintuitive at first, but transitions are declared in the rule for the **object you wish to change** rather than on the "trigger" rule (such as `:hover`).

1. Back in your code editor, in the `.category .description` rule, add this transition:

```
.category .description {
 CODE OMITTED TO SAVE SPACE
 transition: all .2s;
}
```

- **all** means that we want to **transition** all properties (which includes the line-height or any other property that changes) on this element.
- **.2s** sets the transition's **duration** to last for 0.2 seconds.

This can be in seconds (**s**) or milliseconds (**ms**). So **.2s** is the same as **200ms**.

2. Save the file, then reload the page in your browser.

Hover over one of the featured art images to watch the description animate taller.

---

### Animating the Nav Links

Hover over the links in the header (John Schmidt and the links in the navbar) to see that they immediately change to green on hover. Let's make the color fade in instead of abruptly changing.

1. Back in your code editor, in the **header a** rule, add the code shown below in bold:

```
header a {
```

CODE OMITTED TO SAVE SPACE

```
 transition: all .3s;
```

```
}
```

2. Save and reload the page in your browser.
  - Hover over any of nav link to see the color smoothly transitions.
  - Hover off of the nav link to see the color transitions back to white.
  - Hover over **John Schmidt** to see that the shadow also animates!

---

## Adding Easing

Easing changes the rate of speed of an animation, going slower or faster at certain times. We can use: **ease**, **linear**, **ease-in**, **ease-out**, **ease-in-out**, and **cubic-bezier** (which lets us define our own). The default is **ease**, but let's try another.

1. Let's try easing on the photo description text where it will be easier to see. Back in your code editor, in the **.category .description** rule add **ease-in** as shown below:

```
.category .description {
```

CODE OMITTED TO SAVE SPACE

```
 transition: all .2s ease-out;
```

```
}
```

2. Save the file, then reload the page in your browser.
  - Hover over one of the images.
  - It's such a subtle change you probably can't notice a difference, but **ease-in** will start slow, then go faster at the end. The basic eases are very subtle, but we can define our own.

---

## Custom Easing with Ceaser

There are ways to customize the transition-timing-function (ease). Trying to figure out the coordinates of complex cubic Bézier curves is quite a headache, but luckily for us, developer (and former Noble Desktop instructor) Matthew Lein put together Ceaser (CSS Easer = Ceaser), a handy code generator that outputs the CSS for us.

1. In a new browser tab/window, go to [matthewlein.com/tools/ceaser](https://matthewlein.com/tools/ceaser)

2. From the **Easing** menu, choose **easeOutExpo**

This list of presets includes approximations of most Penner Equations. You can also create your own custom easing curve using the graph.

### Penner Equations

Robert Penner is a Flash developer who is well-known for his open source Flash innovations. The “Penner easing functions” are a de facto standard that now power numerous animation libraries in multiple languages (such as GSAP: GreenSock Animation Platform). For more info visit [robertpenner.com/easing](http://robertpenner.com/easing)

3. Set the **Duration** to **300** (that's 300 milliseconds or 0.3 seconds).
4. Click the **Width** button a few times to preview the look of this animation.
5. Select the **transition** values as shown below and **copy** it.

Code snippets, short and long-hand:

```
transition: all 300ms cubic-bezier(0.190, 1.000, 0.220, 1.000); /* easeOutExpo */
```

6. Switch back to **main.css** in your code editor.
7. In the **.category .description** rule, replace the transition's current values by pasting in the custom cubic-bezier values so the code looks like this:

```
.category .description {
```

CODE OMITTED TO SAVE SPACE

```
 transition: all 300ms cubic-bezier(0.190, 1.000, 0.220, 1.000);
}
```

8. Save and reload the page in your browser.

Easing is often subtle, but hover over the photos and hopefully you can notice the animation starts out fast and slows down quickly at the end. This feels snappier.

