Advanced HTML & CSS

Learn by doing step by step exercises.

Includes downloadable class files that work on Mac & PC.

EDITION 4.0

noble desktop

Published by:
Noble Desktop
185 Madison Ave, 3rd Floor
New York, NY 10016
nobledesktop.com

Publish Date: 10-19-2021

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Downloading the Class Files

Thank You for Purchasing a Noble Desktop Course Workbook!

These instructions tell you how to install the class files you'll need to go through the exercises in this workbook.

Downloading & Installing Class Files

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 **wd2-2110-19**
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.

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).

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.

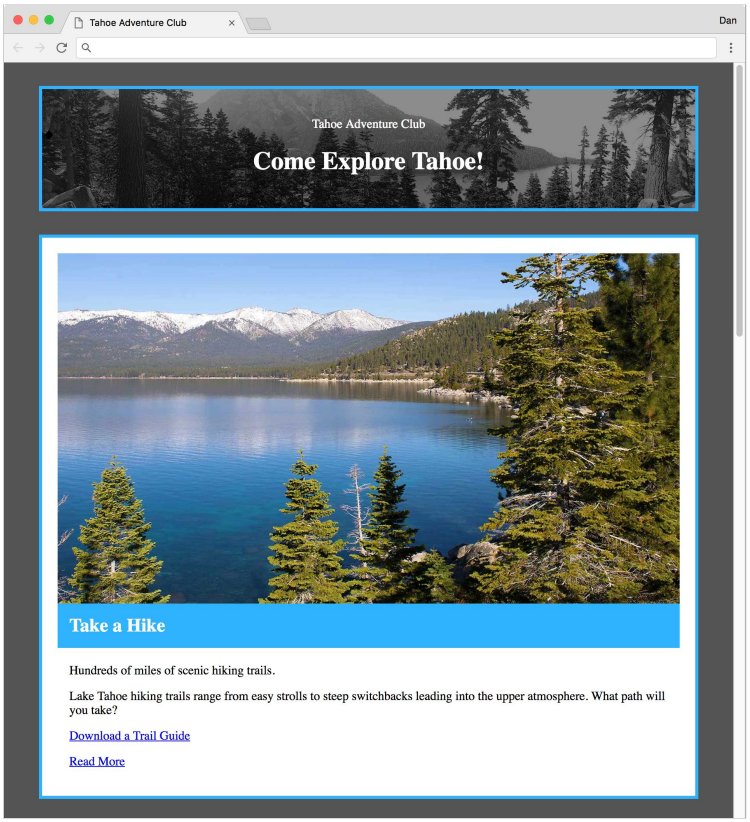
1. On the Mac, launch the **App Store** (it's found in the **Applications** folder).
2. Search for **Xcode** (which includes iOS Simulator) and install it. If you're on an older version of macOS you may need to upgrade your system to be able to run Xcode.
3. Once **Xcode** is installed **launch** it. (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.**

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 style a page using foundational CSS techniques. In this exercise you'll learn about normalize.css, the CSS Box Model (width, height, padding, margin, and borders), creating a fluid layout that works well across different screen sizes, and more.

Getting Started

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 Starter** folder located in **Desktop > Class Files > Advanced HTML CSS Class > Tahoe Starter**. You may want to open that folder in your code editor if it allows you to (like Visual Studio Code does).
3. Open **index.html** from the **Tahoe Starter** folder.
4. Preview **index.html** in Chrome. (We're using Chrome because we'll be using its DevTools later.)

5. Notice the following:
   - There's no styling yet, so it looks pretty rough.
   - You may have a horizontal scrollbar at the bottom of the page. When the window is smaller than the images, they do not scale down to fit in the confines of the browser window. We'll need to fix that.
6. Leave **index.html** open in the browser as you work, so you can reload the page to see the code changes you'll make.

Using Normalize.css

We typically have CSS that we add to every site. We may want to zero out margins on the body tag or fix differences across browsers. Many coders have developed **reset style sheets** to deal with this. Some can be aggressive and override browser defaults, such as removing bolds from headings and bullets from bulleted lists.

Instead of a reset CSS, we prefer to use **normalize.css** because it's less aggressive and doesn't try to override browser defaults. The developer says it "makes browsers render all elements more consistently and in line with modern standards. It precisely targets only the styles that need normalizing." We've provided a copy in the class files, but you can learn more or download it from **necolas.github.io/normalize.css**

1. Switch back to your code editor.
2. Open **normalize.css** from the **css** folder (in the **Tahoe Starter** folder).
3. The comments throughout the code explain why each rule is there. Look over some of the rules, such as:
   - **html** rule: The **text-size-adjust** prevents mobile browsers from enlarging text. The **line-height** ensures consistency across browsers. You may notice the line-height does not specify a unit (such as **px**). You'll learn about unitless line-height in the next exercise.
   - **body** rule: Removes the page's margins.

   While most of the rules standardize defaults across browsers without deviating from them too much, removing the body margins is opinionated. We typically want to do it, but technically that isn't keeping browser defaults (which have a default margin). It is one of the few places where it oversteps the normalize concept and acts more like a reset CSS.
4. Let's link our page to normalize.css. Switch back to **index.html** in your code editor.

5. Link to normalize.css as shown in bold below. TIP: If your code editor has Emmet installed (like Visual Studio Code does), you should be able to type **link** and hit **Tab** to get the base link tag and you'll just have to fill in the href.

```
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1">
   <title>Tahoe Adventure Club</title>
   <link rel="stylesheet" href="css/normalize.css">
</head>
```

6. Save **index.html**.

7. Switch back to Chrome and reload **index.html**. You won't see much change, except the images and text now touch the edge of the browser window because the body's margins have been removed.

   While there currently aren't many visible changes for this page, normalize contains many best practices and fixes so it's good to use.

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

   In the **css** folder, we've created an empty CSS file called **main.css** to which we'll add all of our custom CSS. Let's link our page to that as well. It's important that this link comes after **normalize.css** so our CSS can override normalize, if necessary.

9. In **index.html**, add the following bold code:

```
   <link rel="stylesheet" href="css/normalize.css">
   <link rel="stylesheet" href="css/main.css">
</head>
```

   NOTE: When linking to multiple style sheets, the order is crucial. Browsers read from top to bottom, so the later styles can override rules in the style sheet(s) above. If two CSS rules have the same specificity, such as a class vs. a class, or an ID vs. an ID, the latter rule will win over the first. We always put normalize.css first so we can override it in main.css.

10. Save the file.

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. Hi-res screens are called **HiDPI** (high dots per inch) or **Retina** (as Apple calls them) and typically have twice the number of pixels occupying the same amount of space on the screen. This technology was first uses on mobile phones and tablets, but are becoming more common on laptops and desktops.

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. Open the empty **main.css** file from the **css** folder.
2. To get rid of the horizontal scrollbar, we need to scale the images down to fit inside their parent container. In **main.css**, add the following rule for all images:

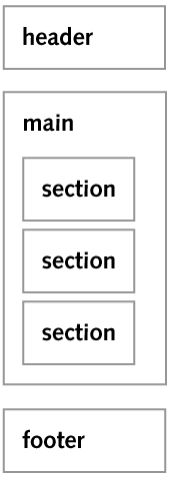
```
img {
   max-width: 100%;
}
```

   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.

Styling the Sections (Borders, Margins, & Padding)

index.html contains 3 primary elements: **header**, **main**, and **footer**. The **main** element contains 3 **section** elements. Here's a visualization of the structure:



We want the **header** and 3 **sections** to all look the same (border, spacing, etc.) so let's create a style to target both types of elements.

1. Return to **main.css** in your code editor.
2. Below the **img** rule, add the following new rule to target the **header** and all **section** elements:

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

   NOTE: Commas in CSS selectors are used to separate different elements that will all get the same styling. They do not increase the specificity of the selector, so we have two tag rules (they're just combined into a single rule to avoid redundancy).

3. Save **main.css**.
4. Switch back to Chrome, and reload **index.html**.

   You should now see four areas outlined in blue with space between them.

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;
}
```

   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. Add the code shown below in bold:

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

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

   The page should now be dark gray with 4 white areas that have blue borders. We don't like how the content within each of the bordered areas touches the blue border, so let's add padding (space **inside** the border) in addition to our existing margins (space **outside** the border).

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

10. Add **padding** as shown below in bold:

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

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

    That looks a bit better, but when the page is too wide the text is hard to read. Let's limit how wide it can get on large screens (but keep the width flexible to work on smaller screens).

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**. Add the **max-width** shown below in bold:

```
header, section {
   CODE OMITTED TO SAVE SPACE
   padding: 20px;
   max-width: 800px;
}
```

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 **auto**matically 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, section** rule we want to keep the top and bottom margin that we already have, but we want to set left and right margins to auto. Do that by adding **auto** as shown below in bold:

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

NOTE: When margin (or padding) has a single value, it's used for all sides (top, right, bottom, and left). When there are two values (as we did above), the first value is for both **top and bottom**, and the second value is for **right and left**.

3. Save **main.css**, switch back to Chrome, and reload **index.html**. Notice the following:

   - Make sure the browser window is wide enough so you can see the column of content is now centered.
   - The spacing is fine when the browser is wide, but if you make the browser narrow, you see that the borders bump right up against the left and right of the body. Let's give it some breathing room.

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

5. To add space around the page, on the **body** rule add **margin** as shown below in bold:

```
body {
   background: #555;
   margin: 30px;
}
```

6. Save **main.css**, switch back to Chrome, and reload **index.html**. Now there's some gray space around the content when the window is smaller.

Visualizing the Box Model in Chrome's DevTools

1. Let's make the h2s look like they're attached to the images above them. Switch back to **main.css** in your code editor.

2. After the rule for **img**, add the following rule:

```
h2 {
   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**.
6. Add padding on all four sides of the h2s, by adding the following code shown in bold:

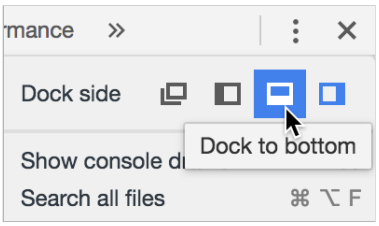
```
h2 {
    background: #2fb3fe;
    color: #fff;
    padding: 15px;
}
```

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

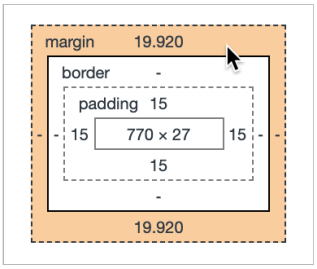
   The spacing inside the blue heading backgrounds (padding) looks a lot better, but we have a lot of space above them. The heading is not flush with the image, as we want. This is most likely caused by margin.

   Let's use developer tools to see what's causing the issue. All major browsers have these tools, but our instructions are written for Chrome's DevTools.

8. **Ctrl–click** (Mac) or **Right–click** (Windows) on the **Take a Hike** h2 and choose **Inspect**.
9. We want the DevTools docked to the bottom. If the DevTools are docked to the right side of the browser window, at the top right of the DevTools panel click the ⋮ button and choose **Dock to bottom** as shown below:



10. On the right side of the DevTools, choose the **Computed** tab to see the box model.
11. As shown below, hover over **margin**. Then in the page notice the margin above and below the h2 in the page will appear as orange.



   NOTE: The 19.920 pixels of margin is a default amount applied by the browser (which is proportionate to the font size).

12. Let's remove the default margin on the top. Switch back to **main.css**.

13. Zero out the **margin-top** by adding the following code shown below in bold:

```
h2 {
   background: #2fb3fe;
   color: #fff;
   padding: 15px;
   margin-top: 0;
}
```

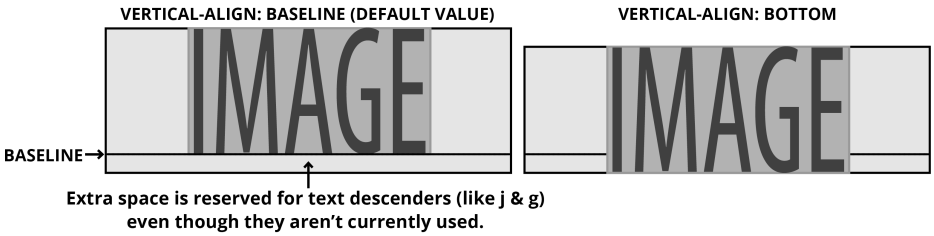
14. Save **main.css**, switch back to Chrome, and reload **index.html**. There's still a gap, even though the heading has no top margin. Perhaps the image has some margin?

15. **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 h2 look attached.

5. Keep the page open, but close Chrome's DevTools.

Paragraph Spacing

Let's align the paragraphs with the h2 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.
2. Below the **h2** 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.

Adding a Background Image to the Header

1. Switch back to **main.css** in your code editor.
2. The white header is boring, so we'll add a dark background image and make the text white. Below all the other rules, add the following new rule:

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

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 you only see some of the center part of the photo. Here's what the entire photo looks like (the photo below wasn't darkened, so it's easier for you to see):

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

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

6. Add the following bold code for background-size:

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

7. 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** declarations and any remaining whitespace.

3. We will add those values back in a moment, but to use the shorthand syntax you must first change **background-image** to **background**:

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

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;
```

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!

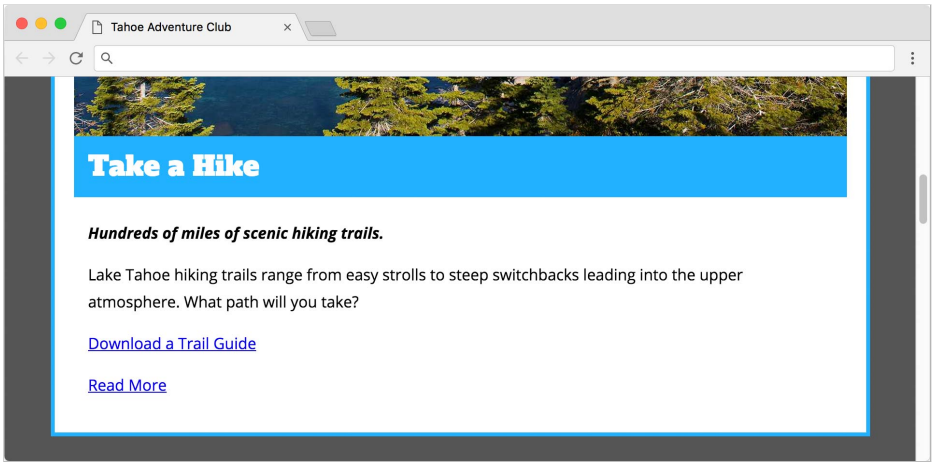
Optional Bonus: Styling the Footer

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 should be centered below the other sections.
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;
}
```

4. Save **main.css** and reload **index.html** in Chrome. It's looking better, but the typography could use improvement, which you'll do in the next exercise.

Exercise Preview



Exercise Overview

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

Getting Started

The files for this exercise are very similar to where the previous exercise left off, but we went ahead and set a couple font sizes to save some time.

1. In your code editor, close any files you may have open.
2. For this exercise we'll be working with the **Tahoe Typography** folder located in **Desktop > Class Files > Advanced HTML CSS Class**. You may want to open that folder in your code editor if it allows you to (like Visual Studio Code does).
3. Open **index.html** from the **Tahoe Typography** folder.
4. Preview **index.html** in a browser. Our design calls for some custom web fonts and better line spacing.
5. 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 font families we want to use (**Alfa Slab One** and **Open Sans**). In the search field at the top of the page, search for **Alfa**.

3. Click on **Alfa Slab One**.
4. On the right, click on **+ Select this style**.

   A column should appear on the right of the page listing the **Selected family**.

   NOTE: If you close this panel, you can reopen it by clicking the button at the top right of the page.
5. Let's find a second font:
   - At the top of the page click **Browse fonts**.
   - Search for **Open**.
   - Click on **Open Sans** in the search results.
6. On the right, click on **+ Select this style** next to these 3 styles:
   - Light 300
   - Regular 400
   - Bold 700 italic
7. You should see a column on the right of the page listing the **Selected families**. If you don't see it, open it by clicking the button at the top right of the page.
8. 2 fonts should be listed: **Alfa Slab One** (with 1 style) and **Opens Sans** (with 3 styles).

   Each style increases the download size and slightly slows down the loading of the page, so only load styles you'll be using.
9. Copy 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;0,400;1,700&display=swap"
rel="stylesheet">
```

10. Keep this page open, but return to **index.html** in your code editor.
11. Paste the link below the **title** tag, as shown below in bold:

```
<title>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;0,400;1,700&display=swap"
rel="stylesheet">
<link rel="stylesheet" href="css/normalize.css">
```

12. 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. Under **CSS rules to specify families**, 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: 30px;
      font-family: 'Open Sans', sans-serif;
   }
   ```

6. We want our two heading levels (h1 and h2) to be the Alfa Slab One font. Above the **h1** rule, add the following new rule so we can style both with one rule:

   ```
   h1, h2 {

   }
   ```

7. Return to Google Fonts in the browser.
8. Under **CSS rules to specify families**, copy the code for the **Alfa Slab One** font:

   ```
   font-family: 'Alfa Slab One', cursive;
   ```

9. Close the Google Fonts browser tab.
10. You should still have **index.html** open in the browser. Leave it open so you can reload the page to see the changes you make in the code.
11. Return to **main.css** in your code editor.

12. Paste the code into the rule for **h1, h2** as shown below:

```
h1, h2 {
   font-family: 'Alfa Slab One', cursive;
}
```

13. There are no good, consistent cursive fonts across platforms, so we don't recommend using cursive as a fallback. 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;
```

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

15. Notice that the headings look extra thick (especially around the **x** in Come 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.

Font-Weight & Font-Style

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

2. In the **h1, h2** rule add the following code shown below in bold:

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

3. 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 Come Explore).
   - After each h2 there are some paragraphs, but the first paragraph is a brief tagline. We want that to stand out from the others. It always follows an h2, so we can style it using an adjacent sibling selector (or next-sibling selector).

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

5. Below the **h2** rule, add the following new rule:

```
h2 + p {
   font-weight: 700;
}
```

NOTE: The **normal** keyword is the same as the numeric **400** weight. **700** is the same as **bold**. We loaded a **bold 700 italic** font, so we're using the 700 weight to be precise. For fonts that only have normal and bold weights, values from 100–500 are rendered as normal, and 600–900 are bold. For more info, see **tinyurl.com/moz-fw**

6. That makes the text **bold**, but we loaded a **bold italic** font. Add the following code (shown in bold) to make the text italic:

```
h2 + p {
   font-weight: 700;
   font-style: italic;
}
```

   NOTE: The font-style property 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.

7. Save **main.css** and reload **index.html** in the browser. The first paragraph after the blue headings should be bold and italic.

Unitless Line-Height

1. Resize the browser window smaller so the **Come Explore Tahoe!** in the header breaks onto multiple lines. Notice that 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 {
   background: #555;
   margin: 30px;
   font-family: 'Open Sans', sans-serif;
   line-height: 28px;
}
```

4. Save the file and reload **index.html** in the browser. The paragraphs are better, but the headings (especially **Come Explore 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 **Come Explore 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, 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 be them (instead of px values) from now on, and recommend you use them in your work.

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

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

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

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

Optional Bonus: More Practice

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.

