Web Development with HTML & CSS

Building Websites with HTML & CSS

Learn by doing step by step exercises.

Includes downloadable class files that work on Mac & PC.

EDITION 5.1

noble desktop

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

Publish Date: 12-13-2024

Table of Contents

Table of Contents

SECTION 2

SECTION 3

Table of Contents

Table of Contents

SECTION 5

SECTION 6

BONUS MATERIAL

Table of Contents

Downloading the Class Files

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

1. Navigate to the **Desktop**.
2. Create a **new folder** called **Class Files** (this is where you'll put the files after they have been downloaded).
3. Go to **nobledesktop.com/download**
4. Enter the code **wd-2407-03**
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 **Web Dev 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.

A basic website is essentially a collection of files in a folder. People access the files when they are hosted on a web server, accessible via the Internet. A website's address (for example google.com) is known as a URL (uniform resource locator).

Webpages are delivered via HTTP (Hypertext Transfer Protocol) or HTTPS (HTTP Secure), the latter of which uses encryption and provides security for the user. The web browser renders the page content according to its HTML markup instructions.

What Is HTML?

HTML stands for **Hypertext Markup Language**. It's a standardized system for tagging content in webpages. HTML provides a way to structure text semantically into paragraphs, headings, lists, links, etc. It also allows images and objects like form elements to be represented on the page. Webpages can link from one to another.

HTML allows computers of various platforms (Mac, Windows, iOS, Android, etc.) to view information in essentially the same way.

Most users will see the webpage as you intended, but it can appear differently depending on the user's browser or operating system. For example, one person may see the text in Times, while another sees Helvetica (because they uninstalled the Times font). While the webpage may not appear exactly the same to every user, the important thing is that all users have an acceptable experience and can access the content. Blind people can even have webpages read to them!

What Is CSS?

CSS stands for **Cascading Style Sheets**. It's a style language used to define the colors, fonts, spacing, layout, and more of elements in an HTML file. HTML files markup (label) content and CSS formats (styles) that content.

HTML and CSS are continually evolving languages, so webpages viewed in older browsers may not not look/work exactly the same as newer browsers. If only a few people still use the older browser (or it's not a major change) it might not be a problem. Otherwise you could wait until more people upgrade their browser to support the feature/code you want to use.

The Structure of an HTML Tag

Information in HTML is surrounded by tags, which are are wrapped by < >
For example, here's how to mark up (code) a paragraph:

```
<p>This is a paragraph</p>
```

Notice the paragraph is followed by a closing tag </p>

Most HTML tags require an ending or closing tag. If you have problems, you may have forgotten to end the tag. Some tags may work even if you don't end them. For example, the tag indicates a new paragraph, so it often works without a closing . It's a best practice to close tags to ensure they work properly.

Some tags do not have a closing tag, such as <img> for an image.

HTML tags are not case sensitive, but some things such as filenames and CSS style names are. It's a best practice to write all tags in lowercase.

File Naming Conventions

- HTML files end with the extension **.html**
- CSS files end with the extension **.css**
- Do not use spaces or special characters. Use hyphens instead of spaces.
- Use only lowercase letters when naming files (some servers are case sensitive).
- The homepage of a website is named **index.html**
- Use descriptive file names that tell what a page is about. For example: **about.html** and **contact.html** rather than **page1.html** and **page2.html**. Descriptive file names are good for SEO (Search Engine Optimization) because they provide keywords that help the page rank better on search engines like Google.

Choosing a Code Editor to Work In

Code editors are specialized text editors which offer code hints and more. Two popular choices are **Visual Studio Code** and **Sublime Text**. You can write code in any editor, 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, at the bottom left of the window, click the Gear  and from the menu that appears choose **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, at the bottom left of the window, click the Gear and from the menu that appears choose **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.

Downloading the Latest Web Browsers

Browsers can handle code slightly differently, so as a web developer it's important to have all the current popular browsers installed for testing.

Chrome (Mac & Windows)

- Get it free at **google.com/chrome**

Firefox (Mac & Windows)

- Get it free at **firefox.com**

Safari (Mac Only)

- To get the most recent version of Safari, you may have to update your macOS.

Microsoft Edge (Mac & Windows)

The latest versions of Microsoft Edge are based on Chrome's rendering engine.