## Optional Bonus: Adding Focus Selectors to Hovers

It's often a good idea to add a `:focus` selector to any `:hover` selector. To save time we didn't do that, but let's do it now

1. Find the **.category a:hover .description** rule and change the selector to the following.  
Don't miss the comma after the first part!

```
.category a:hover .description,
.category a:focus .description {
```

CODE OMITTED TO SAVE SPACE

```
}
```

2. Find the **.logo-wrapper a:hover** rule and change the selector to the following.  
Don't miss the comma after the first part!

```
.logo-wrapper a:hover,
.logo-wrapper a:focus {
```

CODE OMITTED TO SAVE SPACE

```
}
```

---

## Exercise Preview



## Exercise Overview

In this exercise you'll learn how to use CSS to transform elements. You can **scale** them up or down, **rotate** them, and **translate** (move) them. Transforms can even be animated when combined with CSS transitions!

---

## Getting Started

1. If you completed the previous exercise, **index.html** should still be open from the **Artist Website** folder.

If you did not do the previous exercise, open the **Artist Website Ready for Transforms** folder in your code editor and open **index.html**.

2. Preview **index.html** in Chrome (we'll be using its DevTools).

In the previous exercise, we added some nifty transitions that smooth out property changes. While these effects look great by themselves, we can make the page even more exciting using transforms. Transforms are often combined with transitions, but they don't have to be.

---

## Testing Transforms Using the DevTools

1. Before we see how transforms and transitions work together, let's add a basic transform in the DevTools. **Ctrl-click** (Mac) or **Right-click** (Windows) on any of the large featured art images and select **Inspect**.

In the **Elements** panel of DevTools, an **img** or a **description** div should be selected. Click on the **<a href="#">** above it.

2. In the **Styles** panel, find the **.category a** rule with the **box-shadow** property.
  - Click in the empty space to the right of the last property **position: relative** to create a new blank line so we can type in a new property.
  - Type **transform: scale(.5)**

The scale transform sizes the element up or down. We are telling the browser to scale the element down to be 0.5 times (50%) its initial size.
  - Notice that the photos have scaled down to half of their original size. However, everything else on the page is still in the same place it was. This shows that transforms do not remove elements from the flow of the document.
3. Change the scale value in DevTools to be 1.4 time the normal size:

```
transform: scale(1.4);
```

Yikes, those are too huge! However, the layout is still unaffected even when the photos are larger than normal.
4. Close the DevTools, but leave the page open in Chrome.

---

## Adding a Scale Transform & Transitioning It

Let's scale both the featured art images and the overlays up on hover, using CSS transitions to create a smooth hover animation.

1. Back in your code editor, open **main.css** from the **css** folder.

We want to add a scale transform when the user hovers over links inside our **category** divs. This will size up both the images and the text.
2. Below the **.category a** rule, add the following new rule:

```
.category a:hover,
.category a:focus {
 transform: scale(1.2);
}
```



## CSS Transforms with Transitions

3. Save and reload the page in the browser.

Hover over one of the large photos on the left of the page (either **Oil** or **Mixed Media**). The transform works but notice these two issues:

- The image immediately enlarges to be 1.2 times its original size, and does not have an animated transitioned.
- The image is layered behind the image to its right. Hover over one of the images on the right to see it's layered above the left image (as we want all images to behave).
- This is because all links inside category divs are nested at the same level in the code. Elements further down the code will layer on top of previous elements.

4. Let's solve the stacking problem first. Leave the page open in the browser so we can come back to it later.

5. Back in your code editor, to ensure the `:hover` always stacks on top, add the `z-index`:

```
.category a:hover,
.category a:focus {
 transform: scale(1.2);
 z-index: 10;
}
```

6. Save and reload the page in your browser.

- Hover over all of the images to see that they are now always on top. Our `z-index` is working!
- Let's add a transition to create a smoother effect.

7. Return to **main.css** in your code editor.

We're already transitioning the **.description** elements. so let's add the same transition to the **.category a** elements.

In the **.category .description** rule, **copy** the transition property:

```
transition: all 300ms cubic-bezier(0.190, 1.000, 0.220, 1.000);
```

8. Remember that transitions go on the element that will be transitioned, not the state that triggers it. As shown in bold, paste the code into the **.category a** rule:

```
.category a {
 CODE OMITTED TO SAVE SPACE
 transition: all 300ms cubic-bezier(0.190, 1.000, 0.220, 1.000);
}
```

9. Save and reload the page in your browser.

- Hover over the images to see they now have an animation transition!
- This large size was good to test, but let's make it more subtle.