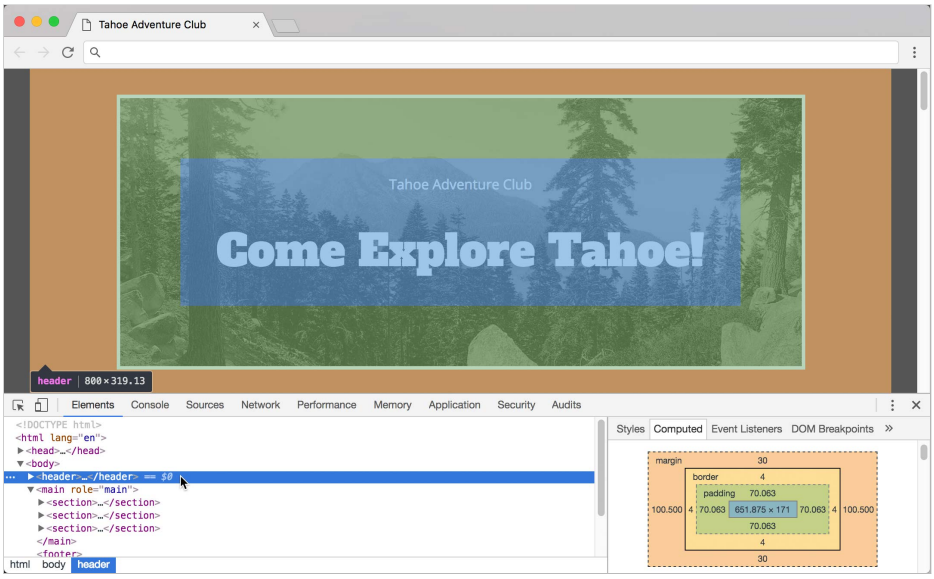
3. In the **footer** rule, add font-weight as shown below in bold:

```
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.

Exercise Preview



Exercise Overview

In a previous exercise we saw how the default box model treats margins, padding, and widths. In this exercise you'll learn about a newer (and often better) box model!

Getting Started

The files for this exercise are very similar to where the previous exercise left off, but we went ahead and did a little more text styling.

1. In your code editor, close any files you may have open.
2. For this exercise we'll be working with the **Tahoe Box Model** folder located in **Desktop > Class Files > Advanced HTML CSS Class**. You may want to open that folder in your code editor if it allows you to (like Visual Studio Code does).
3. Open **index.html** from the **Tahoe Box Model** folder.

Flexible Padding: Using Percentage Amounts

1. Preview **index.html** in a browser.
2. Currently there is 20px of padding in the header and 3 blue bordered sections below it. The header has plenty of padding on small screens, but we'd like to add more on large screens. While we could add media queries to increase the padding, let's try something easier: using a flexible percentage amount instead of a fixed pixel amount.

3. Keep the page open in the browser, but return to your code editor.
4. Open **main.css** from the **css** folder.
5. In the **header** rule (near the bottom), add padding as shown below in bold:

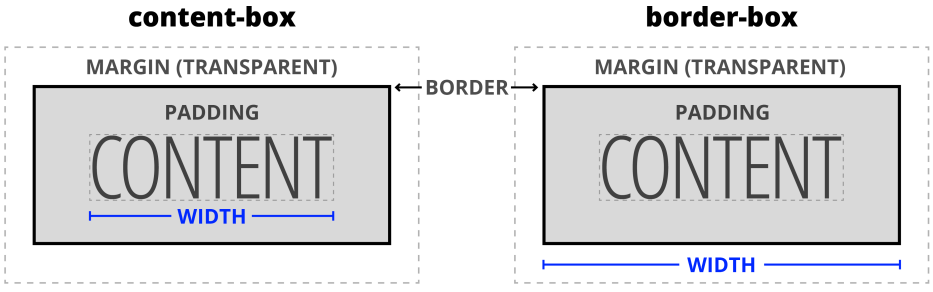
```
header {
   background: url(../img/masthead-darkened.jpg) center / cover;
   color: #fff;
   text-align: center;
   padding: 7%;
}
```

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

6. 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 3 sections 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*2 + 4*2).

Because padding is added to the max-width, when the 7% padding is bigger than 20px, 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 on the left of the DevTools.
4. In the **Styles** panel on the right of the DevTools, the top style should be **header**.
5. Below that is the **header, section** rule that we want to edit. The last property should be **max-width**. Click once on the **800px** value.
6. Hit **Tab** to add a new property below that.
7. Type **box-sizing** and then hit **Tab** to jump its value.
8. Type **border-box** and hit **Tab** to apply it so you end up with the following:

```
header, section {
   border: ▶ 4px solid #2fb3fe;
   margin: ▶ 30px auto;
   background: ▶ #fff;
   padding: ▶ 20px;
   max-width: 800px;
   box-sizing: border-box;
}
```

9. In the page notice the following:
   - The sections got a little 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!
10. Now we must do this in our CSS file, so return to **main.css** in your code editor.
11. 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 could target all elements using the asterisk (*) selector, but there's an even better way that web developers have figured out. We have the code in a snippet to make it easy for you to add. Let's grab the CSS from our code snippet. From the **snippets** folder, open **border-box.css**.
12. Select all the code.
13. Copy the code.
14. Close the file. You should be back in **main.css**.

15. Paste the code above all the other rules (as shown below):

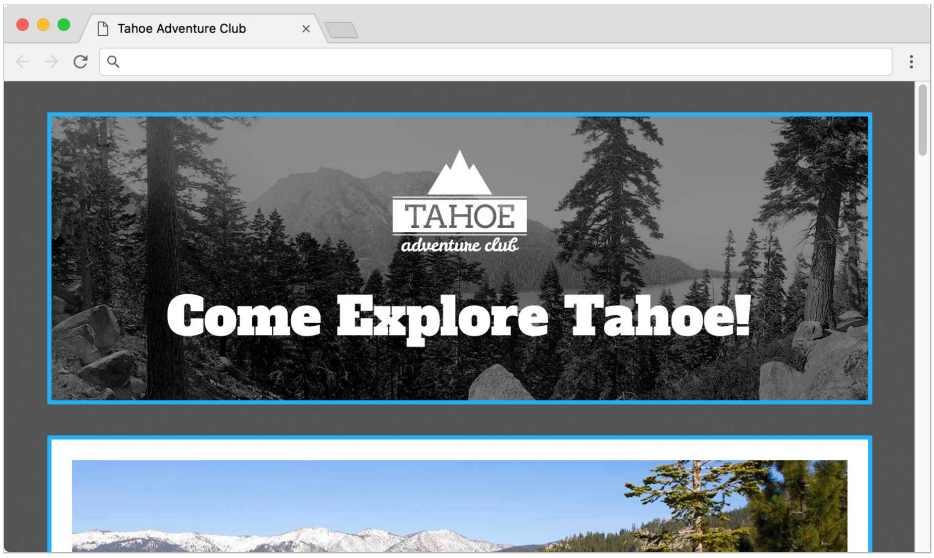
```
*, *::before, *::after { box-sizing: border-box; }

body {
```

NOTE: What does this code mean? The asterisk (*) targets everything. We then also target content added before or after any element (which you'll learn more about later in this training). This way everything gets border-box from a single CSS rule.

16. Save **main.css** and reload **index.html** in the browser. When the browser window is wide, the header and 3 sections below should all be the same width.

We recommending including this code in every site you create.

Exercise Preview



Exercise Overview

In this exercise you'll learn about SVG (scalable vector graphics). High resolution pixel-based graphics can have large file sizes. Vector graphics typically have smaller file sizes. With SVG we can use vectors natively in webpages! While not everything is vector-based (photos are still best as pixel-based files), graphics such as icons and logos are best created as vectors, and therefore saved as SVG.

Getting Started

The files for this exercise are very similar to where the previous exercise left off, but we've added some new files.

1. In your code editor, close any files you may have open.
2. For this exercise we'll be working with the **Tahoe SVG** folder located in **Desktop > Class Files > Advanced HTML CSS Class**. You may want to open that folder in your code editor if it allows you to (like Visual Studio Code does).
3. Open **index.html** from the **Tahoe SVG** folder.
4. Preview **index.html** in a browser.
5. Notice at the top of the page it says **Tahoe Adventure Club**. We want to replace this with the company logo.

Adding SVG to a Webpage

We created two different versions of the company's logo. Let's see how both look in the page. We can put an SVG into a page using a standard **img** tag.

1. Return to your code editor.
2. On line 15, replace the text with an **img** tag, as shown below in bold:

```
<header>
   <p>
      <img src="img/tahoe-logo.svg" alt="Tahoe Adventure Club">
   </p>
```

3. Save the file.
4. Preview **index.html** in a browser to see the logo is there. That was as easy as adding any image. The main difference is that this is vector instead of pixels, so we could make it any size without losing quality (and it looks great on any resolution screen)!
5. Keep the page open in the browser, but return to your code editor.
6. To switch to the other logo we made, on line 15, add **-white** to the filename as shown below in bold:

```
<img src="img/tahoe-logo-white.svg" alt="Tahoe Adventure Club">
```

7. Save the file.
8. Preview **index.html** in a browser to see the logo is there, but this time it's huge!

   Even though the logo is bigger than we designed it, the edges are still crisp (on any resolution screen) because it's made of vectors.
9. Resize the browser to see the logo scales down or up to always fill the header (which has some padding so the content doesn't touch the blue border).

Sizing SVG

When exporting SVG from Adobe Illustrator, there's a **Responsive** option that controls the size. When **Responsive** is **not checked** it codes a width and height 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 the 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 for two reasons:

- Microsoft Edge sometimes has issues when no width and height are in the SVG.
- We always have to code a size, instead of it defaulting to the size it was created at.

Because of those issues, when exporting SVG files from Illustrator we typically recommend you uncheck 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: **Sketch** and **Adobe XD** code width and height into SVG files.

1. Return to your code editor.
2. 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 the wall of code! It's mostly coordinates for all 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 with many similarities, so the syntax should not look completely foreign. While you'll typically create SVG files using a graphics app, you could hand code them.
3. In the first tag (**svg**) find the **viewBox="0 0 256 195.13"** part, which tells us the SVG is 256 pixels wide by 195.13 pixels tall.

   NOTE: The numbers are coordinates. The top-left corner of the box starts at **0 from the left** and **0 from the top** and the bottom-right corner ends at **256 pixels from the left** and **195.13 pixels from the top**.
4. We need to add width and height attributes, so add the folding **bold** code:

   ```
   viewBox="0 0 256 195.13" width="256" height="195.13">
   ```
5. Save **tahoe-logo-white.svg** and close it.
6. Return to the browser 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.
7. Return to your code editor.
8. Open **main.css** from the **css** folder.
9. There are no other images in the header so that will make it easier to target it wth CSS. Near the bottom of the file, after the **header** rule, add the following new rule:

   ```
   header img {
      width: 130px;
   }
   ```
10. Save **main.css** and reload **index.html** in the browser. 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
- **Adobe XD**: When exporting set **Styling** to **Presentation Attributes** and check on **Optimize File Size (Minify)**.
- **Sketch**: Use the standard method of exporting.

Web Servers: Configuring a .htaccess File for SVG & Gzip

SVG files will display locally, but on some web servers they may not work on the live site. Apache is a very common web server, and it may not be configured properly for SVG files. If you upload your files and the SVG files do not display on the live site, you'll need to fix the configuration of Apache servers using an **.htaccess** file. This file is typically invisible on Unix based computers (such as Macs) but most code editors will let you create, see, and edit it.

There are many things you can control with an .htaccess file, and HTML5 Boilerplate has a starter .htaccess file with all the code we need! We visited html5boilerplate.com and clicked the **Source Code** button to go to their GitHub page. In the **dist** folder is a **.htaccess** file from which we copied two relevant sections and saved the code into a file for you.

1. In your code editor, open the **.htaccess** file inside the **Tahoe SVG** folder.
2. On line 12 notice the **AddType image/svg+xml svg svgz** code.

   This tells the server about SVG files, so it will serve them properly. You don't have to change anything here, we just wanted to point out the code.
3. Starting on line 17, notice there's a section to enable compression.

   When **gzip** compression is enabled, the web server will compress certain types of files, send them to the browser, and the browser will decompress them. This means smaller files and faster downloads. Most browsers support gzip, so most people will benefit. If the browser does not support gzip, the server will send the normal uncompressed files, so you're fine either way!

   This can save a lot of file size for text files such as HTML, CSS, JS, and SVG! Pixel-based graphics like JPEGs do not get gzipped (they were already compressed). If you have an SVG file that is larger than a hi-res PNG, it will probably end up being smaller once gzip is enabled.

4. You'll need to upload the **.htaccess** file to your server, so keep the following in mind:

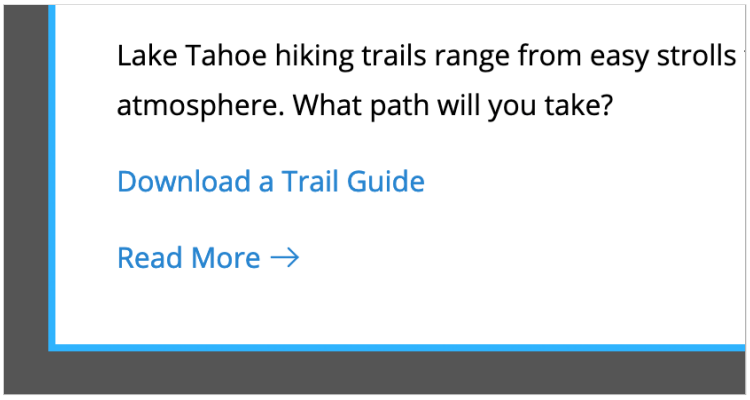
   - If you're on a Mac you can't see invisible files such as .htaccess on the Desktop, unless you hit **Cmd–Shift–Period(.)** to toggle the display of invisible files. (Hit the keystroke again to hide invisible files.)
   - FTP apps that have a two-pane view (local and remote files) often show invisible files, so you should be able to see it there. If not, you may have to enable them with a preference.
   - FTP apps that have a single-pane view (such as Cyberduck), typically have an upload feature (such as **File > Upload**). There you can usually show hidden files.

About HTML5 Boilerplate

HTML5 Boilerplate bills itself as "The web's most popular front-end template". It is a template for front-end development that was originally created by Paul Irish and Divya Manian. This open source project combines the knowledge and effort of many developers.

It contains many best practices, and in the past we used it as a starting point for our initial website folder (containing index.html, js, css, etc). Now we cherry-pick only a few parts that we want. If you would like to investigate HTML5 Boilerplate, visit **html5boilerplate.com**

Exercise Preview



Exercise Overview

In the previous exercise we introduced you to SVG, but 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 that you'll learn about in this exercise.

Getting Started

The files for this exercise are very similar to where the previous exercise left off, but we've added some new files.

1. In your code editor, close any files you may have open.
2. For this exercise we'll be working with the **Tahoe Embedding SVG** folder located in **Desktop > Class Files > Advanced HTML CSS Class**. You may want to open that folder in your code editor if it allows you to (like Visual Studio Code does).
3. Open **index.html** from the **Tahoe Embedding SVG** folder.
4. On line 29 notice the **Read More** link has an **arrow.svg** after the text.
5. Preview **index.html** in a browser.
6. In the **Take a Hike** section notice the **Read 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.
7. Leave the page open in the browser and return to your code editor.

Positioning the Arrow

1. Open **main.css** from the **css** folder.
2. Near the bottom, after the **header img** rule, add the following new rule:

```
.read-more img {
   vertical-align: baseline;
   margin-left: 5px;
}
```

3. Save **main.css** and reload **index.html** in the browser. 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 **Read 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**.
   - In Sublime Text, choose **Find > Replace**.
   - In Atom, choose **Find > Find in Buffer**.
5. Enter the following (the options may be named differently in your code editor):

   Find: **`id=`**
   Replace: **`class=`**
6. Proceed to **Replace All**, and a total of 3 replacements should be made.

7. Save the file.
8. Select all code.
9. Copy it.
10. Close this file (**arrow.svg**).
11. Switch to **index.html** here in your code editor.
12. On line 29 delete the **img** tag for **arrow.svg** so you end up with the following:

```
<p><a href="#" class="read-more">Read More</a></p>
```

13. Make sure the cursor is after **Read More** (so it's before ****).
14. Paste the SVG code.
15. Save the file.
16. Return to the browser and reload **index.html**.
17. In the **Take a Hike** section, after the **Read 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.
18. Return to **main.css** in your code editor.
19. In the **.read-more img** selector, change **img** to **svg** as shown below:

```
.read-more svg {
```

20. Save **main.css** and reload **index.html** in the browser. The arrow should once again be positioned correctly.
21. We need the arrow in two more places, so switch back to **index.html** in your code editor.
22. Around line 38, put the cursor after **Read More** (so it's before ****).
23. Paste the SVG code.
24. Around line 47, put the cursor after **Read More**.
25. Paste the SVG code again.
26. Save the file and reload **index.html** in the browser. All three of the Read More links should have an arrow after them. Now let's work on styling them.

Styling SVG Using CSS

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

2. 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.
3. We can use CSS to style SVG elements, so switch to **main.css**.
4. 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 **.read-more svg** rule, add the following new rule:

   ```
   .read-more svg * {
      stroke: red;
   }
   ```

   The asterisk (*) selector targets all elements (in this case all elements within the **svg**, which are in a **read-more** 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.
5. Save the file and reload **index.html** in the browser. 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.
6. 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**:

   ```
   .read-more 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 case-insensitive though, so it doesn't matter whether you write it as **currentColor** or **currentcolor** or any other case.
8. Save the file and reload **index.html** in the browser to see:
   - The arrow should be the same color as the **Read 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:

```
.read-more svg * {
   stroke: currentColor;
   stroke-width: 1.2;
}
```

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

Adding a Hover

Let's change the color of the **Read 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 **.read-more svg** rule, add the following new rule:

```
.read-more:hover {
   color: red;
}
```

3. Save the file and reload **index.html** in the browser.
4. Hover over a **Read 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 in CSS and JavaScript), 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 **.read-more:hover** selector, add **.read-more:focus** following new rule:

```
.read-more:hover, .read-more:focus {
```

7. Save the file and reload **index.html** in Chrome (which we know you can keyboard navigate).
8. Hit **Tab** to see the logo at the top becomes highlighted.
9. Hit **Tab** a couple more times until the **Read 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 the 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

In this exercise, you'll learn how to create and use SVG sprites.

What is a sprite? Taken from video games, this concept puts many small graphics (called sprites) together into a single graphic. When using pixel-based graphics we'd make one file which contains all the individual graphics (sprites). We would then crop in on each sprite and use it when needed. SVG makes sprites even easier to create and use than pixel-based sprites.

Why don't we just use separate image files? For users, loading multiple files will take longer than loading one. By reducing the number of files required, users will have a faster loading page. Sprites are better for performance, so that's why we use them!

Getting Started

1. For this exercise we'll be working with the **SVG Sprites** 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 **sprites.html** from the **SVG Sprites** folder.
3. Preview **sprites.html** in a browser.

   You should only see a heading **SVG Sprites** with nothing else 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 the browser and return to **sprites.html** in your code editor.

Defining the SVG Sprite

SVG are created with code. We can store the code in a reusable form, which will then allow us to use it anywhere within a page (as many times as we want). We'll start by creating a list of definitions for each graphic (sprite). We'll put this at the top of the file, so we can refer to those definitions lower in the file.

1. Below the **body** tag, add the following bold code:

```
<body>
   <svg xmlns="http://www.w3.org/2000/svg" display="none">

   </svg>

   <h1>SVG Sprites</h1>
```

This is where we'll list all the individual graphics. They aren't meant to appear here, so we're adding **`display="none"`** to make sure people won't see them here in addition to wherever we'll use them in the page.

2. Inside the **svg** tag, add the following bold code:

```
<svg xmlns="http://www.w3.org/2000/svg" display="none">
   <defs>

   </defs>
</svg>
```

Mozilla's MDN web docs say "The `<defs>` element is used to store graphical objects that will be used at a later time. Objects created inside a `<defs>` element are not rendered directly. To display them you have to reference them (with a `<use>` element for example)." **tinyurl.com/svg-defs**

3. Now we can add the SVG code for our first graphic. Still in your code editor, open **icon-twitter.svg** from the **SVG Sprites** folder.
4. Select all the code and copy it.
5. Close the file and you should be back in **sprites.html**.
6. Place the cursor on the empty line inside the **defs** tag.

7. Paste the SVG code so you end up with the following:

```
<svg xmlns="http://www.w3.org/2000/svg" display="none">
   <defs>
      <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24">
         <title>Twitter</title>

            CODE OMITTED TO SAVE SPACE

      </svg>
   </defs>
</svg>
```

8. We can put as many graphics into a sprite as we want, but let's put one more graphic into our sprite for practice. Still in your code editor, open **icon-facebook.svg** from the **SVG Sprites** folder.

9. Select all the code and copy it.

10. Close the file and you should be back in **sprites.html**.

11. Paste the new SVG code below the **svg** tag for Twitter, so you end up with:

```
<svg xmlns="http://www.w3.org/2000/svg" display="none">
   <defs>
      <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24">
         <title>Twitter</title>

            CODE OMITTED TO SAVE SPACE

      </svg>
      <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24">
          <title>Facebook</title>

            CODE OMITTED TO SAVE SPACE

      </svg>
   </defs>
</svg>
```

12. The containing SVG tag has **xmlns="http://www.w3.org/2000/svg"** so remove the redundant attributes from the two nested **svg** tags. You should end up with the code shown below:

```
<svg xmlns="http://www.w3.org/2000/svg" display="none">
   <defs>
      <svg viewBox="0 0 24 24">
         <title>Twitter</title>
            CODE OMITTED TO SAVE SPACE
      </svg>
      <svg viewBox="0 0 24 24">
         <title>Facebook</title>
            CODE OMITTED TO SAVE SPACE
      </svg>
   </defs>
</svg>
```

13. Change both **svg** tags to be a **symbol** tag, as shown below. Don't forget to update the open and close tags for both!

```
<svg xmlns="http://www.w3.org/2000/svg" display="none">
   <defs>
      <symbol viewBox="0 0 24 24">
         <title>Twitter</title>
            CODE OMITTED TO SAVE SPACE
      </symbol>
      <symbol viewBox="0 0 24 24">
         <title>Facebook</title>
            CODE OMITTED TO SAVE SPACE
      </symbol>
   </defs>
</svg>
```

14. We need a way to refer to each symbol, so give each symbol a unique ID, as shown below in bold:

```
<symbol id="twitter" viewBox="0 0 24 24">
   <title>Twitter</title>
   CODE OMITTED TO SAVE SPACE
</symbol>
<symbol id="facebook" viewBox="0 0 24 24">
   <title>Facebook</title>
   CODE OMITTED TO SAVE SPACE
</symbol>
```

NOTE: If you get SVG files that already have an ID, you can either use it or change it to whatever you want.

15. Save the file.
16. Preview **sprites.html** in a browser and you should only see the heading **SVG Sprites** with nothing else on the page. That's correct at this point!

Using a Sprite

The definition of our sprites is done, so now we can use them wherever we want in the page, and as many times as we want!

1. Return to **sprites.html** in your code editor.
2. Below the sprite definitions you'll see the only content currently in this page. Below that h1 tag add the following code to reference the twitter icon.

```
   <h1>SVG Sprites</h1>
   <svg class="social-icon">
      <use xlink:href="#twitter">
   </svg>
</body>
```

NOTE: The **class="social-icon"** is not required for the sprite to work, but we're adding it now so we'll be able to style it in a moment.