- Mac Users: Get it free at **microsoft.com/edge**
- Windows Users: Microsoft Edge comes pre-installed on Windows 10 (it replaced Internet Explorer).

Exercise Preview

Exercise Overview

In this exercise you'll learn how to write HTML to create a webpage and format headings, paragraphs, and lists.

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. All the files for a website are stored in a folder. We'll be working with files in a folder named **News Website**. Open the folder as follows (this should work in most code editors such as Visual Studio Code):
   - Go to **File > Open Folder**.
   - Navigate to **Class Files > Web Dev Class > News Website** and hit **Open** (Mac) or **Select Folder** (Windows).
   - If you get a message about trusting the author, check on **Trust the authors of all files in the parent folder** and click **Yes, I trust the authors**.
3. On the left you should see a panel showing files in the folder you just opened. Click on **microsoft.html** to open it.

4. We've typed out some text, but it doesn't have any HTML tags yet. Add the required tags shown below. The code you need to type is highlighted in bold throughout this book.

   NOTE: As you begin typing a tag, your code editor may display a list of suggested tags. For now, we recommend ignoring those code hints. Also, when you type the **</** part of an ending tag, your code editor may automatically finish the closing tag for you. This is a nice time-saver, but be sure to double-check your code.

```
<html>
<head>
   <title>Microsoft Patents Ones and Zeroes - The Onion</title>
</head>
<body>
   Microsoft Patents Ones and Zeroes
```

5. Scroll to the bottom and add the following closing tags highlighted in bold:

```
   Wall Street reacts to MS Patent News. Read more...
   This report was written by The Onion
</body>
</html>
```

   Let's break this code down:

   - HTML tag names are enclosed in less-than (<) and greater-than (>) signs.
   - Each tag wraps the content it describes, so there is a start and an end tag.
   - The end tag has a slash (/) before the tag name.

   Notice that the **<html>** tag wraps around the entire document. Inside that, there are two sections: the **<head>** and the **<body>**. The **head** contains information about the document, such as its title and other data that the visitor to the site will not interact with, whereas the **body** contains the document's actual content that a visitor will see and interact with.

   The **<title>** should be a concise description of this page's content. That is important for Search Engine Optimization (SEO), as the title plays an important role in most search engines' ranking. Additionally, it's often the text search engines display in their search results. Titles appear in the current browser tab, as well as a browser's bookmarks (if a user bookmarks the page). Because the **title** is inside the **head** tag, we indented it to make it easier to see the structure.

6. Let's see what this page looks like in a browser. Note that when you preview this file, the text is going to be one big blob of text because we haven't formatted it yet. Before we preview, we need to save the file. Hit **Cmd–S** (Mac) or **Ctrl–S** (Windows).

7. Keep this file open in your code editor, but switch to the Desktop (the **Finder** on Mac or **Windows Explorer** on Windows).

8. Go into the **Class Files** folder, then **Web Dev Class** folder, then **News Website**.
9. **Ctrl–click** (Mac) or **Right–click** (Windows) on **microsoft.html**, go to **Open with** and select your favorite browser.
10. In the browser's tab (at the top of the window), you should see the text you put the **<title>** tag. (In Safari you won't see the title unless the tab bar is visible.)
11. Next, take a look at the page. The actual content of the page—which is what is wrapped inside the **<body>** tag—looks like one long paragraph of text. That's because line breaks in the code are ignored by web browsers.

    NOTE: We recommend leaving **microsoft.html** open in the browser as you work, so you can reload the page to see the changes as you make them in the code.
12. Return to **microsoft.html** in your code editor.

Headings

Headings help organize content, so visitors can quickly skim a page. They also make the page more accessible (visually impaired visitors using a screen reader can hear the headings, and jump between them using keystrokes). Headings also help search engines (like Google) to better understand what the page's content is about. There are six levels of headings from **h1** (the main topic) to **h6** (the most nested subtopic).

1. Our page has a single heading, with no subtopics. In your code editor, at the top of the **body** section, wrap the heading text in an **h1** tag as shown below in bold:

```
<body>
   <h1>Microsoft Patents Ones and Zeroes</h1>
```

2. Save the file.
3. Return to the browser and click the **Reload** button to refresh the browser window with the new code.
4. Notice that the heading is bold. More important headings would also be larger than less important ones. These defaults, along with the space around the heading, can be modified using Cascading Style Sheets (CSS). For now, we are just focusing on the markup, so we won't change their appearance.