10. Back in your code editor, change the **.category a:hover** transform to a more subtle value:

```
.category a:hover,
.category a:focus {
 transform: scale(1.05);
 z-index: 10;
}
```

11. Save and reload the page in your browser.

Hover over any of the large images to see that the transform/transition combo looks more elegant now that they aren't sizing up as much.

---

## Transform-Origin

By default, elements transform from their center, but we can change the transformation's origin (reference or registration point).

1. Back in your code editor, in the **.category a** rule, add the following bold code:

```
.category a {
 CODE OMITTED TO SAVE SPACE
 transition: all 300ms cubic-bezier(0.190, 1.000, 0.220, 1.000);
 transform-origin: center bottom;
}
```

The x (horizontal) position comes first, followed by the y (vertical) position. This code tells the element to scale from its horizontal **center** and vertical **bottom**.

2. Save and reload the page in your browser.

Hover over any of images to see it scale up from the bottom edge's center point.

## Transform Origin Values

The value describes the **horizontal** and **vertical** positions. You can define values for the transform-origin property using the following:

- Keywords (left, right, or center for the horizontal position and top, bottom, or center for the vertical position), such as (right bottom).
- Percentages: 100% 100% is the same as the previous example.
- Pixel values: 100px 100px is 100 pixels from both the top and left. This approach is the most exact.
- A combination of these approaches, such as 50% bottom for 50% horizontal at bottom.

## Adding a Rotate Transform

In addition to scaling, we can also rotate elements.

1. Back in your code editor, in the **.category a:hover**, **.category a:focus** rule, add **rotate()** as follows:

```
transform: scale(1.05) rotate(-2deg);
```

2. Save and reload the page in your browser.
  - Hover over one of the images to see a slight rotation that starts from our specified transform-origin.
  - Positive values rotate clockwise, while negative values rotate counter clockwise.

## Using the Translate Transform to Nudge the Nav Links Up

1. Below **John Schmidt** at the top of the page, hover over the links in the navbar. They currently have a color transition, but let's also nudge them slightly upwards on hover.
2. Back in your code editor, in the **.category a:hover**, **.category a:focus** rule, delete the rotate so you end up with:

```
transform: scale(1.05);
```

3. Below the **nav a** rule, add the following new rule that translates (moves) the elements on the y-axis (vertically):

```
nav a:hover,
nav a:focus {
 transform: translateY(-5px);
}
```

NOTE: Negative values move an element up. Positive values move down.

4. Save and reload the page in Chrome.

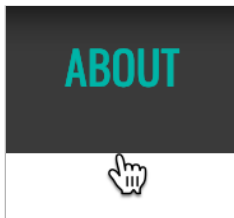
Hover over any of the navigation links to see the element smoothly animate upwards as the text color changes due to the transition/transform combo. Nice and subtle!

---

## Fixing Anchor “Flutter” by Translating a Child Element

There is a downside to the approach we just used, which will be most apparent if you are viewing the page in Firefox.

1. Preview **index.html** in **Firefox**.
2. Place the cursor so it just barely enters one of the links from the bottom, as shown below:



3. The text should be flickering in color and fluttering up and down. Move the cursor horizontally along the very bottom edge of the nav to see the unexpected movement. Yikes, that's distracting!

The flutter effect is happening as we hover over the bottom edge because we are moving the anchor tags upward. Once the anchor passes above the cursor, the hover style becomes inactive so it moves back to its normal position. The cursor is still at the bottom of the anchor so the event fires again, and so on and so forth.

4. Leave the page open in Firefox so we can reload it once we make changes.
5. Return to **index.html** in your code editor.

## CSS Transforms with Transitions

6. Wrap a **span** tag around the text in each link in the **nav**, as shown in bold:

```
<nav>

 Shows
 About
 Contact

</nav>
```

7. Save **index.html**.

8. Let's update the CSS to match the change. Switch to **main.css**.

9. Add **span** to the **nav a:hover**, **nav a:focus** selector:

```
nav a:hover span,
nav a:focus span {
 transform: translateY(-5px);
}
```

10. Save the file.

11. Switch back to Firefox, reload the page, and:

- Hover over the nav links to see that they are not moving up.
- This is because span tags are inline elements, which do not respond to transforms.