3. Save the file.
4. Preview **sprites.html** in a browser and you should see the Twitter logo below the **SVG Sprites** heading!
5. Return to **sprites.html** in your code editor.

6. Add the Facebook icon by writing the code shown below in bold:

```
   <h1>SVG Sprites</h1>
   <svg class="social-icon">
      <use xlink:href="#twitter">
   </svg>
   <svg class="social-icon">
      <use xlink:href="#facebook">
   </svg>
</body>
```

7. Save the file.

8. Preview **sprites.html** in a browser and you should the Twitter and Facebook logos.

Styling Sprites

The **svg** tags we just added (the ones with the **use** tag) don't have width and height attributes so there's extra space being added. Instead of adding those attributes to the HTML, let's use CSS.

1. In the **style** tag at the top of this file, below the **body** rule, add the following bold code:

```
.social-icon {
   width: 50px;
   height: 50px;
   margin-right: 10px;
   vertical-align: middle;
}
```

2. Save the file.

3. Preview **sprites.html** in a browser.

   The icons should now be smaller, with less space between them. They are vector, so we can make them any size and they'll look great.

4. Return to **sprites.html** in your code editor.

5. As we learned in the previous exercise, we can change the color of embedded SVG. That works exactly the same for sprites. Let's color the Facebook logo, but first we'll need a way to identify the place we are using it in the page.

   Don't confuse the ID used to define the sprite, with its instance on the page. You can reuse the same definition in many places on the page. You'll need a way to refer to those instance. We'll do that with a class.

6. Elements can have multiple classes (each class is separated by a space). Add a **facebook** class as shown below in bold:

```
<svg class="social-icon facebook">
   <use xlink:href="#facebook">
</svg>
```

7. In the **style** tag, below the **.social-icon** rule, add the following bold code:

```
.social-icon.facebook {
   fill: #3b5998;
}
```

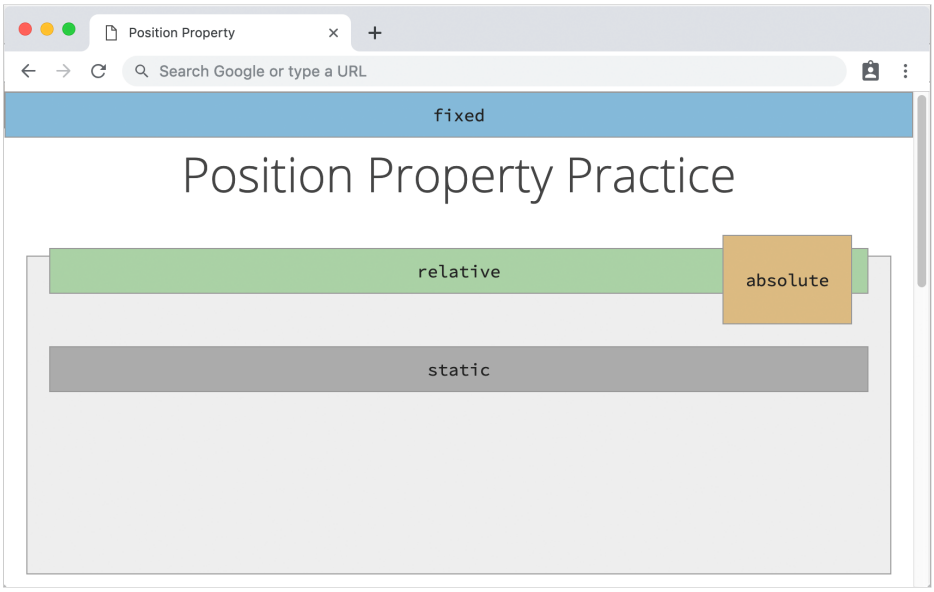
   NOTE: There's no space between **.social-icon** and **.facebook** which means the element we're targeting must have both classes. If the selector had a space, it would target element with a class of **facebook** inside an element with a class of **social-icon**.

8. Save the file.
9. Preview **sprites.html** in a browser and the Facebook logo should be blue.
10. Return to **sprites.html** in your code editor.
11. While we're at, we could make the Facebook logo slightly smaller.
12. In the rule you just created, add the following bold code:

```
.social-icon.facebook {
   fill: #3b5998;
   width: 46px;
   height: 46px;
}
```

13. Save the file.
14. Preview **sprites.html** in a browser and the Facebook logo should be a bit smaller than it was previously.

Exercise Preview



Exercise Overview

Browsers 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 called the **normal document flow**.

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

The Static Value & 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.

Position: Relative

1. Switch to your code editor.
2. Open **main.css** from the **css** folder (in the **Position Property** folder).
3. In the **.relative-div** rule that starts around line 26, declare a relative position:

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

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

   Declaring a relative position in and of itself does not do anything. Thankfully, this makes it relatively harmless, as it cannot break your layout. However, declaring this value does allow us to utilize a set of CSS properties that would otherwise have no effect on static elements. Specifying **top**, **right**, **bottom**, and/or **left** properties allow you to move any element that is not statically-positioned. The value we are using is named **relative** because these coordinates are determined **relative to** the element's natural position in the normal document flow.

6. In order to demystify what that means, 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.

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 on the right, find the **.relative-div** rule.

11. Try checking **position: relative;** off and on to see that the top coordinate has no effect when it is checked off and once more becomes static.

    Make sure the property declaration is checked on before you go on to the next step.

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

13. 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.

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

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

16. 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.

If we gave the top property a negative value, it would move **in the same direction**, moving up instead of down the page. A negative left position moves the element further to the left. The **top** and **left** properties are so useful that it's usually not necessary to set a bottom or right coordinate.

Position: Absolute

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 that starts around line 34, add the following bold code:

```
.absolute-div {
   background: #dcbb81;
   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 and the absolute div lost its natural width, collapsing to be only as big as it needs to be to hold the content inside it. That's similar to what happens with a floated element. Like floats, absolutely-positioned elements are **removed from the normal document flow** and other elements are no longer aware of it. It may seem horrible, but that's why it's the best candidate for elements that you want to move anywhere in the document.

6. Let's see if we can salvage this mess. Switch back to **main.css**.

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. Oh no, where'd it go? Strange, it seems like the absolute div just vanished into thin air.

12. Scroll up to the very top of the page to see that the div did in fact move down 40 pixels… but from the top of the page! So how can we control an absolutely-positioned element?

The Dynamic Duo: Relative Parent, Absolute Child

Absolutely-positioned elements are freed from the restriction of their position in the normal document flow, which means that unlike relatively-positioned elements, they do not "know" where they originated in the document once you move them. Our absolute div can use some assistance understanding what you meant by top.

The browser will look 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 browser positions the absolute div in relation to the **<html>** element—the document itself. This is why our div is 20 pixels from the top of the browser window.

1. Return to your code editor.

2. Whenever you declare an **absolute** position, you should think "I absolutely need to determine where this element gets its coordinates." Switch to **position.html** in your code editor so we can find a suitable parent element to use as a positioning anchor.

3. Around lines 22–29, locate the markup for the segment of the page we are working with, as shown below:

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

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

4. Switch to **main.css**.

5. In the rule for **.parent** that starts around line 19, declare a **relative** position as shown below in bold:

```
.parent {
   background: #eee;
   border: 1px solid #999;
   height: 250px;
   margin-top: 40px;
   padding: 20px;
   position: relative;
}
```

   NOTE: We could have used either absolute or relative positioning. Relatively-positioning the parent is the natural choice because you don't need to then tell the parent where to get its coordinates from, and it will stay in its natural place in the document with no extra work needed.

6. Save **main.css** and reload **position.html** in Chrome. Ah! Now the absolute div is 40 pixels from the top of its parent container.

7. Let's experiment some more. Switch back to **main.css** in your code editor.

8. Change the .absolute-div's **top** coordinate as shown in bold below:

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

9. Save **main.css** and reload **position.html** in Chrome. Now the absolute div's top edge is flush with its parent's top edge.

10. Feel free to use the DevTools to add or modify any of the **.absolute-div**'s coordinate properties. For instance, you could add a **left** property.

Position: Fixed

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 next 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 that starts around line 41 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?
5. Unless you have a tall enough display, the fixed div is nowhere to be seen.

   It may appear as if the element has gotten sucked into the void, but it's there. We just need to tell our fixed div where it needs to be in relation to the viewport in order to get it to show up.
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!
9. Scroll through the page to see that the fixed div never budges. It's clamped onto that browser window for dear life, impervious to any scrolling.

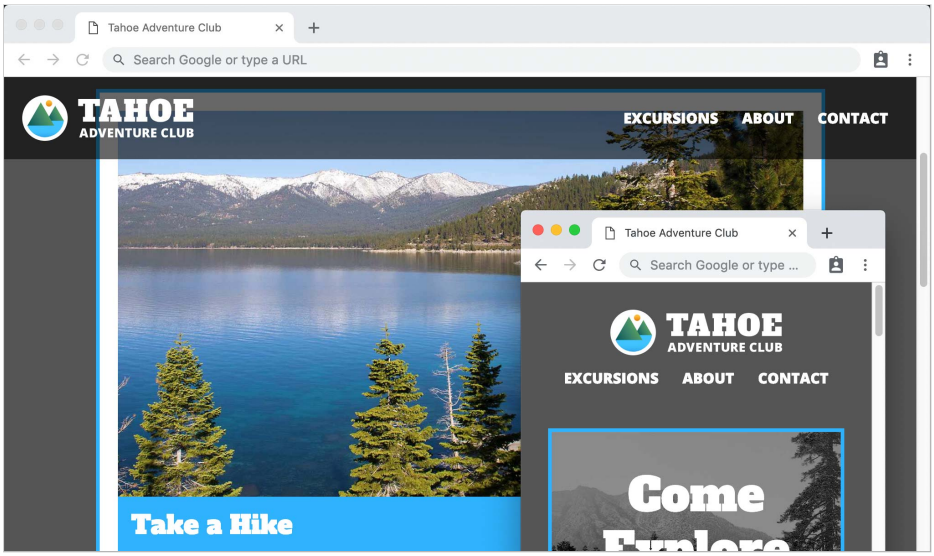
   The element has also collapsed down to the minimum size required to hold the "fixed" text. Let's expand the element so it stretches across the entire browser window, mimicking a common feature on modern websites: a header that stays put as you scroll.

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

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

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

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. In your code editor, close any files you may have open.
2. For this exercise we'll be working with the **Tahoe Fixed Navbar** folder located in **Desktop > Class File > 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 Fixed Navbar** folder.
4. 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 device menu above the webpage, select **iPhone 6/7/8**.
   - Click the **Reload** button, or hit **Cmd–R** (Mac) or **Ctrl–R** (Windows).

5. 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 open in the browser with the DevTools open, 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. Open **main.css** from the **css** folder (in the **Tahoe Fixed Navbar** folder).
2. Below the **h2 + p** rule, add the following new rule:

```
nav ul {
   list-style-type: none;
   margin: 0;
   padding: 0;
}
```

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 the vertical stack we have now.

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

```
nav {
   text-align: center;
}
```

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 to be styled.

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

11. Below the rule for **nav li** add the following new rule:

```
nav ul a {
   color: #fff;
   font-weight: 800;
   text-transform: uppercase;
   padding: 11px;
   display: inline-block;
}
```

NOTE: We have **`display: inline-block;`** so the padding will work the way we want it to.

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

Creating a Fixed Navbar 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 for screens 600 pixels or wider.

1. In the upper left of the DevTools panel click the **Toggle device toolbar** button to leave device mode.
2. Close the Devtools.
3. Make the window fairly wide if it's not already.
4. Switch back to **main.css** in your code editor.
5. At the bottom of the file, below all the other rules add the following:

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

6. Save **main.css** and reload **index.html** in the browser.
7. Scroll through the page to see that the fixed header stays at the top of the page. However, notice the following issues we need to solve:
   - The fixed header has collapsed down to the minimum size required to hold the content, so it no longer stretches across the page like we want.
   - The **Come Explore 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.

8. We'll solve the width issue by adding some coordinates. While we're at it, let's also add a dark background so it will be easier to see the navbar's content as it scrolls over the page's content.

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

9. Add the following bold code to the fixed **header**:

```
@media (min-width: 600px) {
   nav {
      position: fixed;
      background: rgba(0,0,0, .6);
      top: 0;
      left: 0;
      right: 0;
   }
}
```

   The hex colors we've been using are not transparent. A newer way to specify color is **RGBA** (Red Green Blue Alpha). RGBA 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. Any other number combination equals a different color. **A** stands for **Alpha**, which specifies how transparent the color is. Alpha values range from 0 (100% transparent) to 1 (100% opaque). A decimal value is something in between (.6 is 60%).

   While hex values were eventually updated to support alpha (using 4 and 8 digit values instead of the standard 3 and 6), browser support is not as good as RGBA.

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

    - The navbar should now have a transparent darker background and be the full width of the page.
    - Scroll down/up and notice how you can see the content through the semi-transparent navbar.
    - The **Come Explore Tahoe!** section (at the top of the page) is being 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, so it will be under the navbar when the page first loads.

11. 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 {
```

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

    - The **Come Explore Tahoe!** section should no longer appear behind the navbar when the page first loads.
    - 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!

Refining the Navbar Appearance on Larger Screens

On larger screens, the navbar is too tall with the logo above the links. Let's move the logo to the left and the links to the right.

That means we don't want the text aligned centered as it is on small screens. One way to fix this is to add a **`text-align: left;`** in the media query for larger screens. But that means we have to set text-align for both screen sizes. Instead, we add a new media query and make the **`text-align: center;`** only apply to smaller screens.

1. Find the **nav** rule that sets **`text-align: center;`** It's not the one in the media query, it's higher up in the code.
2. Select the entire **nav** rule and **cut** it. Be sure to cut it (not copy) so it will be removed from its current location.
3. Above the **min-width: 600px** media query, add a new **max-width** shown below in bold (note that it's **max** and not **min**):

```
@media (max-width: 599px) {

}
@media (min-width: 600px) {
```

NOTE: Notice the 599px media query is one pixel less than the 600px media query? If they were both 600px, then both media queries would apply for screens that are exactly 600 pixels wide. Making them one pixel different ensures that only one set of styles will be applied!

4. Inside the new media query paste the nav rule so you end up with the following:

```
@media (max-width: 599px) {
   nav {
      text-align: center;
   }
}
@media (min-width: 600px) {
```

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

   The nav should be left aligned on large screens, and centered on small screens.

6. Switch back to **main.css** in your code editor.
7. Now we can move the links to the right. In the **min-width: 600px** media query, below the **nav** rule, add the following bold code:

```
@media (min-width: 600px) {
   body {
      margin-top: 130px;
   }
   nav {
      CODE OMITTED TO SAVE SPACE
   }
   nav ul {
      float: right;
   }
}
```

8. Save **main.css** and reload **index.html** in the browser.
   - Make sure the window is large, so you see the fixed navbar.
   - The links should be on the right, with the logo on the left.
   - The navbar needs some padding so the logo and links don't hit the edge.
9. Return to your code editor.
10. In the **min-width: 600px** media query, below the **nav** rule add the following bold code:

```
nav {
   position: fixed;
   background: rgba(0,0,0, .6);
   top: 0;
   left: 0;
   right: 0;
   padding: 20px;
}
```

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

    The navbar should now look good at any screen size!

Optional Bonus: Add a Hover State to the Nav Links

Let's add a hover to the links in the navbar. While mobile users won't see the hover effect, desktop users can.

1. Return to your code editor.

2. Below the **nav ul a** rule (higher up in the code, not in a media query), add the following code:

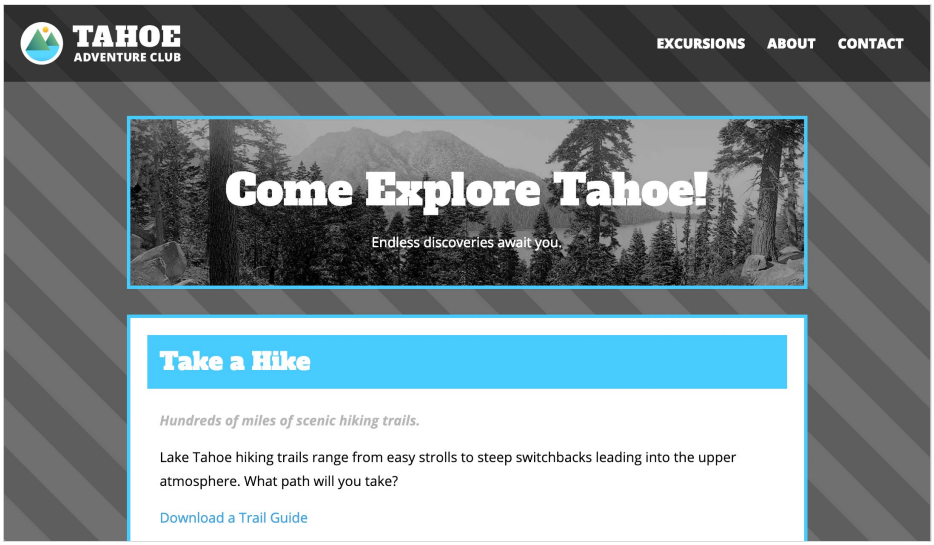
```
nav ul a:hover,
nav ul a:focus {
   background: #2fb3fe;
}
```

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!

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

- Hover over the links to see a blue background appears.
- 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.

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 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 (in the **Tahoe Gradients** 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, yellow, green, blue, violet);
   margin: 30px;
   font-family: 'Open Sans', sans-serif;
   line-height: 1.75;
}
```

   NOTE: Really old browsers that don't support gradients can fall back to the solid hex color. The main reason we are using this solid gray background color though, is 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, very trippy! Let's tone that down. We can also try a linear, rather than radial gradient.
4. Return to **main.css**.
5. Edit the gradient style and trim down the rainbow as follows:

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

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, yellow);
```

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, yellow);
```

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, yellow);
```

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, yellow);
   ```

3. Then set yellow 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, yellow 80px, yellow 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 yellow). 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, yellow 80px, yellow 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, yellow 80px, yellow 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 yellow stripe to pare things down.

9. Return to **main.css** and **delete** the yellow 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 the orange value 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;
   margin: 30px;
   font-family: 'Open Sans', sans-serif;
   line-height: 1.75;
}
```

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).

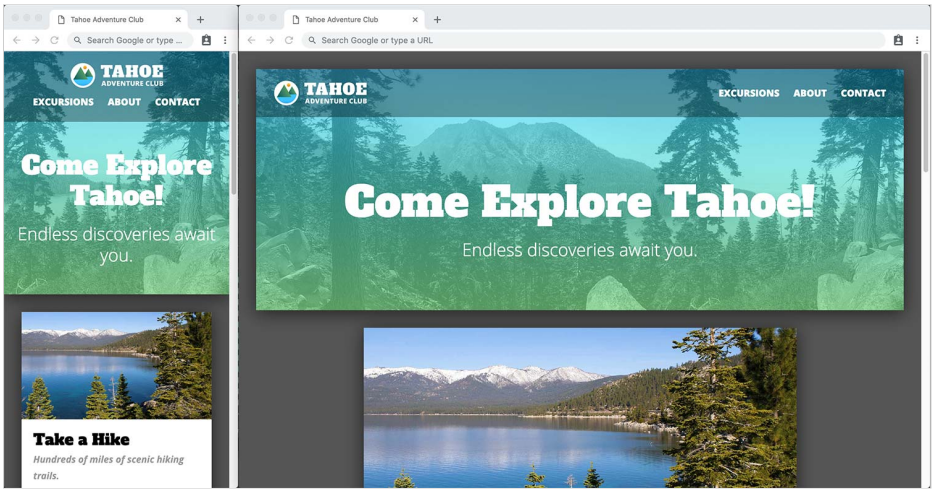
15. Return to **main.css** and try one more option for thinner stripes::

```
background-image: repeating-linear-gradient(45deg, transparent, transparent
4px, rgba(0,0,0, .2) 4px, rgba(0,0,0, .2) 8px);
```

16. Save the file and reload the browser. Feel free to experiment and create different gradients and stripe patterns.

    You can combine multiple CSS backgrounds (as you'll learn about in the next exercise) and create many different effects. By combining multiple gradients you can create all sorts of interesting patterns. Check out **leaverou.github.io/css3patterns** for some inspiration!

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 Multiple Backgrounds** 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 Multiple Backgrounds** 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 (in the **Tahoe Multiple Backgrounds** folder).

6. Find the **header** rule and 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.

7. 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!