Paragraphs

Even though there appear to be paragraphs in the provided text, the browser doesn't know where the paragraphs start and end. We have to code that.

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

2. For the first paragraph, add the following opening and closing tags shown in bold:

```
<h1>Microsoft Patents Ones and Zeroes</h1>

<p>REDMOND, WA: In what CEO Bill Gates called an unfortunate but necessary
step to protect our intellectual property from theft and exploitation by
competitors, the Microsoft Corporation patented the numbers one and zero
Monday.</p>
```

3. Wrap the remaining paragraphs, as shown below:

```
<p>With the patent, Microsoft's rivals are prohibited from manufacturing or
selling products containing zeroes and ones unless a royalty fee of 10 cents
per digit used is paid to the software giant.</p>

<p>Some other patents that Microsoft owns:</p>

   Windows
   Microsoft Office
   Xbox

<p>Wall Street reacts to MS Patent News. Read more...</p>
<p>This report was written by The Onion</p>
```

4. Save the file.

5. Return to the browser and click the **Reload** button. You should now see multiple paragraphs that are separated by some vertical space.

Lists

To create a bulleted list, we must use two tags: **<ul>** and **<li>**:

- The **<ul>** tag creates an **unordered list**, which is a bulleted list. There is also an **<ol>** tag for an ordered list, which uses numbers or letters (1, 2, 3 or A, B, C).
- The **<li>** tag is wrapped around the contents of each **l**ist **i**tem.

1. Return to **microsoft.html** in your code editor. Mark up a bulleted list of Microsoft's patents by adding the following code highlighted in bold:

```
<p>Some other patents that Microsoft owns:</p>
<ul>
   <li>Windows</li>
   <li>Microsoft Office</li>
   <li>Xbox</li>
</ul>

<p>Wall Street reacts to MS Patent News. Read more...</p>
```

2. Save the file.
3. Return to the browser and click the **Reload** button to refresh the browser to see a bulleted list with three items.

   Congratulations, you've made a webpage! Leave the file open so you can use it in the next exercise.

Exercise Preview

REDMOND, WA: In what CEO Bill Gates called an unfortunate but necessary step to protect our intellectual property from theft and exploitation by competitors, the Microsoft Corporation patented the numbers one and zero Monday.

With the patent, Microsoft's rivals are prohibited from manufacturing or selling products containing zeroes and ones unless a royalty fee of 10 cents per digit used is paid to the software giant.

Some other patents that Microsoft owns:

- Windows
- Microsoft Office
- Xbox

Wall Street reacts to MS Patent News. *Read more*...

Exercise Overview

In this exercise you'll learn how to bold and italicize text, as well as some important tags and attributes that every webpage needs.

1. If you completed the previous exercise, **microsoft.html** should still be open in your code editor, and you can skip the following sidebar. If you closed **microsoft.html**, re-open it now (from the **News Website** folder). We recommend you finish the previous exercise (1A) before starting this one. If you haven't finished it, do the following sidebar.

If You Did Not Do the Previous Exercise (1A)

1. Close any files you may have open.
2. In your code editor, open **microsoft-ready-for-doctype.html** from the **News Website** folder.
3. Do a **File > Save As** and save the file as **microsoft.html**, replacing the older version in your folder.

Strong & Em (Bold & Italic)

The **`<strong>`** tag is rendered as **bold** and **`<em>`** (short for emphasis) is rendered as *italic* in a browser. These tags may change the speaking style of a screen reader, which is an app used by people who are blind or visually impaired.

1. Return to **microsoft.html** in your code editor and add the following tags to make some text bold:

```
<p><strong>REDMOND, WA:</strong> In what CEO Bill Gates called an unfortunate
but necessary step to protect our intellectual property from theft and
exploitation by competitors, the Microsoft Corporation patented the numbers
one and zero Monday.</p>
```

2. Near the bottom, add the following tags to italicize some text:

```
<p>Wall Street reacts to MS Patent News. <em>Read more...</em></p>
<p>This report was written by The Onion</p>
```

3. Save the file.
4. Return to the browser and click the **Reload** button to see the bold and italic text.

Adding Attributes

1. Return to **microsoft.html** in your code editor.
2. Add the following attribute (highlighted in bold) to the **html** tag:

```
<html lang="en">
```

Take a moment to review the code and check for typos. Notice that **lang="en"** is written after the tag name but before the greater-than (>) sign.