12. Back in your code editor, below the **nav a** rule, add the following new rule:

```
nav a span {
 display: inline-block;
}
```

13. Switch back to Firefox, reload the page, and:

- Hover over the very bottom of the nav links to see that they once again move up, but without the bad flicker.
- While the transform is working properly, we've lost the transition because our transitions were applied to nav links, not the span.

14. Back in your code editor, in the **nav a span** rule, add the code shown in bold:

```
nav a span {
 display: inline-block;
 transition: all 300ms;
}
```

15. Save the file and preview it in Chrome.

- Hover over the nav links. Notice the text slides up nicely, but the color transition happens after that. We want them to transition at the same time.
- Firefox does not have this issue, but Chrome and Safari do. We're not exactly sure why this is happening, but sometimes we've seen issues with transition **all**. We only need to transition our **transform**, so let's try being more specific.

16. In the **nav a span** rule, change **all** to **transform**:

```
nav a span {
 display: inline-block;
 transition: transform 300ms;
}
```

17. Save the file and preview it in Chrome.

Hover over the nav links one final time to see the nice smooth transition of movement and color at the same time!

---

# Responsive Images

## Exercise Preview



## Exercise Overview

In this exercise, you'll learn how to create responsive images, so users will get the most optimized image for their screen size. This helps pages to load faster, which leads to happier users. Page loading speed also affects your website's page rank in Google (faster loading pages rank higher)!

You'll learn two techniques: the **srcset** attribute for **img** tags, and the **picture** element. Here's a quick comparison:

- **Img Srcset:** We use an **img** tag with **srcset** when we want the same image to work across various screen sizes, but with the optimal image size/resolution. Mobile users get a small file, desktop users get a large size, and users with hi-res or low-res screens get the appropriate image.
- **Picture Element:** The **picture** element offers more control, or "art direction" over the images used across screen sizes. We use it when we want to provide different images, with different croppings or aspect ratios, such as a vertical image for mobile devices, and a horizontal image for desktops.

---

## Getting Started

1. For this exercise we'll be working with the **Responsive Images** folder located in **Desktop > Class Files > Advanced HTML CSS Class**. Open that folder in your code editor if it allows you to (like Visual Studio Code does).
2. Open **srcset.html** from the **Responsive Images** folder.

3. Preview **srcset.html** in Chrome (we'll be using its DevTools).

You should see one image with some text on the page. For this exercise we're going to work in a bare-bones page so it will be easier to focus on the code required for this technique.

4. Leave the page open in Chrome.

---

## Using Img Srcset for 1x & 2x (Retina) Graphics

We created a low-res (1x) and hi-res (2x) version of an image. Using the **img** tag's **srcset** attribute, we can tell the browser about our 1x and 2x images, and it will choose the appropriate one based on our screen resolution!

1. Switch back to **srcset.html** in your code editor.
2. Add the **srcset** attribute with our 1x and 2x images, as shown below in bold:

```

```

NOTE: The order of HTML attributes such as **srcset** and **src** do not matter, so **srcset** could come before or after **src**. The **src** attribute is for older browsers that don't understand the newer **srcset**. Refer to [caniuse.com/#search=srcset](http://caniuse.com/#search=srcset) for browser support.

3. Save the file, reload the page in Chrome, and:
  - If you're on a 1x display, you should see an image that says 320px. This is displayed at 320px wide.
  - If you're on a 2x display, you should see an image that says 640px. This is displayed at 320px wide, making it a 2x image!
  - Older browsers that don't support **srcset** will fall back to the **src** img, which is a 320px image that would be the same width in the page.

It's pretty awesome that the browser understands 1x and 2x and automatically chooses the proper image and size!



4. Let's double-check both screen resolutions to see it working first hand. Still in Chrome, **Ctrl-click** (Mac) or **Right-click** (Windows) anywhere on the page and select **Inspect** to open Chrome's DevTools.

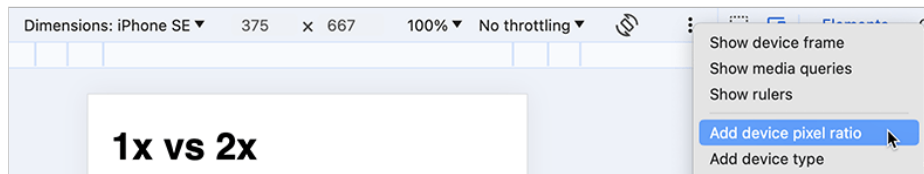
Once an image is downloaded, the browser may keep it in cache. This is good for users, but not for while we're testing. Let's disable Chrome's cache.