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-bottom: 20px;
   CODE OMITTED TO SAVE SPACE
}
```

   NOTE: How does this work? 11vw is like 11% of the viewport width. On a 320px screen, 320 x .11 = 35.2px type. On a 1200px screen, 1200 x .11 = 132px 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.

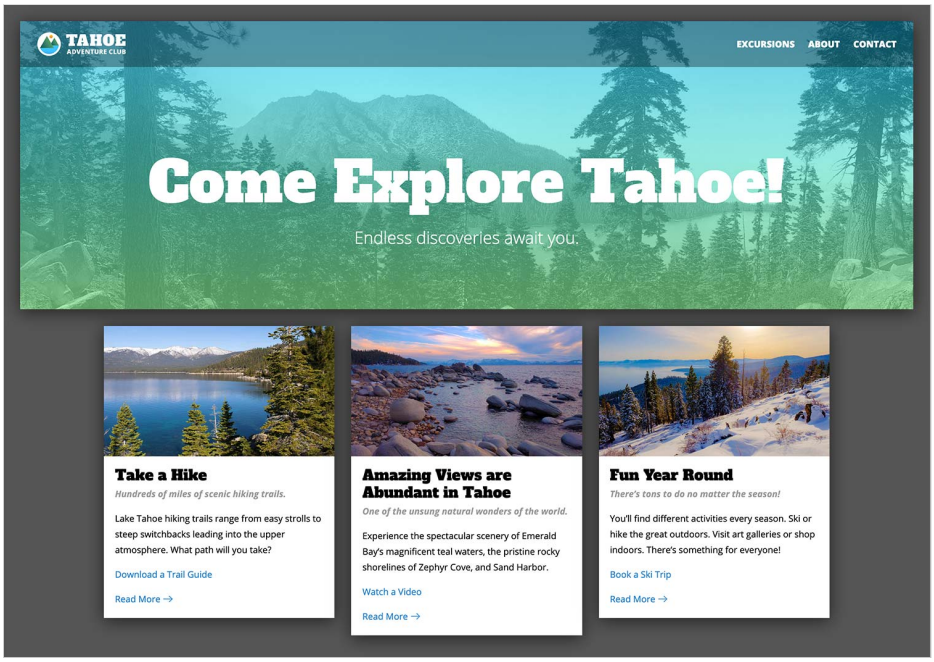
   - Resize the window from mobile through desktop size to check out the type as the size changes.
   - The type size is good on small and medium screens, but is too big on wide screens.

10. Return to your code editor.
11. At the bottom of the file, inside the **min-width: 1100px** media query add a new rule for the **h1** as shown below in bold:

```
@media (min-width: 1100px) {
   h1 {
      font-size: 6vw;
   }
   body {
```

12. Save the file.
13. 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 inline-block. Along the way you'll learn about some new CSS selectors (:not and :last-child) and how you can use CSS to calculate numeric values that involve different types of measurements (such as percentages and pixels).

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 **Tahoe 3 Columns** 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 3 Columns** folder.
4. Preview **index.html** in a browser.

   Below the header photo are 3 sections. On wide screens these can be columns so users don't have to scroll as much to see the content.
5. Leave the page open in the browser so we can come back to it and reload as we make changes.

Creating a 3-Column Layout Using Inline-Block

1. Switch back to your code editor and open **main.css** from the **css** folder (in the **Tahoe 3 Columns** folder).

2. In the **min-width: 950px** media query add the following code shown below in bold:

```
section {
   width: 33.33%;
   margin: 30px 0;
}
```

   NOTE: We want the three columns to evenly fill the area, so we're making each column 33.33% wide. In this rule we removed the left/right margins added by a previous rule. In a moment we'll be adding some margin back to add space between the columns, but we'll need to do so in a very specific way.

   How did we choose 950px for the media query? We initially guessed an amount, tested it, and then adjusted until we decided on 950. This amount is based on how the content fits, not on any specific device size.

3. Save the file, then reload the page in your browser. The 3 sections are narrow, but we need to get them side-by-side.

4. Go back to **main.css** and add:

```
section {
   width: 33.33%;
   display: inline-block;
   margin: 30px 0;
}
```

5. Save the file and reload the page in your browser. When the window is wide, only two of the sections will be next to each other, so why aren't all three columns fitting as we'd expect?

6. Notice there's a small space between the 2 columns that are next to each other. Think of each of the column like a word, and that gap is like a space character between words. In the HTML code, our **section** tags are on different lines, and that's creating the extra space. With the extra space there's not enough room for the 3rd column to fit so it wraps to the next line. There are multiple ways we can fix this:

   - Put the **section** tags on the same line of code with no space between them. We're not going to do this because it makes the code a bit harder to read.
   - Use an HTML comment to tell the browser to ignore the space between the **section** tags. This is what we'll do, because we think it makes the code easier to read. HTML comments are written like this:
     `<!-- anything in here is ignored -->`

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

8. Add two HTML comments (after the first and second **section** tags) as shown below in bold. Be careful to open and close each comment.

```
<main>
   <section>
      CODE OMITTED TO SAVE SPACE
   </section><!--
   --><section>
      CODE OMITTED TO SAVE SPACE
   </section><!--
   --><section>
      CODE OMITTED TO SAVE SPACE
   </section>
</main>
```

9. Save the file, then reload the page in your browser. Now the columns should all fit!

Aligning Sections to the Top

The columns are displaying side-by-side but the alignment is not what we want. With inline-block, elements behave like text, aligning on the baseline (bottom) by default. Let's align their tops instead.

1. Return to **main.css** in the code editor and, in the rule for **section**, set the following:

```
section {
   width: 33.33%;
   display: inline-block;
   vertical-align: top;
   margin: 30px 0;
}
```

2. Save the file, then return to the browser and reload the page. It looks better with all the sections aligning at the top, but the columns could use some space between them. We'll make it exactly 30px to match the space above them.

Using CSS calc()

We want to add margin to the right of the all the sections, except for the last column. If the last column had extra space on its right, the design would not be properly centered. We could add right margin to all the columns, and then remove it on the last, but there's a way to target all of them… except the last!

1. Return to **main.css** in your code editor and in the **min-width: 950px** media query, below the **section** rule, add the following new rule shown below in bold:

```
section {
   width: 33.33%;
   display: inline-block;
   vertical-align: top;
   margin: 30px 0;
}
section:not(:last-child) {
   margin-right: 30px;
}
```

   This selector means: any **section** that is **not** the **last element** (within the parent container). We'll cover more of these types of advanced selectors in a later exercise.

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

   Again we see the columns don't all fit on the same line, because we added space. Our columns can't be 33.33% wide plus the space. We need to subtract 60px from the total space (2 columns have 30px of margin on their right). H ow do we subtract a fixed pixel value from a percent? We let the browser do it using a calc() function!

3. Return to **main.css** in your code editor and in the **section** rule replace the 33.33% width as shown below in bold:

```
section {
   width: calc();
   display: inline-block;
   vertical-align: top;
   margin: 30px 0;
}
```

4. Add the following math, paying close attention to spaces!

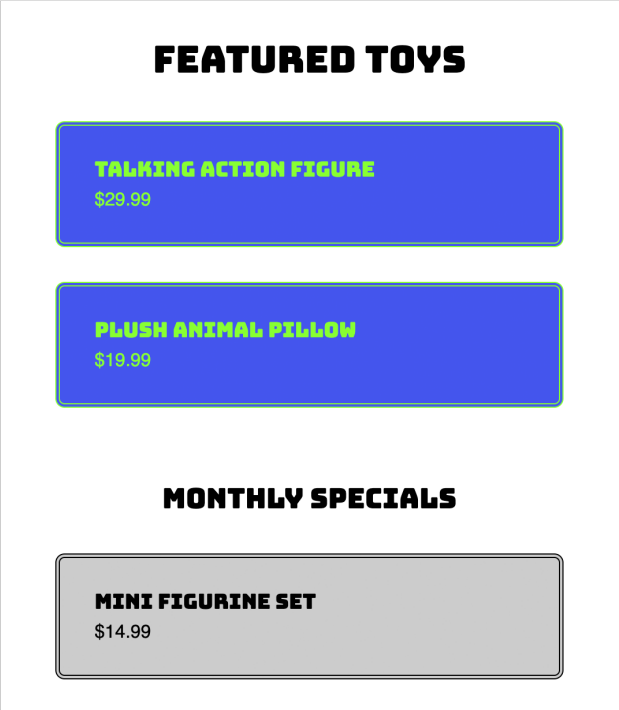
```
section {
   width: calc( (100% - 60px)/3 );
   display: inline-block;
   vertical-align: top;
   margin: 30px 0;
}
```

   So how does this math work? 100% of the space, minus the 60px of margin, divided by the number of columns (3) gives us the exact size we need for each column!

   NOTE: For addition and subtraction, the + and - operators must always be surrounded by whitespace. For multiplication and division, the * and / operators do not need whitespace, but adding it for consistency is allowed, and recommended.

5. Save the file, then reload the page in your browser. The columns should now fit!

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 (we'll be using its DevTools).

   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 so you can come back to it and reload as you make changes.

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 {
      --primary-font: 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**

   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 **primary-font** 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(--primary-font);
   font-weight: normal;
```

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

```
.product-name {
   font-family: var(--primary-font);
   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!

Updating the Variable

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

```
:root {
   --primary-font: 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 {
   --primary-font: Coiny, sans-serif;
   --standard-spacing: 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(--standard-spacing);
   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(--standard-spacing) * 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(--standard-spacing);
   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(--standard-spacing);
   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(--standard-spacing) - 5px ) var(--standard-spacing);
   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.

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 **--standard-spacing** 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 **–standard-spacing** value to **30px**:

```
:root {
   --primary-font: Coiny, sans-serif;
   --standard-spacing: 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 {
   --primary-font: Bungee, sans-serif;
   --standard-spacing: 30px;
   --card-bgcolor: #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-bgcolor);
   CODE OMITTED TO SAVE SPACE
}
```

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

```
:root {
   --primary-font: Bungee, sans-serif;
   --standard-spacing: 30px;
   --card-bgcolor: #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-bgcolor);
   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-bgcolor);
   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 {
   --primary-font: Bungee, sans-serif;
   --standard-spacing: 30px;
   --card-bgcolor: #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.

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-bgcolor: #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).

Browser Support

Support for CSS variables is very good in modern browsers, but you should know that older browsers (like Internet Explorer) do not support them. In many cases, variables are not simply progressive enhancement and may degrade poorly in older browsers, so make sure you don't need to support those older browsers before choosing to use variables. You can see more about browser support at caniuse.com/#feat=css-variables

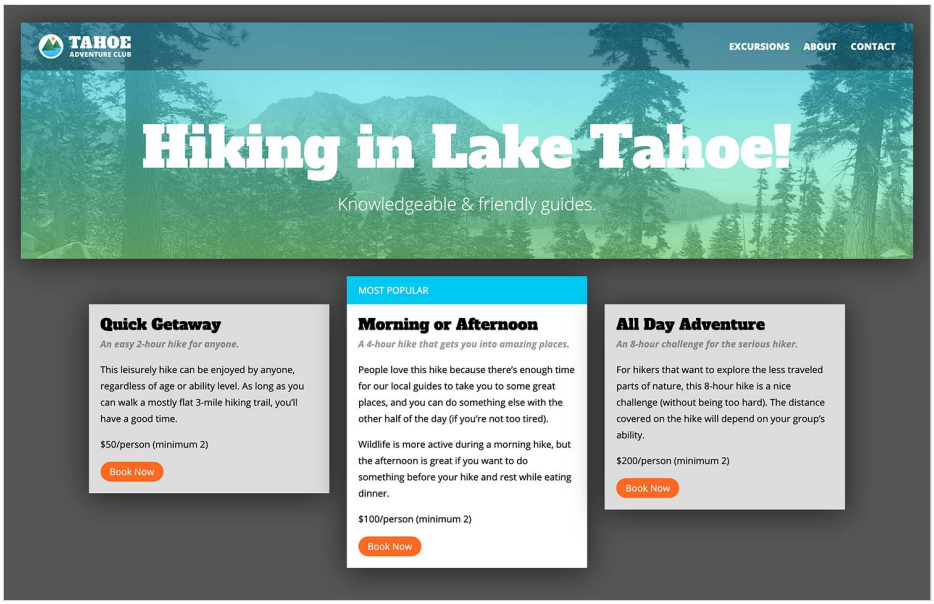
CSS Variables, Preprocessors, & JavaScript

One of the powerful things about CSS variables (especially when compared to preprocessor variables like Sass) is that they are accessible via JavaScript. JavaScript can change CSS variables at any time (or based on a user's interaction), which then updates the appearance of a page.

For example, imagine allowing users to change the font. JavaScript could update our CSS font variable and all the places using it will change. Powerful!

To learn more, read Smashing Magazine's **Strategy Guide To CSS Custom Properties** tinyurl.com/css-var-strategy

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 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 **Tahoe Relational Selectors** 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 Relational Selectors** folder.
4. Preview **index.html** in Chrome (we'll be using its DevTools).

   While this page is similar to the one you've worked on in previous exercises, there are some minor differences. A lot of styling has been done, but we're going to add some final touches.
5. Leave the page open in Chrome.

Adjacent Selectors

1. Return to **index.html** in your code editor.
2. In the first **section** tag, find the first paragraph below the **h2**:

   ```
   <p>Hundreds of miles of scenic hiking trails.</p>
   ```

   Let's make this first paragraph after each section's h2 stand out from the others. We can target it with an **adjacent selector**, which targets a specific element only when it immediately follows another specific element. The elements must be siblings (next to each other, not nested), which is why this selector is sometimes referred to as an adjacent sibling selector.

3. Open **main.css** from the **css** folder (in the **Tahoe Relational Selectors** folder).
4. Below the **h2** rule (which is near the top of the file), add this new rule:

   ```
   h2 + p {
      font-weight: 700;
      font-style: italic;
      opacity: .4;
      margin-top: 3px;
   }
   ```

   The **h2 + p** selector targets a paragraph that comes directly after an h2. Sometimes with CSS selectors, especially adjacent selectors, it may be easier to read them right to left.

5. Save the file.
6. Return to Chrome, reload the page, and:

   - In each of the 3 columns, the paragraph below the heading should now be italic, bold, gray, and closer to the heading above. It's nice that we could do this without creating a new class name.
   - At the bottom of each column are links. It's hard to see that there are actually two different links, so let's make the first link (Read More) stand out.

Using first-child & last-child

1. Return to your code editor.
2. Switch to **index.html**.
3. At the bottom of any of the three **section** tags, notice the last paragraph has a **links** class and contains the two links we just saw in the browser.
4. Switch to **main.css**.

5. Below the **section .text-wrapper** rule, add the following new rule:

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

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

   - Notice the **Read 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.

7. Return to your code editor.

8. 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:fist-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.

9. 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.

10. In Chrome, at the bottom of the first column find the **Download a Trail Guide** link.

11. **Ctrl–click** (Mac) or **Right–click** (Windows) on the **Download a Trail Guide** link and choose **Inspect**.

12. In the DevTools **Style** 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.

13. 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, consider using a class instead.
12. Reload the page.

Using last-of-type

All the links in the nav have right margin, which adds a little extra space to the right of the nav (when compared with the space to the left of the logo). Let's remove that extra space.

1. Return to your code editor.
2. In **min-width: 600px** media query, below the **nav li** rule add the following new rule:

```
nav li:last-of-type {
   margin-right: 0;
}
```

3. Save the file, and reload the page in Chrome. The space to the right of the nav should be less, which matches the space to the left of the logo.

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** from the **Tahoe Relational Selectors** folder.
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.

6. Before we target the second column, let's perform a test of the nth-child selector. In **min-width: 950px** media query, below the **section:not(:last-child)** 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, grand-children, 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.

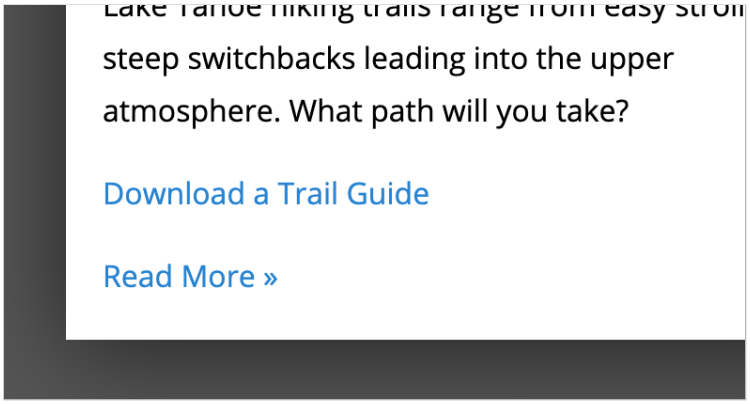
14. Return to **main.css** in your code editor.

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: 78px;
}
```

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.

Exercise Preview



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. 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 **Tahoe Pseudo-Elements** 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 Pseudo-Elements** folder.
4. Preview **index.html** in Chrome (we'll be using its DevTools).

   This page is similar to the previous exercise, but notice the links at the bottom of each column are on their own line. This helps them to stand out, and gives us space to add some cool styling. We want to add a special character to all three of the **Read More** links, which we can do with CSS!
5. Leave the page open in Chrome.

Pseudo-Elements & the Content Property

1. First let's figure out how to target these links. Go to **index.html** in your code editor.
2. Find the **Read More** link at the bottom of the first **section**.
3. Notice it has a **read-more** class we can use to style it.

4. Open **main.css** from the **css** folder (in the **Tahoe Pseudo-Elements** folder).
5. Above the **.book-now-btn** rule add the following new rule

```
.read-more::before {

}
```

The ::before creates a **pseudo-element**, which is a virtual (fake) 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.

6. 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).

```
.read-more::before {
   content: "» ";
}
```

7. Save the file and reload the page in Chrome.
8. Check out the **Read More** links to see they now have the special » character added at the beginning of the link.
9. Return to your code editor.
10. Change **before** to **after**:

```
.read-more::after {
   content: "» ";
}
```

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

```
.read-more::after {
   content: " »";
}
```

12. Save the file and reload the page in Chrome. Now the » character should be at the end of the link.

Seeing Pseudo-Elements in Chrome's DevTools

1. **Ctrl–click** (Mac) or **Right–click** (Windows) on a **Read More** link's » character and choose **Inspect**.
2. In the DevTools **Elements** panel you'll see the HTML code. Expand the **<a href="#" class="read-more">** element to see the **::after** inside.
3. Click on the **::after** and you'll see the style that creates it listed in the **Styles** panel.

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:

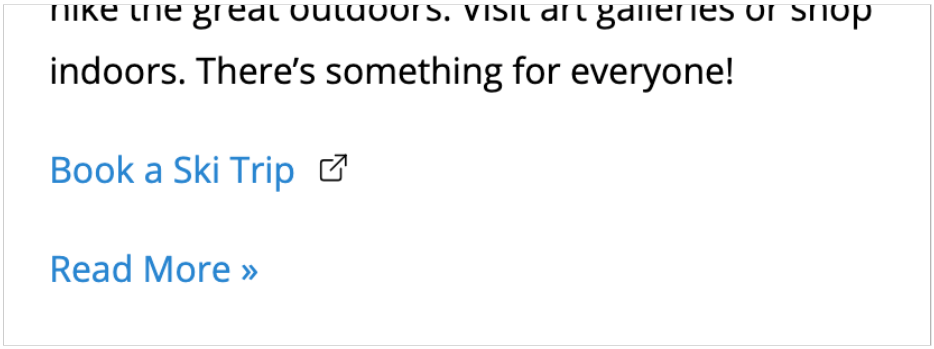
- Styles an element when it's in a certain state.
- Preceded by one colon.
- Examples are :hover, :first-of-type, :last-child, :not, :checked

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



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. 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 **Tahoe Attribute Selectors** 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 Attribute Selectors** folder.
4. Preview **index.html** in a browser. This is where we left off in the previous exercise. 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.
5. 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 **Read 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 **.book-now-btn** 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.

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**).

4. Save the file and reload the page in the browser. Look at the **Book a Ski Trip** link to see that the external links icon appears 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:

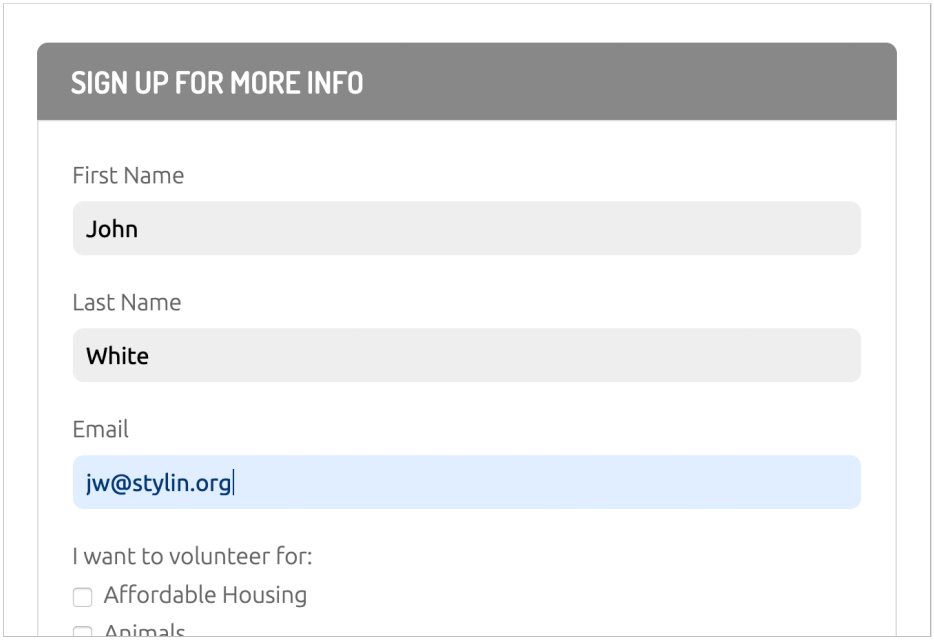
```
a[href^="http"] {
   background: url(../img/icon-external-link.svg) no-repeat right center /
16px 16px;
   padding-right: 24px;
}
a[href*="youtube.com"] {
   background: url(../img/icon-youtube.svg) no-repeat right center / 16px 16px;
   padding-right: 24px;
}
```

   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).

Exercise Preview



Exercise Overview

In this exercise, you'll learn new things about styling forms as well as practice techniques you've been learning in previous exercises.