The **lang** attribute declares the language of a page (or part of a page) and **en** is the international standard code for English. This is helpful for accessibility and Search Engine Optimization (SEO).

Attributes

Many HTML tags can be enhanced with **attributes**, modifiers of HTML elements. Most attributes have a name and a value. Attributes are only added to a start tag and are written like this `<tag attribute="value">`

Meta & Doctype Tags

1. Add the following new line of code highlighted in bold:

```
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Microsoft Patents Ones and Zeroes - The Onion</title>
</head>
```

Meta tags can do various things, but this one says that special characters are encoded as Unicode (UTF-8). This means that special characters (like ©, ™, etc.) can be typed into the code and they'll display properly across platforms and devices.

This particular meta tag must occur within the first 512 bytes of the file, so it is considered good practice for this to be the first child of the `<head>` element.

2. Place the cursor at the very beginning of the code—directly before the **html** tag—and hit **Return** (Mac) or **Enter** (Windows) to get a new line on top.

3. Now add the following code highlighted in bold:

```
<!DOCTYPE html>
<html lang="en">
```

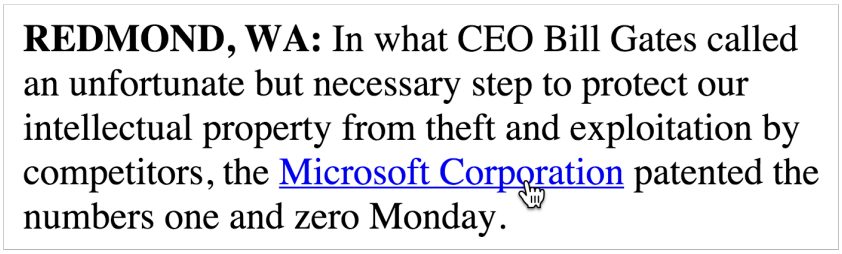
This **doctype** is for HTML5 and beyond. It tells the browser to render the page according to the latest HTML and CSS specifications (which is called "standards mode"). Using older doctypes (or omitting a doctype) can render the page in "quirks mode", which emulates the nonstandard behavior of old browsers.

4. Notice the doctype and meta tags do not have an ending tag (unlike most of the tags we've used so far). That's because they do not wrap around content.

5. Save the file.

Congratulations, you've finished your first webpage! Leave the file open so you can use it in the next exercise.

Exercise Preview



Exercise Overview

What would a webpage be without links? In this exercise, you'll code a few links to learn how it's done for both external websites and other pages on your site.

1. If you completed the previous exercise, **microsoft.html** should still be open in your code editor, and you can skip the following sidebar. If you closed **microsoft.html**, re-open it now (from the **News Website** folder). We recommend you finish the previous exercise (1B) before starting this one. If you haven't finished it, do the following sidebar.

If You Did Not Do the Previous Exercise (1B)

1. Close any files you may have open.
2. In your code editor, open **microsoft-ready-for-links.html** from the **News Website** folder.
3. Do a **File > Save As** and save the file as **microsoft.html**, replacing the older version in your folder.

The Anchor Tag & HREF Attribute

1. In the first paragraph, wrap an **<a>** tag (which stands for anchor) around **Microsoft Corporation**, as shown below in bold:

```
<p><strong>REDMOND, WA:</strong> In what CEO Bill Gates called an unfortunate
but necessary step to protect our intellectual property from theft and
exploitation by competitors, the <a href="https://www.microsoft.com">Microsoft
Corporation</a> patented the numbers one and zero Monday.</p>
```

NOTE: The anchor tag wouldn't do anything without the **href** attribute. Href stands for hyperlink reference. Its value is equal to the URL (uniform resource locator) of the page you're linking to.

2. Take a moment to review the code and check for typos. Remember that attributes are only added to a start tag and are written like this `<tag attribute="value">`
3. Save the file.
4. Preview **microsoft.html** in a browser.

> **Browser Preview Shortcuts**
>
> If you're using Visual Studio Code with the **open in browser** extension installed, hit **Option–B** (Mac) or **Alt–B** (Windows) to open the saved HTML document in your default browser. If asked what browser you want to open it with, ideally you should choose Google Chrome (which is the most widely used browser).