5. At the top right of the DevTools, click the gear button (**Settings**).
6. Scroll down and under **Network**, check on **Disable cache (while DevTools is open)**.
7. Close the settings by clicking the X at the top right.



## Responsive Images

8. In the upper left of the DevTools panel click the **Toggle device toolbar** button  to enter device mode.
9. From the device menu above the webpage, select **iPhone SE**.
10. Click the **Reload** button, or hit **Cmd-R** (Mac) or **Ctrl-R** (Windows).
11. This is a 2x device, so you should see the **640px** image.
12. Let's use with the device pixel ratio to change to a 1x display. This setting is not shown by default, so if you don't see it, enable it by clicking the 3 dotted button  and choosing **Add device pixel ratio**.



13. You should see a **DPR: 2.0** option to the right of the dimensions. It's grayed out though. From the device menu (to the left of the dimensions) choose **Responsive**.
14. The **DPR** menu will no longer be disabled. Click on the **DPR** menu and choose **1**.
15. Click the **Reload** button, or hit **Cmd-R** (Mac) or **Ctrl-R** (Windows).  
You should now see the **320px** image.
16. Click on the **DPR** menu and set it back to **Default: 2.0**.
17. Click the **Toggle device toolbar** button  to leave device mode.
18. Keep the DevTools open.

---

### Using Img Srcset with Sizes

There's another way to use srcset. We saved our image at multiple pixel widths (320, 640, 1280, and 2560). Using the img tag's **srcset** attribute, we can tell the browser about all the images and it will decide which is best to display... based on the size of the screen, size of the image in the page, screen resolution, and potentially the speed of the user's internet connection!

1. Switch back to **srcset.html** in your code editor.
2. Comment out the **1x vs 2x** heading and **img** tag below it.

```
<!-- <h1>1x vs 2x</h1>

```

3. Between the commented code and the paragraph below, add this heading and img:

```
<h1>Sizes</h1>

```

4. Add the **srcset** and **sizes** attributes, as shown below in bold:

```

```

5. We're going to be adding a lot of code, so break the new attributes onto new lines:

```

```

6. Add the list of images, as shown below:

```

```

NOTE: After each image is a number with a w. The w stands for width, and refers to the px width of the image. We need to tell the browser how many pixels wide each file is, so it doesn't have to download them to figure that out. It will use this info to determine which image it should download.

7. Add the list of sizes shown below in bold:

```

```


## Responsive Images

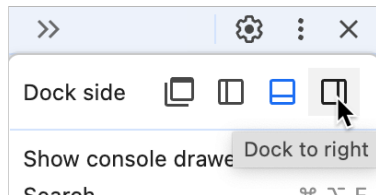
8. With **sizes** we tell the browser how big the image will be in the page, in relation to the viewport. This is similar to setting a width in HTML, in that it makes the image that size (although that can be overridden with CSS). More importantly, this tells the browser how big the image will be so it knows which size image to download.

Why doesn't the browser just figure out how big the image for us? The browser must decide which file to download before it renders the page and figures out how big the image will be. This is done to speed up the downloading of assets and rendering of the page. You can learn more about this (and responsive images) at [tinyurl.com/dev-onri](https://tinyurl.com/dev-onri)

Each size is a media condition, a space, and then the width will be in the final page layout. The width can be absolute (px, em) or relative (vw, which is viewport width), but cannot be percentages. You can even use CSS `calc()`.

The first media condition that is met will be used. The last size does not have a media condition so it will be used when none of the other conditions are met.

9. Save the file, and reload the page in Chrome.
10. Make your window as wide as possible.
11. If the DevTools are not docked to the right side of the browser window, at the top right of the DevTools panel click the  button and choose **Dock to right** as shown below:



12. Resize the DevTools panel to make the page area narrow (around 320px). You can see the size in the top right of the page as you resize the DevTools.
13. Click the **Reload** button, or hit **Cmd-R** (Mac) or **Ctrl-R** (Windows).
14. You should see the 320px image if you're on a 1x screen, or the 640px image if you're on a 2x screen.
15. Resize the DevTools area wider, and watch how the other sizes automatically load when appropriate.
16. Resize the DevTools area narrow, and notice the stays as the biggest image.  
  