This exercise will focus on styling the front-end of the form, not programming the back-end functionality. Creating a functioning form requires knowledge of a back-end programming language such as PHP, Ruby on Rails, Node.js, etc.

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 Forms** 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 Forms** folder.
4. Preview **volunteer.html** in a browser.

   This page has some standard form elements, but they look bad because we have not applied any styling yet.
5. Leave the page open in the browser.

Looking Over the Form Content

1. Return to **volunteer.html** in your code editor.
2. Find **`<form action="" class="volunteer-form">`** and take a moment to look over the form code inside it, noticing the following:
   - Each input has a label, which is connected by its ID.
   - There are different types of inputs such as text, email, checkbox, and submit.
   - The fieldset groups together related elements, in this case a set of checkboxes.
   - In addition to inputs, there is also a textarea (which is a multi-line input).
3. Just after the start of the **body** tag, there's a **nav** which contains a **form** that has one search input.

Styling the Labels & Inputs

Let's put the text labels and text fields onto their own lines.

1. Open **main.css** from the **css** folder (in the **Helping Hands Forms** folder).
2. Below the **.form-heading** rule, add a rule for labels in the form:

```
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.

3. Save the file and reload the page in your browser. Notice the text labels are now on their own line, with the text fields below them. Now let's improve the text fields.
4. Return to **main.css** in your code editor.
5. 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.

6. Save the file, reload the page in your browser, and notice:
   - The **First Name** and **Last Name** text fields are the full width of the column.
   - The **Email** text field is not the full width like the other fields, because its type is email, not text like our CSS rule targeted.
   - The textarea below **Anything else you'd like to add?** is not on its own line because we didn't target textareas.
7. Return to your code editor.
8. Edit the input selector as shown in bold, making sure you don't miss the commas!

```
input[type="text"],
input[type="email"],
textarea {
   display: block;
   width: 100%;
}
```

9. Save the file and reload the page in your browser. The email field and textarea should both be 100% wide and on their own lines.
10. Return to your code editor.
11. Add more styling to make them look better:

```
input[type="text"],
input[type="email"],
textarea {
   display: block;
   width: 100%;
   background: #eee;
   border: none;
   border-radius: 8px;
   padding: 10px;
   margin-bottom: 20px;
}
```

12. Save the file, reload the page in your browser, and:
    - Notice the text fields should have a gray background, no border, and rounded corners which better matches the rest of the page's design.
    - There's too much space above **First Name**, so let's fix that.
13. Return to your code editor.

14. Below the **form > label** rule, add this new rule:

```
form > label:first-of-type {
   margin-top: 0;
}
```

15. Save the file, reload the page in your browser, and:
    - Notice the space above **First 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.
16. Return to your code editor.
17. Let's style the textarea. Above the **aside section** rule, add the following new rule:

```
textarea {
   min-height: 120px;
   resize: vertical;
}
```

18. Save the file, reload the page in your browser, 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.
19. Return to your code editor.
20. Below the **textarea** rule, add the following new rule:

```
input[type="text"],
input[type="email"],
input[type="submit"],
input[type="search"],
textarea {
   outline: none;
}
```

21. Save the file and reload the page in your browser.
22. 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.

1. Return to your code editor.
2. Above the **aside section** rule, add the following new rule:

```
input:focus,
textarea:focus {
   background: #e1efff;
   color: #02387a;
}
```

   NOTE: This rule targets the inputs and textarea when they are focused on (selected by a mouse or keyboard).

3. Save the file, reload the page in your browser, 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. Return to your code editor.
2. Above the **aside section** rule, add the following new rule:

```
fieldset {
   border: 0;
   margin: 0;
   padding: 0;
}
```

3. 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.
4. Return to your code editor.
5. Our form contains a list (**ul** tag) with all the checkboxes. We gave that a class of **interests-list** so we can style it. Below the **fieldset** rule, add the following new rule:

```
.interests-list {
    list-style-type: none;
    margin: 0;
    padding: 0;
}
```

6. Save the file and reload the page in the browser. It's getting close, but we could use more space between the checkboxes and the text label to the right.

7. Return to your code editor.
8. Below the **.interests-list** rule, add the following new rule:

```
input[type="checkbox"] {
   margin-right: 5px;
}
```

9. Save the file 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. Return to your code editor.
2. The **Sign Me Up** button is an input with **type="submit"** which we can use to target it. Below the **input[type="checkbox"]** rule, add the following new rule:

```
input[type="submit"] {
   font-family: Dosis, sans-serif;
   font-weight: 600;
   font-size: 20px;
   text-transform: uppercase;
   background: #6bb359;
   color: #fff;
   padding: 10px 20px;
   border: none;
   border-radius: 8px;
}
```

3. Save the file and reload the page in the browser. Much improved! Let's make the button change color on hover.
4. Return to your code editor.
5. Below the **input[type="submit"]** rule, add the following new rule:

```
input[type="submit"]:hover, input[type="submit"]:focus {
   background: #449d44;
}
```

6. Save the file, reload the page in your browser, and:
   - Hover over the **Sign Me Up** button to see it darken.
   - In the navbar at the top right of the page notice there's an unstyled search field. Let's make that look better.

Styling the Search Field

1. Return to your code editor.
2. Above the **aside section** rule, add the following new rule:

```
nav input[type="search"] {
   width: 160px;
   padding: 6px 15px;
   border-radius: 50px;
   border: 1px solid #a4c1e6;
   -webkit-appearance: none;
}
```

   NOTE: Setting **-webkit-appearance** to **none** removes an inner shadow on iOS.

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

   That looks better. Now let's style the **Search** placeholder text.

4. Return to your code editor.
5. Below the **nav input[type="search"]** rule, add the following new rule:

```
nav input[type="search"]::placeholder {
   color: #4197ff;
   opacity: .5;
   line-height: normal;
}
```

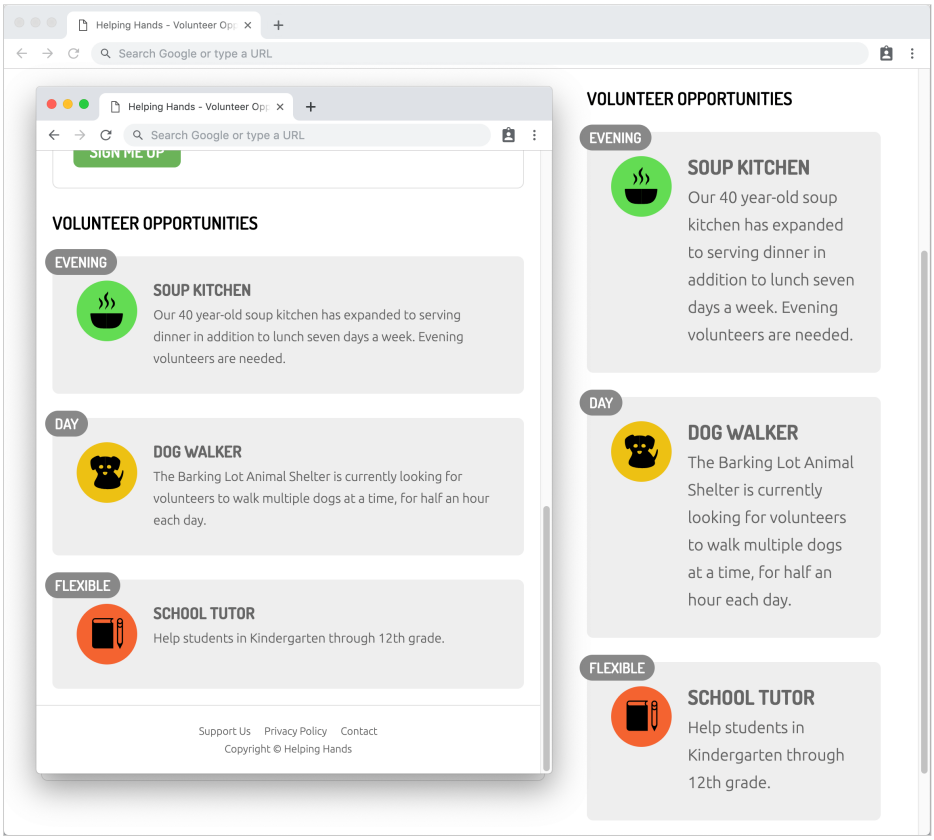
   NOTE: Unlike other browsers, Firefox reduces the opacity of placeholder text. If you don't want opacity, set it to 1 to make Firefox consistent with other browsers. The line-height fixes the vertical alignment on iOS.

6. Save the file and reload the page in the browser. The **Search** placeholder text should now be blue (it was dark gray).

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.

Exercise Preview



Exercise Overview

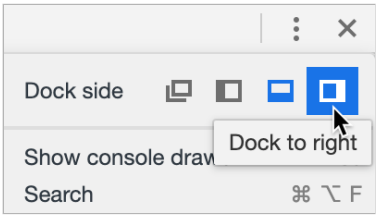
In this exercise, you will learn the difference between a fixed size (such as pixels) and relative sizes (such as ems and rems).

Em Units

1. 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 **Helping Hands Em vs Rem** 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 Em vs Rem** folder.
4. Preview **volunteer.html** in Chrome (we'll be using its DevTools).

   This page is similar to the previous exercise, but we've made a few adjustments for teaching this topic.

5. In Chrome, **Ctrl–click** (Mac) or **Right–click** (Windows) on the **Sign Me Up** button and choose **Inspect**.
6. 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:



7. In the **Styles** panel, the top style should be **input[type="submit"]**
8. In that style, click on the **20px** value for **font-size**.
9. 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.
10. 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.

11. Return to your code editor.
12. Open **main.css** from the **css** folder (in the **Helping Hands Em vs Rem** folder).

13. 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.

14. Save the file 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!

15. 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**.

16. Click on the **20px** value for **font-size**.
17. 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!
18. In the DevTools, if you see a **Computed** tab, click on it. Otherwise look at the panel (to the right of Styles) with a colored diagram of margin, border, and padding.
19. 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**
20. Here you can see how the padding's em units have been translated into pixels.
21. Hit the **Esc** key to remove the filter.
22. If needed, 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. Return to your code editor.

2. Near the top of the file, find the **h3** rule and change font-size to **1.25em**:

```
h3 {
   color: currentColor;
   font-size: 1.25em;
   margin: 0 0 7px;
}
```

3. Scroll down, and change the font-size of the **aside section p** rule:

```
aside section p {
   font-size: 1em;
   margin: 0;
   overflow: hidden;
}
```

4. 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;
   background: #eee;
   border-radius: 8px;
   margin-top: 30px;
   padding: 30px;
   position: relative;
}
```

   Within the aside's section tags, 1em will now equal 14px.

5. Save the file 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**

9. Click on the **14px** value for **font-size**.

10. 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 using using a **px** font-size.

    This is the strength of ems: Making one size change can affect multiple elements. Additionally, those relative sizes could help us across media queries.

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 **r**oot **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. Return to your code editor.
2. In the **aside section p** rule, add an "r" to change "em" to "rem":

```
aside section p {
   font-size: 1rem;
   margin: 0;
   overflow: hidden;
}
```

3. In the **h3** rule, add an "r" to change "em" to "rem":

```
h3 {
   color: currentColor;
   font-size: 1.25rem;
   margin: 0 0 7px;
}
```

4. Let's define a root font-size. We can set this on the **html** or **:root** element. At top of the file, just above the **body** rule, add this new rule:

```
:root {
   font-size: 1rem;
}
```

   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.

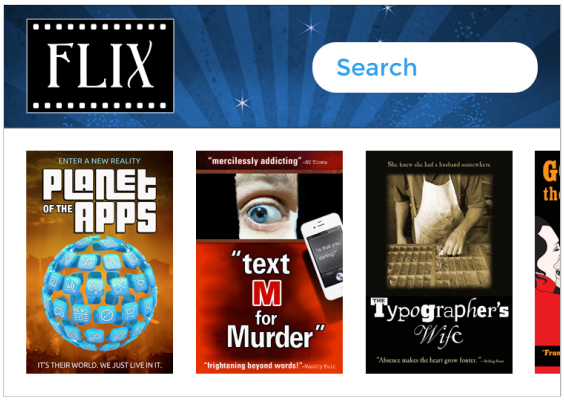
5. Towards the bottom of the file, in the media query for **min-width: 930px** add this new rule:

```
@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).

6. Save the file and reload the page in Chrome.
7. Resize the browser window to see the size of the heading and paragraph text in the aside is bigger on desktop and and smaller on mobile.

Exercise Preview



Photos courtesy of istockphoto, unizyne, Image #19302441, Marcello Bortolino, Image #17472015, Sergey Kashkin, Image #318828, Sveta Demidoff, Image #2712135.

Exercise Overview

In this exercise you'll learn how to make one section of a page scrollable. You'll even optimize it to add the "native" scroll bounce people expect on iOS devices.

Styling the Horizontal Scroll Area

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 **Flix Scrollable Area** 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 **Flix Scrollable Area** folder.
4. Preview **index.html** in a browser.

   For now we've only started the mobile version of this page, so make your window narrow like a mobile device.

   Notice the movie posters are currently stacked on top of each other. Our design calls for the movie posters to be on a single scrollable line. On smaller devices, we want users to be able to scroll from left to right if all the movie posters don't fit on-screen. First let's get the movie posters in a single line.
5. Return to your code editor.
6. Open **main.css** from the **css** folder (in the **Flix Scrollable Area** folder).

7. The movie posters are in an unordered list that has a **movies** class applied to it. Find the **.movies li** rule. This targets the list items in our **movies** unordered list and sets their size and spacing.

8. Add the following bold code to get all the movie posters on one line:

```
.movies li {
    display: inline-block;
    width: 99px;
    margin-right: 11px;
}
```

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

   The posters are now on the same line, but make the browser window narrow and notice that the movie posters wrap to a second line. On smaller devices we want to prevent this behavior.

10. Switch back to your code editor.

11. Directly above the **.movies li** rule, find the **.movies** rule. This targets our movies list. Add the following bold code:

```
.movies {
   background-color: #fff;
   padding: 15px;
   margin: 0;
   white-space: nowrap;
}
```

12. Save the file, reload the page in your browser, and:

    - Make the browser window narrow so you can see that the movie posters no longer wrap to a second line.
    - There's a new problem though. Now that the posters don't fit, scroll the page to the right to see it looks weird because the posters are wider than everything else. The problem is the whole page gets scrolled, but we only want the posters section to scroll.

13. Switch back to your code editor.

14. Add the following bold code to the **.movies** rule:

```
.movies {
   background-color: #fff;
   padding: 15px;
   margin: 0;
   white-space: nowrap;
   overflow-x: auto;
}
```

NOTE: overflow-x refers to the x-axis (which is horizontal scrolling). The **auto** value will only add scrollbars if they are needed.

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

Only the poster area scrolls now. This works on desktop browsers, which typically show a scrollbar. Mobile browsers typically will not show a scrollbar. On touch devices, users swipe over the area to scroll it!

Optional Bonus: Removing Extra Margin

Each movie poster has 11px of margin on the right. That creates the space between the movie posters. It also adds extra space on the far right, because there is already padding all around the movies section. We can remove that unwanted space with one CSS rule.

1. Switch back to your code editor.
2. Below the **.movies li** rule add the following new rule:

```
.movies li:last-child {
   margin-right: 0;
}
```

3. Save the file, then reload the page in your browser.
4. Scroll to the far right of the posters, and notice the margin to the right of the last poster now matches the margins between all the other posters.

Visually Indicating Scrolling

We purposely designed the size of the posters so that the right-most poster will be partially cut off (on most devices). That implies there is more for the user to scroll and see. In your designs, if you think cutting something off doesn't imply scrolling well enough, consider adding a visual indicator or instructions below the scrollable area.

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:

   ```
   <img srcset="img/sunset-320.jpg 1x, img/sunset-640.jpg 2x" src="img/
   sunset-320.jpg" alt="sunset">
   ```

   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 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.

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 6/7/8**.
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).
16. You should now see the **320px** image.
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. 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>
<img srcset="img/sunset-320.jpg 1x, img/sunset-640.jpg 2x" src="img/
sunset-320.jpg" alt="sunset"> -->
```

3. Between the commented code and the paragraph below, add this heading and img:

```
<h1>Sizes</h1>
<img src="img/sunset-1280.jpg" alt="sunset">
```

4. Add the **srcset** and **sizes** attributes, as shown below in bold:

```
<img src="img/sunset-1280.jpg" alt="sunset" srcset="" sizes="">
```

5. We're going to be adding a lot of code, so break the new attributes onto new lines:

```
<img src="img/sunset-1280.jpg" alt="sunset"
     srcset=""
     sizes="">
```

6. Add the list of images, as shown below:

```
<img src="img/sunset-1280.jpg" alt="sunset"
     srcset="img/sunset-320.jpg 320w,
             img/sunset-640.jpg 640w,
             img/sunset-1280.jpg 1280w,
             img/sunset-2560.jpg 2560w"
     sizes="">
```

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:

```
<img src="img/sunset-1280.jpg" alt="sunset"
     srcset="img/sunset-320.jpg 320w,
             img/sunset-640.jpg 640w,
             img/sunset-1280.jpg 1280w,
             img/sunset-2560.jpg 2560w"
     sizes="(min-width: 800px) 50vw,
           (min-width: 640px) 75vw,
           100vw">
```

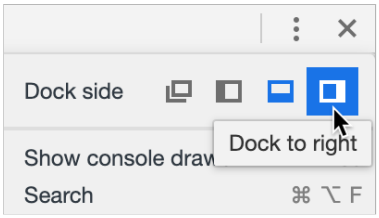
8. With **sizes** we tell the browser how big the image will be in the page, in relation to the viewport. This is similar to setting a width in HTML, in that it makes the image that size (although that can be overridden with CSS). More importantly, this tells the browser how big the image will be so 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**

   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** from the **Responsive Images** folder.
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">
   <img src="img/model-mobile-375.jpg" alt="Model">
</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, tinyurl.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.

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">
   <img src="img/model-mobile-375.jpg" alt="Model">
</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">
   <img src="img/model-mobile-375.jpg" alt="Model">
</picture>
```

9. Add **srcset** to the img tag, as shown below in bold:

```
<img src="img/model-mobile-375.jpg" alt="Model" srcset="">
```

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">
   <img src="img/model-mobile-375.jpg" alt="Model"
        srcset="img/model-mobile-375.jpg 1x,
                img/model-mobile-750.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 6/7/8**.
16. Click the **Reload** button, or hit **Cmd–R** (Mac) or **Ctrl–R** (Windows).
17. This is a 2x device, so you should see the **Mobile 750px** image.
18. From the device menu (to the left of the dimensions) choose **Responsive**.
19. Click on the **DPR** menu and choose **1**.
20. Click the **Reload** button, or hit **Cmd–R** (Mac) or **Ctrl–R** (Windows).
21. You should now see the **Mobile 375px** image. (If you don't, your mobile preview may be too wide. Resize it narrower and refresh.)
22. Click on the **DPR** menu and set it back to **Default: 2.0**.
23. Click the **Toggle device toolbar** button to leave device mode.
24. 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">
   <img src="img/model-mobile-375.jpg" alt="Model"
        sizes="100vw"
        srcset="img/model-mobile-375.jpg 1x,
                img/model-mobile-750.jpg 2x">
</picture>
```

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: 768x)"
           sizes="75vw"
           srcset="img/model-tablet-768.jpg 768w,
                   img/model-tablet-1536.jpg 1536w">
   <img src="img/model-mobile-375.jpg" alt="Model"
        sizes="100vw"
        srcset="img/model-mobile-375.jpg 375w,
                img/model-mobile-750.jpg 750w">
</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 767px 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

Media Queries & Sizes

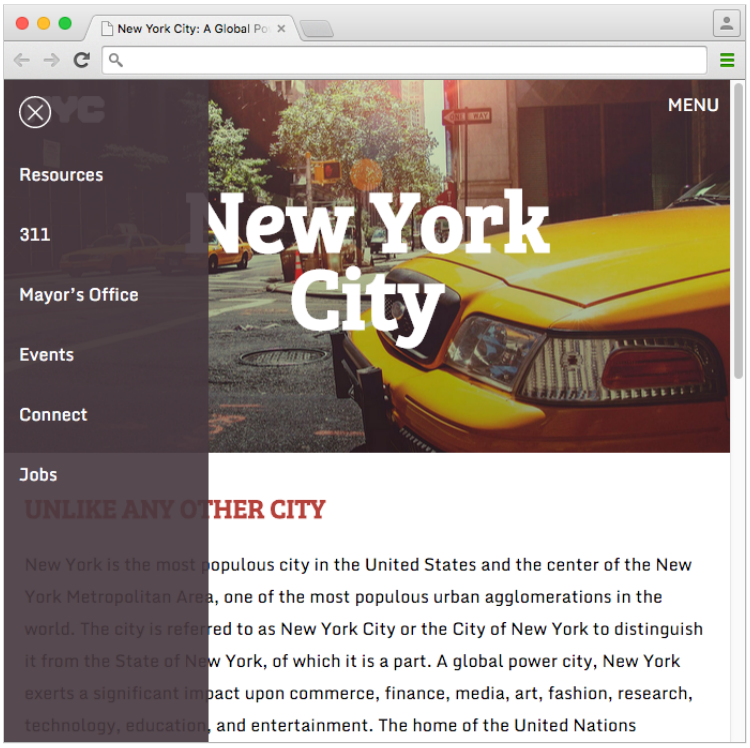
The image sizes and media queries used in this exercise are not some magical set of numbers/sizes that you'll always use. Every layout is different, so be sure to adapt these concepts to your site and content!