5. Click the link to make sure it works. Links that include the entire address (including the **`http://www`**) are called **absolute** links.
6. Return to **microsoft.html** in your code editor. Create another link, this time to The Onion's website (in the last paragraph, just before the end **body** tag):

```
   <p>Wall Street reacts to MS Patent News. <em>Read more...</em></p>
   <p>This report was written by <a href="https://www.theonion.com">
The Onion</a></p>
</body>
```

7. Save the file.
8. Preview **microsoft.html** in a browser and test the new link.

   We just made two links to outside websites, but what about a link to a page in this site? We made another page (called **wall-street.html**) which is in the same folder as the page you're currently editing.

9. Return to **microsoft.html** in your code editor and below the bulleted list, create that link by adding the code shown in bold:

```
   <p>Wall Street reacts to MS Patent News. <a href="wall-
street.html"><em>Read more...</em></a></p>
   <p>This report was written by <a href="https://www.theonion.com">The
Onion</a></p>
</body>
```

   Links like this, which don't include the full address, are called **relative** links. The link is relative to the current .html file. In this case, the link will look for a page called **wall-street.html** in the same folder as this file (**microsoft.html**).

10. Save the file.

11. Preview **microsoft.html** in a browser and test out the links. If the browser is still open to the page, you can hit the **Reload** button.

Opening a Link in a New Browser Tab/Window

We can make a link open in a new browser tab or window (depending on the user's preference) using the **target** attribute.

1. Return to **microsoft.html** in your code editor.
2. In the first paragraph, add the **target** attribute to the **microsoft.com** link as shown below. NOTE: Attributes must be separated by a space character.

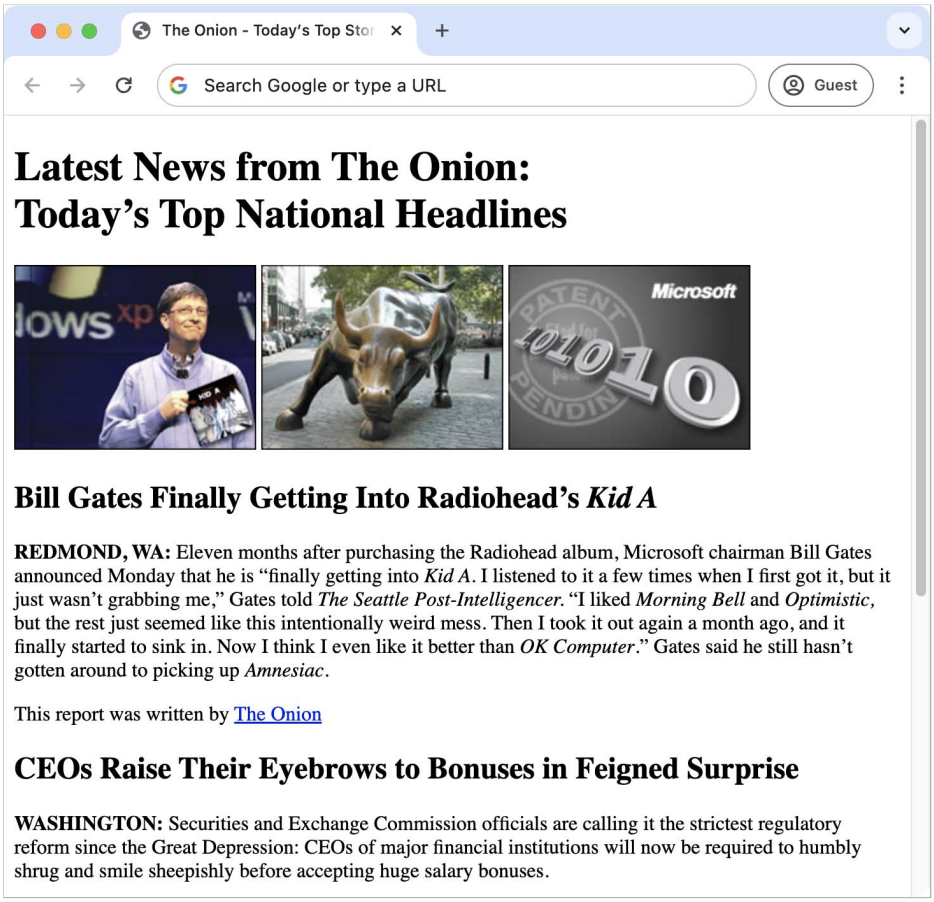
```
<p><strong>REDMOND, WA:</strong> In what CEO Bill Gates called an unfortunate
but necessary step to protect our intellectual property from theft and
exploitation by competitors, the <a href="https://www.microsoft.com"
target="_blank">Microsoft Corporation</a> patented the numbers one and zero
Monday.</p>
```