Once the higher quality image is loaded it's already cached so there's no need to switch to a smaller image. They look the same, but are lesser quality so the browser doesn't bother switching.
17. Resize the page area to around **600px**.
18. Click the **Reload** button, or hit **Cmd-R** (Mac) or **Ctrl-R** (Windows).

19. You should see the 640px image if you're on a 1x screen, or the 1280px image if you're on a 2x screen.
20. Close the DevTools, but keep the page open in Chrome.

---

## Using the Picture Element for 1x & 2x (Retina) Graphics

Let's look at the picture element, which is another way to create responsive images. This method is good when you don't want the image to look the same across all screen sizes.

1. Switch back to your code editor.
2. Let's switch to another file in this folder. Open **picture.html**.
3. There's no content in this file. In the body tag, create a picture element, as shown below in bold:

```
<picture>
 <source media="(min-width: 1280px)" srcset="img/model-desktop-1280.jpg">
 <source media="(min-width: 768px)" srcset="img/model-tablet-768.jpg">

</picture>
```

NOTE: Browsers will pick the first source that matches. If none match, the img tag will be used.

"Unlike srcset and sizes, when you use the media attribute, you are dictating to the browser which source should be used. The browser has no discretion to pick a different source. It must use the first element whose media attribute matches the current browser conditions." — Cloud Four, [tinypurl.com/ri-part6](https://tinypurl.com/ri-part6)

4. Preview **picture.html** in Chrome.
5. Resize the page from narrow to wide, and you should see 3 images (labeled **Mobile 375px**, **Tablet 768px**, and **Desktop 1280px**) which are all different sizes, aspect ratios, and croppings.

We're seeing 1x images because those are the images we told it to use, but we can have the browser display the appropriate 1x/2x depending on our screen resolution.

6. Switch back to **picture.html** in your code editor.

## Responsive Images

7. We're going to be adding more code, so break the code onto new lines like this:

```
<picture>
 <source media="(min-width: 1280px)"
 srcset="img/model-desktop-1280.jpg">
 <source media="(min-width: 768px)"
 srcset="img/model-tablet-768.jpg">

</picture>
```

8. As shown below in bold, add 1x, a comma, and the 2x images:

```
<picture>
 <source media="(min-width: 1280px)"
 srcset="img/model-desktop-1280.jpg 1x,
 img/model-desktop-2560.jpg 2x">
 <source media="(min-width: 768px)"
 srcset="img/model-tablet-768.jpg 1x,
 img/model-tablet-1536.jpg 2x">

</picture>
```

9. Add **srcset** to the img tag, as shown below in bold:

```

```



10. Move **srcset** down to its own line and add images to it:

```
<picture>
 <source media="(min-width: 1280px)"
 srcset="img/model-desktop-1280.jpg 1x,
 img/model-desktop-2560.jpg 2x">
 <source media="(min-width: 768px)"
 srcset="img/model-tablet-768.jpg 1x,
 img/model-tablet-1536.jpg 2x">

</picture>
```

The **img** tag will be a fallback for when the other media conditions are not met, which would be screens less than 768px wide.

11. Save the file and reload the page in Chrome.
12. Resize the page from narrow to wide, and you should see 3 images, but this time you'll see 1x sizes (375px, 768px, and 1280px) if you're on a 1x display, or 2x sizes (750px, 1536px, and 2560px) on 2x displays.

13. Let's double-check both screen resolutions to see it working first hand. Still in Chrome, **Ctrl-click** (Mac) or **Right-click** (Windows) anywhere on the page and select **Inspect** to open Chrome's DevTools.
14. In the upper left of the DevTools panel click the **Toggle device toolbar** button  to enter device mode.
15. From the device menu above the webpage, select **iPhone SE**.
  - Click the **Reload** button, or hit **Cmd-R** (Mac) or **Ctrl-R** (Windows).
  - This is a 2x device, so you should see the **Mobile 750px** image.
16. From the device menu (to the left of the dimensions) choose **Responsive**.
  - Click on the **DPR** menu and choose **1**.
  - Click the **Reload** button, or hit **Cmd-R** (Mac) or **Ctrl-R** (Windows).
  - You should now see the **Mobile 375px** image. (If you don't, your mobile preview may be too wide. Resize it narrower and refresh.)
17. Click on the **DPR** menu and set it back to **Default: 2.0**.
18. Click the **Toggle device toolbar** button  to leave device mode.
19. Close the DevTools, but keep the page open in Chrome.