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)

Exercise Preview



Exercise Overview

In this exercise you will 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 (in the **Off-Screen Nav** folder).
6. Scroll down to the **Media Queries** comment. Below that, add the following code:

```
@media (max-width: 700px) {
   nav {
      position: fixed;
   }
}
```

7. Save the file.
8. Reload your browser.
9. 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.
10. Return to **main.css** in your code editor.
11. Add the following width and height to the **nav** rule you just created:

```
nav {
   position: fixed;
   width: 12em;
   height: 100%;
}
```

12. Save the file.
13. Reload your browser. The nav looks pretty close to what we want on mobile.
14. Resize the window very short, so some of the nav items get cut off.
15. 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.
16. Return to **main.css** in your code editor and add the following property to the **nav** rule as shown below in bold:

```
nav {
   position: fixed;
   width: 12em;
   height: 100%;
   overflow-y: auto;
}
```

17. Save the file and reload the page in your browser. Now you should be able to scroll through the nav items that are cut off.
18. 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.

19. Return to **main.css** in your code editor.
20. At the bottom of the styles, after the current media query, add the following new media query:

```
@media (min-width: 701px) {
   nav ul li {
      display: inline-block;
   }
}
```

21. Save the file.
22. Reload your 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.
23. Return to your code editor.
24. 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 li {
      display: inline-block;
   }
}
```

25. Save the file.
26. Reload your browser.
27. 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)!

1. Let's add the checkbox. In your code editor open **index.html**.

2. Directly below the opening **body** tag, add the following input tag as shown below:

```
<body>
   <input id="nav-checkbox" type="checkbox">
   <img src="img/nyc-logo.svg" class="nyc-logo" alt="NYC Logo">
```

3. Save the file.
4. Go to **main.css**.
5. At the bottom of the general styles, below the **nav li a:hover** rule, add this new rule:

```
#nav-checkbox {
   position: absolute;
   left: 50%;
}
```

6. Save the file.
7. Reload your 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.
8. Now we need to make the checkbox hide the menu. In your code editor, go to **main.css**.
9. In the **max-width: 700px** media query, below the **nav** rule, add the following rule as shown below in bold:

```
#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).

10. Save the file.
11. Reload your browser, making sure the window is smaller than 700px.
12. 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**.

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 the file.

5. Reload your 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.

6. Return to **main.css**.

7. 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;
}
```

8. Save the file.

9. Reload your browser. Toggle the checkbox to see that the nav now slides in nicely from the side. Cool!

Adding a Proper Menu Button

We need something better for our users to click than a checkbox. We can use a form **label**. 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. Return to **index.html** in your code editor.
2. Just under the opening **body** tag, add the code as shown below in bold:

```
<input id="nav-checkbox" type="checkbox">
<label for="nav-checkbox" class="menu-button">MENU</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: 10px;
   color: #fff;
}
```

7. Save the file.
8. Reload your browser.
9. In the upper-right corner, click on **MENU** and:
   - Notice that the checkbox changes to the checked state, even though we're just clicking on the label. Cool!
   - Notice that when you mouse 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 button.
   - If you double–click the **MENU** button it gets selected like text. This doesn't happen on links or buttons, so it would be nice to prevent the text from being selected.
10. Return to **main.css** in your code editor.

11. At the bottom of the **.menu-button** rule, add the new code as shown below in bold:

```
.menu-button {
   position: absolute;
   top: 10px;
   right: 10px;
   color: #fff;
   cursor: pointer;
   -webkit-user-select: none;
   -moz-user-select: none;
   -ms-user-select: none;
   user-select: none;
}
```

12. Save the file.
13. Reload your browser.
14. Mouse over **MENU** to see the proper hand cursor.
15. Double–click the **MENU** button to see we can no longer select it like text.
16. Resize the window larger than 700px. We need to hide the MENU button here on desktop view.
17. Return to **main.css** in your code editor.
18. At the start of the min-width: **701px** media query, add the following new rule:

```
.menu-button {
   display: none;
}
```

19. Save the file.
20. Reload your browser. The MENU button should no longer appear in the desktop layout. The checkbox is still showing, but we'll hide that later.
21. 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. Return 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">
      <img src="img/close.svg" alt="close">
   </label>
   <ul>
```

Note again that we are giving this label the same **for** value as the MENU button. 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 {
   width: 30px;
   margin: 15px;
   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 the file.

7. Reload your browser.

8. Click the **MENU** button to open the nav, then click the circular close button to close it.

Wow, it already works! That's because it's just another label that triggers the same checkbox. The functionality of the checkbox was already working, this is just a second way to trigger it.

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. Return 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">MENU</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 the file.

6. Reload your browser, noticing the slightly transparent black background over most of the page.

7. 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.

8. We only want the overlay to be available when the menu is open, and invisible at all other times. Return to **main.css**.

9. Find the **overlay** rule and add the following property shown in bold:

```
.overlay {
   display: none;
   position: fixed;
   height: 100%;
   width: 100%;
   background: rgba(0,0,0, .5);
}
```

10. Below the **overlay** rule, add the following new rule:

```
#nav-checkbox:checked ~ .overlay {
   display: block;
}
```

11. Save the file.

12. Reload your browser. Notice that the overlay is hidden.

13. Click the **MENU** button to see that the nav appears, as well as the semi-transparent black background behind it.

14. Click anywhere on the semi-transparent dark background to close the menu.

15. Return to **main.css**.
16. You could keep the dark overlay if you like it, but we want to hide the overlay (while keeping the functionality). In the **.overlay** rule, delete the **background** property.
17. Save the file.
18. Reload your browser.
19. Open the menu, then click anywhere outside the menu and it will still close.

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. Return to **main.css**.
2. In the general styles, find the **#nav-checkbox** rule.
3. Delete the **left: 50%** line of code.
4. Also make the following edits shows in bold:

```
#nav-checkbox {
   position: fixed;
   clip: rect(0,0,0,0);
}
```

Here are a few things to note:

- While **display: none** may have seemed more intuitive, the clip property is more accessible (allowing users to tab into the checkbox with the keyboard).
- 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.
- The **clip** property is deprecated and the W3C advises the use of clip-path instead. Clip-path, however, is not yet supported by Internet Explorer or Edge, so we prefer to use **clip** in this instance despite its deprecated state.

5. Save the file.
6. Reload your browser and test out the completed page!

Optional Bonus: Flipping the Nav to the Right Side

With a small tweak, we can make the nav come in from the right.

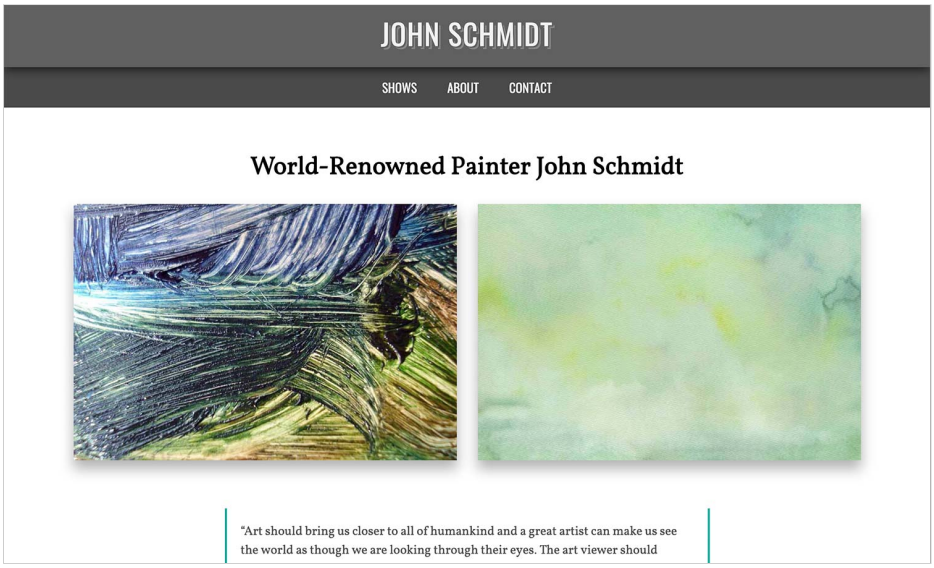
1. Return to **main.css**.

2. In the **max-width: 700px** media query, change **left** to **right** in both of the following rules (changes are shown in bold). Don't miss both places!

```
nav {
   position: fixed;
   width: 12em;
   right: -12em;
   height: 100%;
   overflow-y: auto;
   z-index: 100;
   transition: all .2s ease;
}
#nav-checkbox:checked ~ nav {
   right: 0;
}
```

3. Save the file.
4. Reload your browser. Test out the nav and see that it comes in from the right side now.
5. If you get done early, check out the bonus exercise **Slide-Down Top Nav Using Only CSS** where you'll learn how a few edits to the CSS can make this menu slide down from the top, instead of from the side.

Exercise Preview



Exercise Overview

In this exercise, you'll use box-shadow and text-shadow to add visual depth. You will also learn how to adjust an element's front to back stacking order using relative position and the z-index property.

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 **Box-Shadow and Text-Shadow** 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 **Box-Shadow and Text-Shadow** folder.
4. Preview **index.html** in Chrome (we'll be using its DevTools).

   Behold the mostly finished portfolio website of world-famous artist John Schmidt. Our only task is to add drop shadows.
5. The first thing we want to do is add a box-shadow to the four large featured art images on the page 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**.)
6. Leave the page open in Chrome so we can come back to it later.

Using the CSS Box-Shadow Property

The CSS3 property **box-shadow** adds a drop shadow behind an element with a single line of code.

1. Return to your code editor.
2. Open **main.css** from the **css** folder (in the **Box-Shadow and Text-Shadow** folder).
3. 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;
}
```

4. Take a moment to review the code you wrote. The first value sets the horizontal (x) offset, the second sets the vertical (y) offset, the third value sets the blur radius, and the fourth sets the color.
5. Save the file, then reload the page in Chrome. Wow, that shadow is very prominent.
6. Let's open the DevTools so we can tweak it. **Ctrl–click** (Mac) or **Right–click** (Windows) on any of the large featured art images and select **Inspect**.
7. An `<img>` tag should be selected. In the **Elements** panel of the DevTools, click the line above the selected image: **`<a href="#">`**.
8. In the **Styles** panel:
   - There may be two rules for **.category a** so find the one with the box-shadow.
   - Click on the box-shadow's first px value (horizontal offset) and change it to **-2px**.
   - 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 off the bottom right of an element. Negative values place the shadow off the top left of an element.
9. We can move the shadow slightly to the bottom (no horizontal offset) and create a larger softer shadow. Change to these bold values:

   `box-shadow: 0 2px 20px #000;`
10. To make the shadow much harder and smaller, change the blur as shown in bold:

    `box-shadow: 0 2px 2px #000;`

11. Let's experiment with the color. Change the color as shown:

```
box-shadow: 0 2px 2px #f00;
```

Wow that's red! While we could change the color to a subtle hexadecimal gray, hex colors are opaque. It would look more realistic if we had a partially transparent shadow. Let's use RGBA.

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

13. 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.

14. Save the file, then reload the page in Chrome. Nice, we have a soft, subtle shadow that works well on top of a white background! RGBA shadows look natural over any color background due to the partial transparency.

15. Still in Chrome, notice the gray band that contains **John Schmidt** at the top of the page. We want to add box-shadow to this section to make it look like it is casting a shadow on top of the darker gray navigation directly below it. This top section (above the nav) is a div with a class of **logo-wrapper**.

16. Leave the page open in the browser and switch to **main.css** in your code editor.

17. Add a box-shadow to the existing **.logo-wrapper** rule, as follows:

```
.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 color;**. 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.

Because we want the drop shadow to go straight down, we input a horizontal offset of 0. Our shadow is 80% gray so it will be visible over the dark gray nav.

18. Save the file, then reload the page in Chrome. Oh no! We expected to see a very dark shadow to be cast onto the navbar, but it's nowhere to be seen 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 the file, then reload the page in Chrome. Yes, our shadow is showing!

   But why is it working? **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**.
2. An `<img>` tag should be selected. In the **Elements** panel of the DevTools, click the line above the selected image: **`<a href="#">`**.

3. In the **Styles** panel:
   - In the **.category a** rule, click on the **box-shadow's** value.
   - Add **inset** at the beginning so you end up with **inset 0 12px 18px rgba(0,0,0, .3)**
   - Don't be surprised when the shadow disappears!

   Why did the shadow disappear? Because the image inside the link is covering over it! Let's send the image backward in the stacking order.
4. In the **Elements** panel of the DevTools, click on the **img** tag inside the selected link `<a href="#">`.
5. In the **Styles** panel there should be an **img** rule.
6. 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.
7. Type **`z-index: -1;`** which should complete that property and put you on another line, ready to type another property.
8. We still don't see the shadow because z-index can't work with the default static position. Type out **position: relative;** and the inner 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.
9. 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

CSS lets us add text shadows as well. Let's give the **John Schmidt** text at the top of the page more pizazz by adding **text-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**. In the existing **.logo-wrapper a** rule, add the following bold code:

```
.logo-wrapper a {
   display: inline-block;
   text-shadow: 4px 4px 4px #000;
}
```

3. Save the file, then reload the page in Chrome.
4. The text should now have a black shadow. **Ctrl–click** (Mac) or **Right–click** (Windows) on **John Schmidt** text and choose **Inspect**.
5. On the page (not in the DevTools), click on any blank area to deselect the text.

6. Let's try out a super sharp shadow. In the **Styles** panel, in the **.logo-wrapper a** rule, change both the blur value to **0** as shown in bold:

```
text-shadow: 4px 4px 0 #000;
```

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 need to layer two text-shadow effects on top of each other. We want to create a shadow like the one we just created in the DevTools, but we want it to be partially covered by a shadow that is indistinguishable from the logo-wrapper's background. For this to work properly, we must stack the effects in the correct order.

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. In the **.logo-wrapper a** rule, set the blur to **0** and the color to a gray (because we thought black was too dark):

```
.logo-wrapper a {
   display: inline-block;
   text-shadow: 4px 4px 0 #666;
}
```

NOTE: Make sure you didn't miss setting the blur to 0!

4. 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 property and do not miss the **comma**!

```
.logo-wrapper a {
   display: inline-block;
   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.

5. Save the file, then reload the page in Chrome. Neat-o!
6. Hover over **John Schmidt** text. It would be interesting if we made the shadow shift position on hover.

7. Switch back to **main.css** in your code editor.
8. Copy the entire **.logo-wrapper a** rule.
9. Paste the code directly below and add the following bold edits to specify negative values for the horizontal offset on hover. Be sure to delete the **display** property from the **hover** rule.

```
.logo-wrapper a {
   display: inline-block;
   text-shadow: 2px 2px 0 #444, 4px 4px 0 #666;
}
.logo-wrapper a:hover {
   text-shadow: -2px 2px 0 #444, -4px 4px 0 #666;
}
```

10. Save the file, then reload the page in Chrome.
11. Hover over the text to see the shadow shift. Artsy!

CSS Box Shadow

Box-shadow can have up to 6 values: inset hoff voff blur color spread

- **Inset (optional)**: Inset provides an inner shadow to the element rather than placing it outside the box. Inset can be declared first or last. All other values must be in the order listed above.
- **Horizontal offset (required)**: A positive value means the shadow will be on the right. A negative value will put the shadow on the left.
- **Vertical offset (required)**: A positive value means the shadow will be below the box. A negative value means the shadow will be above.
- **Blur radius (optional)**: The higher the number value, the more blurred the shadow will be. A value of 0 will give you a hard edge.
- **Color (required)**: The value can be expressed as hexadecimal, RGB, RGBA, HSL, or HSLA.
- **Spread radius (optional)**: Spread can be added between the blur and color values. Positive values will increase the size of the shadow. Negative values will decrease the size of the shadow. The default spread value is 0, so the shadow is the same size as the blur.

CSS Text Shadow

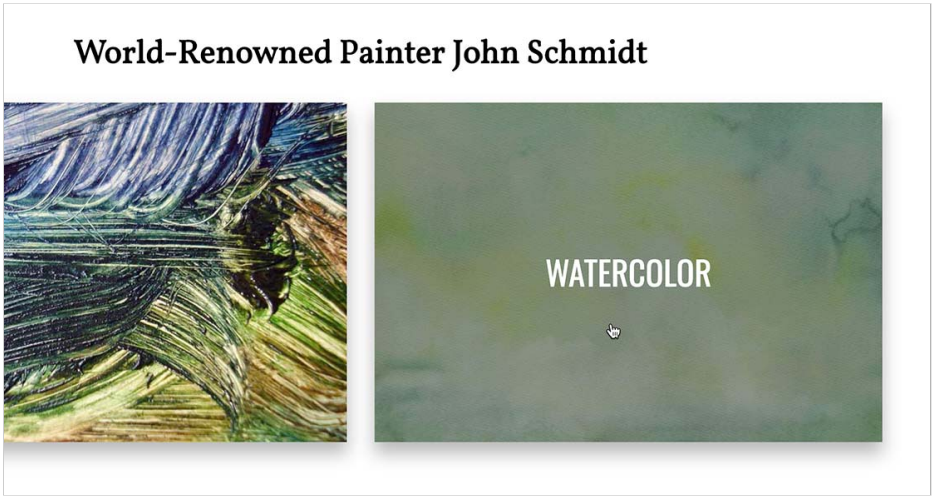
Each shadow value must specify a shadow offset and, optionally, a blur radius and color.

The **offset** is specified using two length values: the first value represents the horizontal distance to the right of the text (if it's positive), or to the left of the text (if the value is negative); the second value represents the vertical distance below the text (if it's positive) or above the text (if it's negative).

The **blur** radius is specified after the offset values; it's a length value that represents the size of the blur effect. If no radius is specified, the shadow will not be blurred.

The **color** can be specified before or after the offset and blur radius values.

Exercise Preview



Exercise Overview

In this exercise, we will investigate various ways to hide and show elements: setting the display property to none, the visibility to hidden, and setting opacity to 0. All of these approaches make content invisible for sighted users, but they have different effects on how the elements are rendered and whether or not the hidden text is accessible to the visually impaired.

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 **Hiding and Showing Elements** 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 **Hiding and Showing Elements** folder.
4. Preview **index.html** in Chrome (we'll be using its DevTools).

   We want to create an overlay on top of each of the four large featured art images. Each photo represents a different category (Oil, Watercolor, Mixed Media, and Installation), so we want to add this text to the respective overlays. Let's create the Oil overlay first.
5. Leave the page open in Chrome so we can come back to it later.

Styling & Positioning the Overlay

1. Switch back to **index.html** in your code editor.
2. In the first ** ** tag, add the following bold markup:

```
<div class=category>
   <a href="#">
      <img src="img/oil.jpg" alt="">
      <div class="description">Oil</div>
   </a>
</div>
```

3. Save the file.
4. Open **main.css** from the **css** folder (in the **Hiding and Showing Elements** folder).
5. Below the **.category a** rule (in the general styles, not the media query), style the overlay as shown in bold:

```
.category .description {
   text-align: center;
   text-transform: uppercase;
   font-family: Oswald, sans-serif;
   font-size: 2rem;
   line-height: 4rem;
   color: #fff;
   background: rgba(0,0,0, .4);
}
```

6. Save the file, then reload the page in Chrome. It's a start, but in order to place the overlay on top of the image, we need to use absolute positioning.
7. Switch back to **main.css** in your code editor.
8. 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 {
   box-shadow: 0 12px 18px rgba(0,0,0, .3);
   margin: 30px 0;
   display: block;
   text-decoration: none;
   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).

9. In the **.category .description** rule, set a position as follows:

```
.category .description {
   text-align: center;
   text-transform: uppercase;
   font-family: Oswald, sans-serif;
   font-size: 2rem;
   line-height: 4rem;
   color: #fff;
   background: rgba(0,0,0, .4);
   position: absolute;
}
```

10. Save the file, then reload the page in Chrome. It's a start, but the description has collapsed. To properly position it and open it up, let's give it positioning coordinates.

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

12. 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;
   top: 0;
   right: 0;
   bottom: 0;
   left: 0;
}
```

13. Save the file, then reload the page in Chrome. The overlay is stretching to fill its container but we want to center the text vertically.

14. Resize the browser window narrow and wide, to see that our layout is responsive. The image containers size up and down and maintain the same aspect ratio. This means we can't use a fixed-pixel value to vertically center the text.

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

    Unfortunately, setting the vertical-align property to middle will not work. That property only works on inline or table elements, and we need the div to display as its native block so the overlay keeps on filling its container.

16. Fortunately there is solution that uses some math to calculate exactly how much top padding will center the text. Add the following to the **.category .description** rule:

```
.category .description {
    CODE OMITTED TO SAVE SPACE
    left: 0;
    padding-top: calc(33.33% - 4rem/2);
}
```

> **How Did We Calcuate This?**
>
> Keep in mind that our text's line height is 4rem and that padding is determined by the width of the container, not the height. All our images are the same aspect ratio. Once we know the ratio, we can divide it in half (for the top half) which will push the text half way down. That would put the top of the text half way down, but we want the middle of the text centered, so then we subtract half of the current line height. Here's our calculation:
>
> - (image height / image width)/2 - line height/2
> - In our case (560px/840px)/2 = .3333 (which is 33.33%)

17. Save the file, then reload the page in Chrome.
18. Resize the browser window narrow and wide, to see that the Oil text is always centered. Leave the browser window as wide as possible when you're done.