3. In the last paragraph, add the **target** attribute to the **theonion.com** link as shown below:

```
<p>This report was written by <a href="https://www.theonion.com"
target="_blank">The Onion</a></p>
```

4. Save the file.
5. Preview **microsoft.html** in a browser. Click the links for **Microsoft Corporation** and **The Onion** and those pages should open in a new browser tab or window!
6. Return to your code editor and close any files you have open. We'll be moving on to a new file in the next exercise.

Exercise Preview



Exercise Overview

In this exercise, you'll learn how to add images to a page, and include alternative text for accessibility. You will also learn about two other tags: one for creating line breaks and another for creating the appearance of content division. These three tags have something in common… none of them wrap around content.

1. In your code editor, go to **File > Open**.
2. Navigate to **Desktop > Class Files > Web Dev Class > News Website**.
3. Double–click on **index.html** to open it.

   NOTE: **index.html** is a special filename reserved for the first page (the homepage) of a website. When you go to a .com URL in a browser, the **index.html** is the first page that will be displayed.

4. Preview in a browser to get a feeling for the page. TIP: If you're using Visual Studio Code with the **open in browser** extension installed, hit **Option–B** (Mac) or **Alt–B** (Windows) and your saved HTML document will open in your default browser.

Adding a Line Break

The main heading at the top (which is an h1) is a bit long. A **break** tag `<br>` would help to make it more legible.

1. Return to **index.html** in your code editor. Add a break tag to push Today's Top National Headlines to the next line:

```
<body>
   <h1>
      Latest News from The Onion:<br>
      Today's Top National Headlines
   </h1>
```

   Like the **doctype** and **meta** tags, the **
** tag does not have a closing tag because it has no content inside of it. It simply performs its own function.

2. Save the file.
3. Return to the browser and reload the page to see the line break in the top heading.

Adding Image Files

Images are inserted into the HTML document with a single tag (they have no end tag). There are two main graphic file formats we'll use in this book: JPEG and PNG. Please see the **Graphic File Formats** reference in the front of the workbook for more information on file formats.

1. Return to **index.html** in your code editor.
2. To add an image below the main heading, add the following line of code highlighted in bold. It may not look like a single line in the book, but be sure to enter it on one line in your code editor.

```
<body>
   <h1>
      Latest News from The Onion:<br>
      Today's Top National Headlines
   </h1>
   <img src="images/newsthumb-bill-gates.jpg" height="145" width="190"
alt="Bill Gates, Radiohead Fan">

   <h2>Bill Gates Finally Getting Into Radiohead's <em>Kid A</em></h2>
```

3. Take a moment to review the code you just typed. The **<img>** tag requires a **src** attribute (**src** is an abbreviation for **source**) to call the appropriate image file. Images are typically stored in a subfolder of the website, which we've named **images**. There's nothing special about that name (it could be named **img** if you want a shorter name), but we do have to know the folder name to link to images inside it.

   Notice that the code includes a **width** and a **height** attribute. Specifying the dimensions of an image can slightly speed up the rendering of the page.

   Alt Text

   The img tag's **alt** attribute (commonly referred to as **alt text**) is important for accessibility and Search Engine Optimization (SEO). It is a brief text description of a graphic that is used by screen readers, search engines, and is displayed if the graphic does not load.

   It's best practice to add alt text to all graphics. The only time no alt text is needed is if the graphic is purely decorative. In those rare cases, add an empty alt attribute (**alt=""**) or use CSS techniques so the graphics can be ignored by screen readers.

4. Save the file.

5. Return to the browser and reload the page to see the image.

   NOTE: Image files are not actually embedded into an HTML page. The image tag is a placeholder for the linked source file, so images must be uploaded (along with the HTML pages) to your remote web server in order for visitors to see them.