---

## Using the Picture Element with Sizes

Currently our picture element is only considering screen resolution, and not how large the image will be in the page. We can use **sizes** to take that into consideration.

1. Switch back to **picture.html** in your code editor.
2. As shown below in bold, add **sizes**:

```
<picture>
 <source media="(min-width: 1280px)"
 sizes="50vw"
 srcset="img/model-desktop-1280.jpg 1x,
 img/model-desktop-2560.jpg 2x">
 <source media="(min-width: 768px)"
 sizes="75vw"
 srcset="img/model-tablet-768.jpg 1x,
 img/model-tablet-1536.jpg 2x">

</picture>
```

## Responsive Images

3. Change the 1x and 2x to image sizes, as shown below in bold:

```
<picture>
 <source media="(min-width: 1280px)"
 sizes="50vw"
 srcset="img/model-desktop-1280.jpg 1280w,
 img/model-desktop-2560.jpg 2560w">
 <source media="(min-width: 768px)"
 sizes="75vw"
 srcset="img/model-tablet-768.jpg 768w,
 img/model-tablet-1536.jpg 1536w">

</picture>
```

The browser will choose appropriately based on screen width, image size in the page, screen resolution, etc:

- On screens 1280px and above, **model-desktop-1280.jpg** or **model-desktop-2560.jpg** will be displayed.
  - On screens 768px to 1279px, **model-tablet-768.jpg** or **model-tablet-1536.jpg** will be displayed.
  - On screens 375px or smaller, **model-mobile-375.jpg** or **model-mobile-750.jpg** will be displayed.
4. Save the file and reload the page in Chrome.
5. Resize the page from narrow to wide, and you'll see the images appear in the following order (which matches the bottom to top ordering in the picture element):

- **Mobile 375px** or **750px** is 100% wide and appears on small screens (depending on your screen resolution. This is the img with srcset.
- **Tablet 768px** or **1536px** appear starting at 768px (it's 75% wide).
- **Desktop 1280px** or **2560px** appear starting at 1280px (it's 50% wide).

What's important is that you're seeing different images change as you resize the window. Here's the big picture takeaway:

- With **img srcset** the image chosen by the browser will be used when the window is resized smaller. It does not automatically switch to the smaller images, because they are supposed to be the same images, just lower resolution.
- With the **picture** element, the image must change as the window gets smaller because those images are supposed to look different and you want to see those images because they look better at that size/aspect ratio, which effects the design.

## Summary

While the picture element lets us control which images are displayed at various sizes, it requires precisely made images that will fit into the proper places within a design. That can be useful sometimes, but overkill when it's not.

For most images, we simply want the browser to figure out which resolution image to use. So most of the time, an `img` tag with `srcset` will be the best choice. To learn more, read Cloud Four's article **Don't use `<picture>` (most of the time)** at [tinyurl.com/dupmott](https://tinyurl.com/dupmott)

### Media Queries & Sizes

The image sizes and media queries used in this exercise are not some magical set of numbers/sizes that you'll always use. Every layout is different, so be sure to adapt these concepts to your site and content!

### Remembering Which To Use

Having a hard time remembering which to use? This may help:

- **Srcset** = browser set (browsers are in charge of choosing the right file)
  - **Picture element** = picture it your way! (art direction, you're in charge and it will show every image you tell it to)
-





# Noble Desktop Training

## Learn live online or in person in NYC

Front-End Web Development

Full-Stack Web Development

JavaScript

Python

Software Engineering

Data Science & Data Analytics

SQL

WordPress

Motion Graphics & Video Editing

Adobe Premiere Pro

Adobe After Effects

InDesign, Illustrator, & Photoshop

Web, UX, & UI Design

Figma

Digital & Social Media Marketing

and much more...

---

**nobledesktop.com**