Removing an Element from the Normal Document Flow with Display: None

Let's make this Oil text overlay only appear when a user mouses over the link. This means we need to hide it initially, and then show it on hover. There are multiple ways to hide things. Let's start by taking a look at setting **display** to **none**.

1. Switch back to **main.css** in your code editor.
2. At the end of the **.category .description** rule, add the following bold code:

```
.category .description {
    CODE OMITTED TO SAVE SPACE
    padding-top: calc(33.33% - 4rem/2);
    display: none;
}
```

3. Save the file, then reload the page in Chrome.

   The Oil text overlay should now be hidden, as if the element doesn't exist. This is because display: none removes an element from the normal document flow. We'll take a closer look at this shortly.

4. Let's redisplay the element on hover. Switch back to **main.css** in your code editor.

   We can't hover over the description because it has been removed from the flow of the document. We can hover over the category's anchor tag (which is on-screen at all times) and then show the description nested inside.

5. Below the **.category .description** rule, add this new following bold rule:

```
.category a:hover .description {
   display: block;
}
```

   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 anchor tag we're hovering over (but only for anchors in the .category div). Because this rule is more specific than the rule above it, it will override that rule's display property.

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

   - The overlay should initially be hidden.
   - Hover over the oil painting image to see the overlay reappear. Great!

Hiding/Showing Elements with Visibility

Setting the display property to none removes an element from the flow of the document, making other elements on the page unaware of it. This poses a challenge for visually impaired users because screen readers also treat these elements as if they do not exist. These users may get lucky and hear their screen reader read out "Oil" if they hover over the link, but it's not welcoming to people with special needs.

The visually impaired depend on web developers to follow best-practices for writing accessible code. There are two properties that can hide an element without hiding it from screen readers. Let's first try setting **visibility** to **hidden**.

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

2. We want to temporarily disable the two display property declarations. Comment them out as shown in bold. TIP: Visual Studio Code users can put the cursor in each line and hit **Cmd–/** (Mac) or **Ctrl–/** (Windows).

```
.category .description {
    CODE OMITTED TO SAVE SPACE
    padding-top: calc(33.33% - 4rem/2);
    /* display: none; */
}
.category a:hover .description {
    /* display: block; */
}
```

3. Setting the **visibility** property to **hidden** hides an element from sighted people but not from screen readers. We can reshow the element by setting it to **visible**. Add each property in the appropriate rule, as shown in bold:

```
.category .description {
    CODE OMITTED TO SAVE SPACE
    padding-top: calc(33.33% - 4rem/2);
    /* display: none; */
    visibility: hidden;
}
.category a:hover .description {
    /* display: block; */
    visibility: visible;
}
```

4. Save the file, then reload the page in Chrome.
   - Hover over the image of the oil painting to see the overlay reappear, just like it did with display: none.
   - This looks the same to sighted users, but is treated differently by screen readers.

Hiding/Showing Elements with Opacity

Another way to hide elements without "hiding" them from screen readers is to set **opacity** to **0**. Opacity is more flexible than the previous two solutions because you can give it any value between 0 (fully transparent) and 1 (fully opaque).

1. Switch back to **main.css** in your code editor.
2. Comment out the **visibility** properties in both rules.

3. Make the **.category .description** 50% transparent, as shown in bold:

```
.category .description {
    CODE OMITTED TO SAVE SPACE
    /* display: none; */
    /* visibility: hidden; */
    opacity: .5;
}
```

4. Save the file, then reload the page in Chrome to see the description is semi-transparent. Interesting to see, but not what we want.

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

6. Let's make the .description totally transparent initially, and then fully opaque on hover with the following bold code:

```
.category .description {
    CODE OMITTED TO SAVE SPACE
    /* visibility: hidden; */
    opacity: 0;
}
.category a:hover .description {
    /* display: block; */
    /* visibility: visible; */
    opacity: 1;
}
```

7. Save the file, then reload the page in Chrome. Hover over the image and once again it should visually appear the same, but this code (like visibility) is also accessible to screen readers.

How Display, Visibility, & Opacity Differ

Knowing how these approaches differ will help you choose the right solution. To get a better understand them, let's test some code in the DevTools.

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

2. While previewing the page, you may have noticed the two pullquotes. Each is wrapped in a blockquote tag. In the **blockquote** rule, add the following bold properties, commenting out the last three properties in advance, as shown:

```
blockquote {
   border-left: solid 3px #17a99c;
   border-right: solid 3px #17a99c;
   color: #505050;
   margin: 20px;
   padding: 20px;
   cursor: pointer;
   /* display: none; */
   /* visibility: hidden; */
   /* opacity: 0; */
}
```

   NOTE: We are going to run some tests in the browser. These commented properties can be enabled and disabled via checkboxes in the DevTools. Make sure each property declaration is commented out separately, as we want to be enable/disable each property individually.

3. Save the file, then reload the page in Chrome.

4. Below the top two images, hover over the blockquote that starts with **"Art should bring us closer..."** to see the cursor turn into a pointer. In order to see if the user can interact with the hidden blockquotes, we'll look at one property at a time.

5. **Ctrl–click** (Mac) or **Right–click** (Windows) on the blockquote and select **Inspect**.

6. If the `<blockquote>` is not selected in the **Elements** panel of DevTools, select it now.

7. In the **Styles** panel, find the **blockquote** rule with the disabled properties. You may have to scroll down.

8. Check the box next to **display: none;** to enable that property and:

   - Scroll down the page to see that all the blockquote elements have disappeared and are completely removed from the normal document flow.
   - Hover over the empty white area where the blockquote was to see that the pointer does not appear.

9. Back in the **Styles** panel, uncheck **display: none;** to disable it once more.

10. Check the box next to **visibility: hidden;** to enable that property and:
    - Notice the gaps that appeared in place of each blockquote. This shows that the blockquotes are still part of the flow of the document, even though you can't see the content.
    - Hover over one of the gaps to see that the cursor does not turn into a pointer. Users cannot interact with elements hidden in this way.
11. In the **Styles** panel, uncheck **visibility: hidden;**
12. Check on **opacity: 0** to see:
    - Once again, there is a gap. The blockquotes are in the normal document flow.
    - Hover over one of the gaps to see the pointer hand. This happens because even when an element's opacity is set to 0, it is still clickable and interactive!

Finishing Up

Which solution works best for our project? We want our code to be fully accessible, so that eliminates the display property. In the next exercise, we will add CSS transitions so the overlay smoothly animates between the normal and hover states. There is no way to transition the visibility property. Because opacity values are numeric, they work well with CSS transitions, therefore we'll use opacity.

1. Switch back to **main.css** in your code editor.
2. In the **blockquote** rule, remove the four test properties at the end of the rule so it looks as shown below:

```
blockquote {
   border-left: solid 3px #17a99c;
   border-right: solid 3px #17a99c;
   color: #505050;
   margin: 20px;
   padding: 20px;
}
```

3. In the **.category .description** rule, remove the **display** and **visibility** declarations so you end up with this:

```
.category .description {
   CODE OMITTED TO SAVE SPACE
   padding-top: calc(33.33% - 4rem/2);
   opacity: 0;
}
```

4. Do the same to the **.category a:hover .description** rule so you end up with:

```
.category a:hover .description {
   opacity: 1;
}
```

5. Save **main.css**.

6. Save the file, then reload the page in Chrome. Make sure the hover effect is still working as expected.

Optional Bonus: Adding the Three Remaining Overlays

To complete the page, you'll need to add overlays for the three remaining images.

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

2. Under the **watercolor** image, add the following code:

```
<a href="#">
   <img src="img/watercolor.jpg" alt="">
   <div class="description">Watercolor</div>
</a>
```

3. Do the same for the **mixed-media**:

```
<a href="#">
   <img src="img/mixed-media.jpg" alt="">
   <div class="description">Mixed Media</div>
</a>
```

4. Do the same for the **installation**:

```
<a href="#">
   <img src="img/installation.jpg" alt="">
   <div class="description">Installation</div>
</a>
```

5. Save the file, then reload the page in Chrome.

6. Hover over all the images to make sure the code works and the labels are all correct.

Exercise Preview

Exercise Overview

In the portfolio page we've been working on, the hover styles abruptly change at the moment the user hovers over. In this exercise, you'll smooth out these changes by adding CSS3 transitions. Transitions animate properties (such as opacity and color) much like "tweening" does, making the browser show the in between states over a given time.

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 **Transitions** 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 **Transitions** folder.
4. Preview **index.html** in a browser.

   Hover over the links 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.
5. Leave the page open in the browser so we can come back to it later.

Transition-Property & Transition-Duration

A CSS transition needs to know two things: what property 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 (i.e. the hover). This is because the browser needs to know what to do **before** it starts transitioning. This also frees you up to trigger the transition from other additional pseudo classes like **:focus** or **:active** or even trigger the transition via JavaScript.

1. Switch back to your code editor.
2. Open **main.css** from the **css** folder (in the **Transitions** folder).
3. In the **header a** rule, add the code shown below in bold:

```
header a {
    CODE OMITTED TO SAVE SPACE
    text-transform: uppercase;
    transition-property: color;
    transition-duration: 2s;
}
```

NOTE: These two properties are fairly self-explanatory:

- You're telling the browser to **transition** the color **property** on this element.
- You're setting the transition's **duration** to last for 2 seconds.

You can enter a transition-duration in seconds (**s**) or milliseconds (**ms**). **2s** is the same as **2000ms**. We will make the effect faster after some testing.

4. Save the file, then reload the page in your browser.
5. If you previewed in Chrome or Safari, you may have noticed **John Schmidt** and nav links had a weird color transition when the page loaded. We'll fix that in a moment.
6. Hover over any of the nav links at the top of the page. Instead of immediately toggling to green, the color takes 2 seconds to smoothly transition.
7. Hover off of the element to see the color slowly transition back to white.
8. Hover over **John Schmidt**. The text color smoothly transitions, but the shadow immediately and abruptly shifts position. We can fix that by telling the text-shadow to transition along with the color.
9. Switch back to **main.css** in your code editor.
10. We can add multiple transition-property values by separating them with **commas**. As shown in bold, tell the browser to transition the text-shadow as well:

```
header a {
    CODE OMITTED TO SAVE SPACE
    transition-property: color, text-shadow;
    transition-duration: 2s;
}
```

11. Save the file, then reload the page in your browser.
12. Hover over **John Schmidt** to see that this time, the color and the text-shadow both change simultaneously. Much nicer!

Fixing a Problem on Page Load

As of this writing, Chrome and Safari have a bug that makes transitions fire on page load. They should not, and Firefox does not do this. Safari only does it on the initial page load, but not when you hit refresh. Chrome does it on page load and whenever you hit refresh.

The best workaround we have found is to initially disable all transitions during page load, then turn them back on after the page has loaded. This will require a tiny bit of JavaScript, but don't worry… we've written all the code for you to use!

1. Switch back to your code editor.
2. Open **prevent-transition-flicker.html** from the **snippets** folder (in the **Transitions** folder).
3. There's comments in this file to tell you what to do, but we'll walk you though the steps. Copy the line of code under **STEP #1**:

```
<style>.preload * {transition:none !important;}</style>
```

4. Switch to **index.html**.
5. Paste it at the end of the **head** tag, as shown below:

```
    <link rel="stylesheet" href="css/main.css">
    <style>.preload * {transition:none !important;}</style>
</head>
```

6. On the **body** tag add **class="preload"**, as shown below:

```
<body class="preload">
```

7. Switch back to **prevent-transition-flicker.html**
8. Copy the line of code under **STEP #3**:

```
<script>document.getElementsByTagName("body")[0].classList.remove("preload");</
script>
```

9. Close **prevent-transition-flicker.html**.
10. Switch to **index.html**.
11. Paste the code after of the start of the **body** tag, as shown below:

```
<body class="preload">
    <script>document.getElementsByTagName("body")
[0].classList.remove("preload");</script>

    <header>
```

12. Save the file, then reload the page in your browser.
    - In Chrome and Safari, you should no longer see the transition when the page loads!
    - Hover over **John Schmidt** and the transition should still work.
13. What does this code do?
    - The **preload** class is used to set everything in the page to **transition:none;** (for page load) to disable all transitions. (The * selector means any element.)
    - Once the page has loaded, JavaScript then removes the **preload** class from the **body** tag, which removes **transition:none;** so transitions can work again!

Transition Shorthand & the All Keyword

Now we can get back to working on our transitions. Let's see how to combine the two transition properties into one, using shorthand.

1. Switch back to **main.css** in your code editor.
2. At the end of the **header a** rule, delete the **transition-duration** line.
3. As shown below, change transition-property to **transition** and add back in duration values as shown in bold:

```
header a {
    CODE OMITTED TO SAVE SPACE
    text-transform: uppercase;
    transition: color 2s, text-shadow 4s;
}
```

NOTE: Transition shorthand is the property you wish to transition, followed by the duration (followed by an optional delay, which we're not using).

4. Save the file, then reload the page in your browser.
5. Hover over **John Schmidt** to see the color transitions in 2 seconds, but the shadow takes 4 seconds.

We're only transitioning 2 properties, but what if we wanted to change 5 properties at once? The code could become unwieldy, but instead we can use the **all** keyword.

6. Switch back to **main.css** in your code editor.
7. In the **transition** property, switch to **all** and shorten the duration so the change happens faster:

```
transition: all 300ms;
```

8. Save the file, then reload the page in your browser. Hover over **John Schmidt** or the nav links so see a much better speed (with less code).

Animating the Description Overlays

1. Hover over one of the four featured art images to see the hover we added in the last exercise. This goes straight from an opacity of 0, to an opacity of 1 on hover.
2. Because opacity values are numeric (unlike the equally accessible visibility property), we can transition them. Switch back to **main.css** in your code editor.
3. In the **.category .description** rule, add the following transition:

```
.category .description {
    CODE OMITTED TO SAVE SPACE
    opacity: 0;
    transition: all 1s;
}
```

4. Save the file, then reload the page in your browser.
5. Hover over one of the featured art images to watch the description subtly fade in.

   One problem with this is mobile users don't get the benefit of being able to hover, so they will not see the label and therefore won't know what each image links to. No worries! With a little tweak we can make this awesome for mobile and desktop users alike. We can initially show the label at the bottom of the photo, and then on hover slide it up to fully cover the photo.

6. Switch back to **main.css**.
7. We now want to always see the text overlay, so remove **opacity** from both rules as shown below:

```
.category .description {
    CODE OMITTED TO SAVE SPACE
    padding-top: calc(33.33% - 4rem/2);
    transition: all 1s;
}
.category a:hover .description {

}
```

8. Initially we want the label to be at the bottom of the photo. That means we need to move the top edge down so the overlay is only as tall as one line (keep in mind our line-height is 4rem). Edit the rule for **.category .description** as shown in bold:

```
.category .description {
```

CODE OMITTED TO SAVE SPACE

```
   top: calc(100% - 4rem);
   right: 0;
   bottom: 0;
   left: 0;
   padding-top: calc(33.33% - 4rem/2);
   transition: all 1s;
}
```

9. On hover, the top should move all the way back up to the top of the photo, so in the **.category a:hover .description** rule add the following:

```
.category a:hover .description {
   top: 0;
}
```

10. Cut the **padding-top: calc(33.33% - 4rem/2);** line from the **.category .description** rule.

11. Paste it into the **:hover** rule, so you end up with the following code:

```
.category .description {
```

CODE OMITTED TO SAVE SPACE

```
   top: calc(100% - 4rem);
   right: 0;
   bottom: 0;
   left: 0;
   transition: all 1s;
}
.category a:hover .description {
   top: 0;
   padding-top: calc(33.33% - 4rem/2);
}
```

That extra space above the text is only needed when the overlay expands to be the full height of the photo.

12. Save the file, then reload the page in your browser, and:

- Notice the labels are visible at the bottom of each photo. Now everyone can see what these are without having to do anything!
- Hover over the photo and watch the overlay expand up. Cool!

NOTE: In older versions of Microsoft Edge (before switching to the Chrome rendering engine), CSS transitions do not work with calc(), therefore the overlay won't animate. Everything else works (and the page still looks decent) so we're fine with Edge being different. If you're not OK with that, you'll have to figure out a fixed amount that doesn't require using calc(). That may require more media queries, etc. Now that Edge has switched to using Chrome's rendering engine (where the transitions work), we think this code works well enough as it is.

Adding Easing with Transition-Timing-Function

Easing changes the rate of speed of an animation, going slower or faster at certain times. With the **transition-timing-function** property, we can use: **ease**, **linear**, **ease-in**, **ease-out**, **ease-in-out**, and **cubic-bezier** (which allows you to define your own timing curve). The default is **ease**, but let's try some others.

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

```
.category .description {
    CODE OMITTED TO SAVE SPACE
    transition: all 1s;
    transition-timing-function: linear;
}
```

3. Save the file, then reload the page in your browser.
4. Hover over one of the images. A linear ease moves at a constant speed for the entire animation, which is very robotic. Let's try some other options.
5. Return to **main.css** in your code editor.
6. Delete the line of code you just wrote.
7. Let's incorporate the ease into the shorthand. Add an **ease-in** function to the end of the shorthand as shown in bold:

```
.category .description {
    CODE OMITTED TO SAVE SPACE
    transition: all 1s ease-in;
}
```

8. Save the file, then reload the page in your browser.
9. Hover over one of the images. It's very subtle, but **ease-in** will start slow, then go faster at the end. It "eases into" the animation. It works, but the basic eases are a bit ho-hum and very subtle.

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 CSS snippets with cubic-bezier coordinates you can easily customize.

1. In a new browser tab/window, go to **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, from jQuery to the GreenSock Animation Platform (GSAP), to CSS and more. For more info visit **robertpenner.com/easing**

3. Set the **Duration** to **300** (that's 300 milliseconds or 0.3 seconds).
4. Click the **Height** 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 rule for **.category .description**, 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 the file, then reload the page in your browser.
9. Hover over the photos and notice the animation starts out fast and slows down at the end, so the transition 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

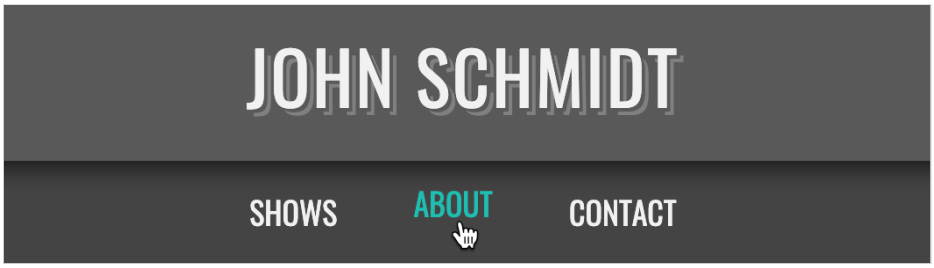
1. Find the **.category a:hover .description** rule and change the selector to the following. Don't miss the comma after the first part!

```
.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, **skew** (tilt) them, **rotate** them, and **translate** (move) them. Transforms can even be animated when combined with CSS transitions!

Testing Transforms Using the DevTools

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 **Transforms and Transitions** 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 **Transforms and Transitions** folder.
4. 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.
5. 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**.
6. In the **Elements** panel of DevTools, the **description** div should be selected. Click on the **`<a href="#">`** above it.
7. In the **Styles** panel, there may be two **.category a** rules, find the **.category a** rule with the **box-shadow** property.
8. Click once to the right of the last property **position: relative** to add a new property.

9. 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.

10. Notice that the photos have immediately 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.

11. Change the scale value in DevTools to be twice the normal size:

    ```
    transform: scale(2);
    ```

12. Yikes, those are too huge! However, the layout is still unaffected even when the photos are larger than normal. To see this more clearly, scale the elements to 140% of their initial size:

    ```
    transform: scale(1.4);
    ```

13. 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. Switch back to your code editor.

2. Open **main.css** from the **css** folder (in the **Transforms and Transitions** 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 overlays that have a class of description.

3. Below the **.category a** rule, add the following new rule:

   ```
   .category a:hover,
   .category a:focus {
      transform: scale(1.2);
   }
   ```

4. Save the file.

5. Return to **index.html** in the browser and reload the page.

6. 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 natural size, and does not have a smooth 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).
7. Let's solve the stacking problem first. Leave the page open in the browser so we can come back to it later.
8. Return to **main.css** in your code editor.
9. All of the links inside category divs are nested at the same level in the code. Elements that are further down the page will layer on top of the previous elements. To make sure that the :hover always stacks on top, add the z-index as shown in bold:

```
.category a:hover,
.category a:focus {
   transform: scale(1.2);
   z-index: 10;
}
```

10. Save the file, then 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.
11. 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.
12. In the **.category .description** rule, **copy** the transition property:

```
transition: all 300ms cubic-bezier(0.190, 1.000, 0.220, 1.000);
```

13. 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
   position: relative;
   transition: all 300ms cubic-bezier(0.190, 1.000, 0.220, 1.000);
}
```

14. Save the file, then 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.
15. Switch back to **main.css** in your code editor.
16. Change the **.category a:hover** transform's value from 1.2 to a more subtle **1.05** as shown in bold.

```
.category a:hover,
.category a:focus {
   transform: scale(1.05);
   z-index: 10;
}
```

17. Save the file, then reload the page in your browser.
18. 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. Switch back to **main.css** in your code editor.
2. 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**.

3. Save the file, then reload the page in your browser.
4. 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.

Rotate & Skew Transforms

In addition to scaling, we can also rotate and tilt elements.

1. Switch back to **main.css** in your code editor.
2. In the **.category a:hover, .category a:focus** rule, add **rotate()** as follows:

   ```
   transform: scale(1.05) rotate(25deg);
   ```

   Here, you're tipping the image 25 degrees to the right on the hover. A negative value would tip the image to the left. It is important to note that there is no comma between the scale and rotate transforms. These are space separated values.
3. Save the file, then reload the page in your browser.
4. Hover over one of the images to see a dramatic clockwise rotation that starts from our specified transform-origin. Let's tone that down.
5. Switch back to **main.css** in your code editor.
6. Modify the rotation with the following negative value:

   ```
   transform: scale(1.05) rotate(-2deg);
   ```
7. Save the file, then reload the page in your browser.
8. Hover again to see a more tasteful counterclockwise rotation.
9. Let's try another transform. Switch back to your code editor.
10. Tilt the elements on their horizontal axis by adding the following **skewX()** effect:

    ```
    transform: scale(1.05) skewX(25deg);
    ```
11. Save the file, then reload the page in your browser.

12. Hover over an image. That's interesting, but doesn't work with our design.
13. Switch back to **main.css** in your code editor.
14. Skew the elements vertically (on the y-axis) by changing **X** to **Y** as shown in bold:

    ```
    transform: scale(1.05) skew(25deg);
    ```

15. Save the file, reload the page in your browser, and test out the effect.
16. Switch back to **main.css** in your code editor.
17. You can combine X and Y coordinates by using the **skew()** effect. The X coordinate is listed first and both values are separated by **commas**. Skew both coordinates counterclockwise by adding the following bold negative coordinates:

    ```
    transform: scale(1.05) skew(-5deg, -5deg);
    ```

18. Save the file, then reload the page in your browser.
19. Hover over the images to see the elements do a disco-style "dance". Not what we want, but interesting to see.

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 thanks to our transition. Let's also nudge them slightly upwards on hover.
2. Switch back to **main.css** in your code editor.
3. In the **.category a:hover, .category a:focus** rule, delete the second transform so you end up with:

    ```
    transform: scale(1.05);
    ```

4. 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);
    }
    ```

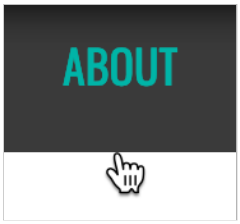
    NOTE: Negative values move an element upward. Positive values do the opposite.

5. Save the file, then reload the page in Chrome.
6. 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.
6. Wrap a **span** tag around the text in each link in the **nav**, as shown in bold:

```
<nav>
   <ul>
      <li><a href="#"><span>Shows</span></a></li>
      <li><a href="#"><span>About</span></a></li>
      <li><a href="#"><span>Contact</span></a></li>
   </ul>
</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. Switch back to **main.css** in your code editor.
13. Below the **nav a** rule, add the following new rule:

```
nav a span {
   display: inline-block;
}
```

14. 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.
15. Return to **main.css** in your code editor.
16. In the **nav a span** rule, add the code shown in bold:

```
nav a span {
   display: inline-block;
   transition: all 300ms;
}
```

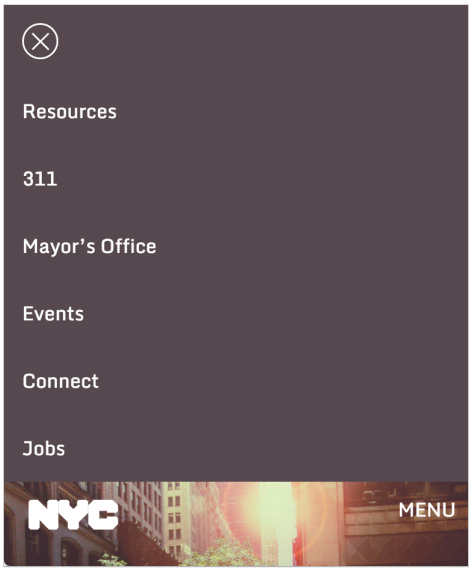
17. Save the file and preview it in Chrome.
18. Hover over the nav links. Notice the text slides up nicely, but the color transition happens after that. We want them to transition at the same time.

    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.
19. In the **nav a span** rule, change **all** to **transform**:

```
nav a span {
   display: inline-block;
   transition: transform 300ms;
}
```

20. Save the file and preview it in Chrome.
21. Hover over the nav links one final time to see the nice smooth transition of movement and color at the same time!

Exercise Preview



Exercise Overview

In a previous exercise we built an off-screen navigation that slid in from the side (overlaying the page content). In this exercise, you'll tweak the HTML/CSS of that page to change the navigation so it slides down from the top of the screen, and pushes the page content down, rather than covering it.

Getting Started

1. This exercise builds on concepts covered in exercise 5D (Off-Screen Side Nav Using Only CSS). We recommend you finish that exercise before starting this one.
2. For this exercise we'll be working with the **Slide-Down Top 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 **Slide-Down Top Nav** folder.
4. Preview **index.html** in a browser.
5. Resize the browser window to a narrow mobile styling, with the **MENU** button at the top right.
6. Click the **MENU** button to notice the nav slides in from the side. The hide/show functionality works, but we want to style the page differently.
7. Leave the browser open at this size, so you can reload it throughout the exercise.
8. In your code editor, open **main.css** from the **css** folder.

Changing the Navigation's Appearance

1. In the **max-width: 700px** media query, find the **nav** rule.
2. Delete the **width** and **left** properties.
3. Change position to **relative** so you end up with the following rule:

```
nav {
   position: relative;
   height: 100%;
   overflow-y: auto;
   z-index: 100;
   transition: all .2s ease;
}
```

   NOTE: We need to specify a relative position, so the z-index will keep the nav above the invisible overlay that lets users click anywhere outside the nav to close it.

4. Save the file and reload your browser.
5. The nav is now at the top instead of the side. That's good, but we want it to be hidden right now. Let's fix that.
6. Return to **main.css** in your code editor.

   In the unchecked state, we want the nav to be hidden. We can achieve this by changing the nav element to have no height. Because we want to use CSS transitions, we can't use the **height** property (browsers cannot animate from height **0** to **auto** using transitions). Luckily we can use **max-height** instead!

7. In the **nav** rule, set **height** to **0** and change it to **max-height**, as shown below in bold:

```
nav {
   position: relative;
   max-height: 0;
   overflow-y: auto;
   z-index: 100;
   transition: all .2s ease;
}
```

8. Now that we've reduced the max-height, we need to hide anything that overflows outside of it. Change the **overflow-y** property as shown below:

```
nav {
   position: relative;
   max-height: 0;
   overflow-y: hidden;
   z-index: 100;
   transition: all .2s ease;
}
```

9. Save the file and reload your browser. Great, the nav is hidden again.
10. Click the **MENU** button to see nothing happens. That's because we need to change what the **:checked** pseudo-class rule does.
11. Switch back to **main.css** in your code editor.
12. Find the **#nav-checkbox:checked ~ nav** rule and replace the **left** property with the code shown below in bold:

```
#nav-checkbox:checked ~ nav {
   max-height: 30em;
}
```

    NOTE: For the transition to work we must specify the height of the element being animated. We are using a **max-height** value that's slightly larger than the element we are transitioning. Through experimenting, we found that 30em works well.

13. Save the file and reload your browser.
14. Click the **MENU** button to see the nav slide down from the top. Neat! There are two things we still need to do:

    - The NYC logo disappears, but we want it to slide down with the rest of the page's content.
    - The menu opens a bit too quickly for our taste. We'd like to slow it down a little.

15. Let's slow down the transition first. Return to **main.css**.
16. In the **nav** rule, edit the transition's timing as shown below in bold:

```
nav {
   position: relative;
   max-height: 0;
   overflow-y: hidden;
   z-index: 100;
   transition: all .5s ease;
}
```

17. Save the file and reload your browser.

18. Click the **MENU** button to see the nav opens a bit slower.

Floating the Logo

The last thing we need to do is fix the positioning of the NYC logo, so it moves down with the rest of the page content.

1. Switch to **index.html** in your code editor.
2. Notice that the logo comes before the nav. We want it below the nav in this layout (so it's with the rest of the page content).
3. Select the logo and cut it.
4. Paste it below the closing **nav** tag, as shown below in bold:

```
</nav>
<img src="img/nyc-logo.svg" class="nyc-logo" alt="NYC Logo">

<h1>New York City</h1>
```

5. Save the file and reload your browser.
6. Click the **MENU** button. Hmm, the logo is still hidden. Let's examine our CSS positioning of this element.
7. Switch back to **main.css**.
8. In the general styles, find the rule for **.nyc-logo**.
9. Notice that the logo has an absolute position. It's currently 15px from the top of the screen. We'll need to remove that positioning and instead make it float left (so it will still be to the left of the **MENU** button when the nav is closed).
10. Edit the **.nyc-logo** rule as shown below in bold. Don't miss adding **margin-** to both top and left!

```
.nyc-logo {
   width: 75px;
   float: left;
   margin-top: 15px;
   margin-left: 20px;
}
```

11. Save the file and reload your browser.
12. Test out the mobile nav to see it's working great.
13. Resize the window larger so you can see the desktop styles (with the navbar across the top).

14. Oh no, the logo isn't in the right position anymore. That's because we edited the general rule.
15. Return to **main.css**.
16. In the **min-width: 701px** media query, after the **nav ul li** rule, add this new rule:

```
.nyc-logo {
   position: absolute;
   top: 15px;
   left: 20px;
   margin: 0;
}
```

17. Save the file and reload your browser to see that the NYC logo is positioned correctly on the desktop.
18. Resize the window down to the mobile styles, and test out your nifty slider nav!
19. After you click the **MENU** button, notice that the **MENU** button disappears. We don't mind this, because users have a close button and they can click anywhere on the page to close the nav. But if you want the MENU button to move down the page like the logo does, continue on to the optional bonus section.

Bonus: Making the MENU Button Slide with the Page

Fixing the **MENU** button is very similar to how we fixed the logo.

1. Switch to **index.html** in your code editor.
2. Around line 14, select the **menu-button** label's line of code.
3. Cut it.
4. Paste it below the **nyc-logo** img, as shown below in bold:

```
<img src="img/nyc-logo.svg" class="nyc-logo" alt="NYC Logo">
<label for="nav-checkbox" class="menu-button">MENU</label>
```

5. Save the file.
6. Return to **main.css**.

7. In the **max-width: 700px** media query, edit the **.menu-button** rule as shown below in bold. Don't miss adding **margin-** to both top and right!

```
.menu-button {
   float: right;
   margin-top: 10px;
   margin-right: 10px;
   color: #fff;
   cursor: pointer;
   -webkit-user-select: none;
   -moz-user-select: none;
   -ms-user-select: none;
   user-select: none;
}
```

8. Save the file, reload your browser.

9. Click the **MENU** button and notice the **MENU** button moved down the page and is still visible. Awesome, you're all done!

Exercise Preview

Exercise Overview

In this exercise, you'll learn how to fix issues that occur when creating columns using CSS floats.

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 **Clearfix** folder located in **Desktop > Class Files > Advanced HTML CSS Class**. You may want to open that folder in your code editor if it allows you to (like Visual Studio Code does).
3. Open **volunteer.html** from the **Clearfix** folder.
4. Preview **volunteer.html** in a browser.
   - There's a form, and below the form are **Volunteer Opportunities**.
   - This layout looks good on smaller and medium screens, but on larger screens we'd like to make these 2 sections into 2 columns.

Creating 2 Columns & Clearing the Float

1. Let's take a look at the markup. Return to **volunteer.html** in your code editor.
   - In the main tag notice there is a div with a class of **form-column** and below that an **aside**.
   - We want these two elements to become columns on larger screens.
2. Open **main.css** from the **css** folder (in the **Clearfix** folder).
3. At the bottom of the file, in the **min-width: 930px** media query, below the **h2** rule, add these new rules:

```
.form-column {
   width: 60%;
   float: left;
}
aside {
   width: 35%;
   float: right;
}
```

4. Save the file, then reload the page in your browser.
   - You should have 2 columns, with the form on the left and **Volunteer Opportunities** on the right.
   - Below the form you'll see the footer text, but that should be on its own line below both columns, not next to the right column.
   - The footer's top gray line is now way too high on the page, behind the tops of the columns we just made. How did it get way up there?

   Floated elements don't take up vertical space in the document flow, so the footer has moved up. We need to push the footer down by having it clear past both floats (so it's not beside them, it will be below them).
5. Return to **main.css** in your code editor.
6. In the **min-width: 930px** media query, below the **aside** rule, add the following new rule:

```
footer {
   clear: both;
}
```

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

8. Scroll down to see the footer (with its top gray line) is back down where it belongs.

   There is still a problem though, it doesn't have space above it. The **main** tag that contains our 2 floated columns has bottom padding, but we're not seeing it. That's because the **main** tag is still collapsed. While clearing did bring the footer down, it didn't actually prevent the collapse of the parent element (**main**) that contains our floated elements (**form-column** and **aside**).

Using Overflow Hidden

Let's see a different way to clear floats.

1. Switch back to **main.css** in your code editor.
2. Delete the **footer** rule you just added (in the **min-width: 930px** media query).
3. In the **main.content-wrapper** rule (in the **min-width: 930px** media query), add the following property:

```
main.content-wrapper {
   padding: 40px 30px;
   overflow: hidden;
}
```

   NOTE: Using the overflow property forces an element to look at its content and adjust its height even if it only contains floated elements!

   Notice this is also the rule that actually adds the bottom padding, so we know it "should" be there if this element does not collapse.

4. Save the file, then reload the page in your browser.
5. Scroll down to see that the footer is down below both columns as it should be, and now it has space above it so it doesn't touch the columns!

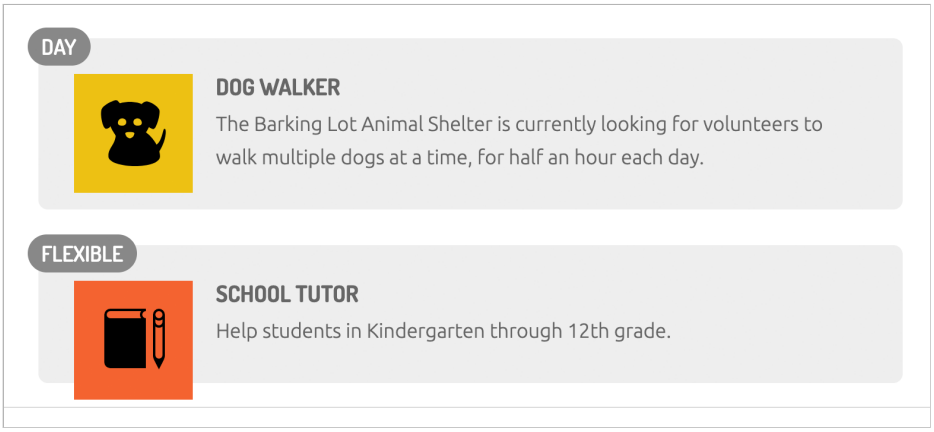
When Overflow Hidden Does Not Work

1. Notice the icons in the aside (right column) are sitting above the text. Let's make the text wrap around the icons by floating the icons. These icons have class of **opportunity-icon** which we can use to target them.
2. Switch back to **main.css** in your code editor.

3. In the general styles (not in a media query), in the **.opportunity-icon** rule add the following code shown in bold:

```
.opportunity-icon {
   width: 100px;
   float: left;
   margin-right: 20px;
}
```

4. Save the file, then reload the page in your browser.
5. In the right column, the text should now be next to, and wrapping around the icons.
6. Resize the browser from wide to narrow, stopping just after the layout changes back to a single column.
7. Scroll down and notice that the **School Tutor** section's icon sticks down too far and is protruding out the bottom of the gray background. It's supposed to have the same amount of padding as the top two sections.



This happens because the icons are floated. Floated elements do not take up vertical space. This means the section container ignores the icons and just resizes to be the height of only the text. There's no element inside each section that we can use to clear the float, so let's try overflow hidden again.

8. Return to **main.css** in your code editor.
9. In the **aside section** rule, add the **overflow** property as follows:

```
aside section {
   CODE OMITTED TO SAVE SPACE
   position: relative;
   overflow: hidden;
}
```

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

11. This did prevent the last section from collapsing (it now has the proper amount of padding below the icon), but now we have a different issue. The **Flexible** label is actually being hidden even though we don't want it to be!

    Fortunately there's a solution to our problem: we can use clearfix.

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

13. Delete **`overflow: hidden;`** from the **aside section** rule.

Using Clearfix

The phrase **clearing floats** is used as a blanket term to refer to any type of layout fix you would have to implement as a result of using the **float** property.

Float was originally used as a method to create text-wrap or runaround for images, but it is also used as a method for positioning elements horizontally when they would normally stack vertically.

Clear is used to ensure that content which sits below a floated element does not wrap up to the side of the floated element. Clear can also be used to fix what is often referred to as element "collapse". Floated content doesn't take up vertical height in the document flow, so something needs to prevent the parent element that holds the floated content from collapsing. Content below the elements that are floated can be set to **clear** and fix any collapse.

If there is no content below the floated element(s), **overflow** can be used to force a parent element to look at its content and adjust its height even if it contains only floated elements. Setting overflow to **hidden** is best because other settings such as auto may cause scrollbars to be rendered. The only issue is that content that doesn't fit will get cropped. If you have elements that use position: absolute in order to pull them up and out of the elements' "box", they'll get cut off.

This last scenario is where we find ourselves now, and where a **clearfix** can save the day. Clearfix is a CSS technique that allows you to fix element collapse without any extra structural HTML markup and without the risk of hiding content via overflow: hidden. Let's investigate.

1. Below the rule for **footer** (in the general styles, not in a media query), add the following new rule:

```
.clearfix::after {
   content: "";
   display: table;
   clear: both;
}
```

   How will this code work? To clear a float, we must apply CSS **clear** to some element that comes after the floated elements. When we don't have an element in our HTML that we can use to clear the floats, we can use the CSS **::after** pseudo-element to create content to use as the clearing element. There's nothing in the quotes for **content**, so the element is essentially nothing (it's invisible). We're only using this element created by CSS to clear the float!

2. Save the file.

   The next step is to add the **clearfix** class to our sections to keep them from collapsing. (Rather than setting the overflow on the parent that's collapsing, you apply the **clearfix** class.)

3. Switch to **volunteer.html** and add the **clearfix** class to the three sections as shown:

```
<section class="clearfix">
   <div class="time-badge">Evening</div>
      CODE OMITTED TO SAVE SPACE
</section>
<section class="clearfix">
   <div class="time-badge">Day</div>
      CODE OMITTED TO SAVE SPACE
</section>
<section class="clearfix">
   <div class="time-badge">Flexible</div>
      CODE OMITTED TO SAVE SPACE
</section>
```

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

5. Now the **Flexible** time badge should not be cut off, and the section should have the correct amount of padding. We have cleared the float, withing hiding anything!

6. Resize the width wider to see two columns. Everything should still look good!

Another Use of Overflow Hidden

In the right column, let's say we wanted the icon to be on the left and the paragraph of text to be on the right, but not wrapping around and flowing below the icon. Let's see a quick way to achieve that.

1. Return to **main.css** in your code editor.
2. The text in these sections is a single paragraph, for which we already have a CSS rule. In the **aside section p** rule, add the following bold code:

```
aside section p {
   font-size: 1rem;
   margin: 0;
   overflow: hidden;
}
```

3. Save the file, then reload the page in your browser.
4. Now the text should no long wrap below the icon!

   It's a bit hard to explain exactly why this works but we'll try. Think about how this section gets laid out. The icon floats to the left and pushes the text to the right. The first lines of text (that are next to the icon) has a certain amount of space but below the image it has more space. But in a way, that extra space is going beyond the bounds of the top line of text. As we've see before, the overflow property makes an element look at overflowing content, and in this case it realizes the content is overflowing and it brings it back in… to be the same width as the first line of text.

Optional Bonus: Using Border-Radius to Round Images

Unrelated to clearing floats, while we're in this page let's take a quick look at rounding corners.

1. Return to **main.css** in your code editor.
2. In the **.opportunity-icon** rule, add the following bold code:

```
.opportunity-icon {
   CODE OMITTED TO SAVE SPACE
   margin-right: 20px;
   border-radius: 8x;
}
```

3. Save the file, reload the page in your browser, and notice the icons in the right column have rounded corners.

   The rounding is nice, but let's make the icons circular. To do this, we need to make the border radius half the width and height of the element.

4. Return to **main.css** in your code editor.
5. Edit the border-radius as follows:

```
.opportunity-icon {
    CODE OMITTED TO SAVE SPACE
    margin-right: 20px;
    border-radius: 50%;
}
```

6. Save the file, then reload the page in your browser. Circular icons, cool!

Check Out

Our Other Workbooks

Web Development Level 1 & 2	Sketch
JavaScript & jQuery	Adobe XD
Flexbox, Grid, & Bootstrap	Adobe After Effects
WordPress	Adobe InDesign
PHP & MySQL	Adobe Illustrator
HTML Email	Adobe Photoshop
Photoshop for Web & UI	Adobe Lightroom
Photoshop Animated GIFs	and more...

nobledesktop.com/books