6. Return to your code editor, and add another image below the previous one by typing the following code shown in bold:

```
<body>
   <h1>
      Latest News from The Onion:<br>
      Today's Top National Headlines
   </h1>
   <img src="images/newsthumb-bill-gates.jpg" height="145" width="190"
alt="Bill Gates, Radiohead Fan">
   <img src="images/newsthumb-wall-street-bull.jpg" height="145" width="190"
alt="Charging Bull">

   <h2>Bill Gates Finally Getting Into Radiohead's <em>Kid A</em></h2>
```

7. Save the file.

8. Return to the browser and reload the page to see that the images sit in a line, rather than stacking on top of each other.

> **Img Is an Inline Element**
>
> Images, text, and anchor tags are considered **inline** elements because each one goes next to the other to form a line of content.
>
> If you want to force inline elements (such as images) to stack on top of each other, you can put them in a container that is too narrow for them to fit next to each other. You could also change their CSS display property.

9. Return to the code and add a third image below the second one you just added:

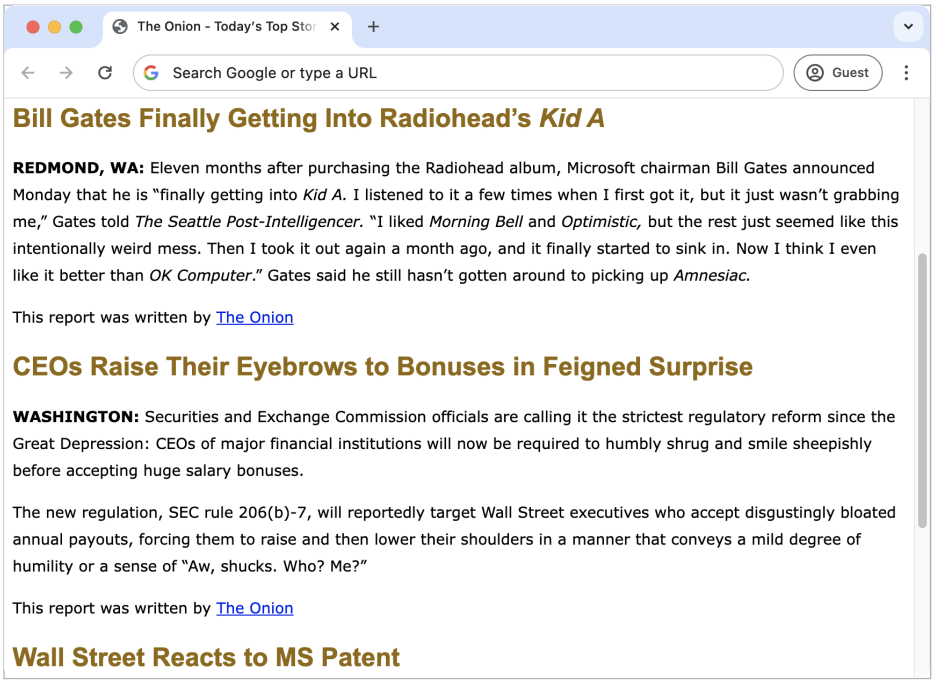
```
<img src="images/newsthumb-bill-gates.jpg" height="145" width="190" alt="Bill
Gates, Radiohead Fan">
<img src="images/newsthumb-wall-street-bull.jpg" height="145" width="190"
alt="Charging Bull">
<img src="images/newsthumb-patent.jpg" height="145" width="190" alt="Microsoft
Patent Illustration">

<h2>Bill Gates Finally Getting Into Radiohead's <em>Kid A</em></h2>
```

10. Save the file, return to the browser, and reload the page to preview the new image.
11. You can leave the file open in the browser and the code editor so you can use it in the next exercise.

> **Graphic File Formats: JPEG, PNG, etc.**
>
> To learn more about which file formats to use for graphics, read the **Graphic File Formats for the Web** reference at the back of this book.

Exercise Preview



Exercise Overview

In this exercise, you'll style text (font, color, size, etc) using Cascading Style Sheets (CSS). HTML defines the type of content: heading, paragraph, list, etc. CSS tells the browser how to style that content.

1. If you completed the previous exercise, **index.html** should still be open in your code editor, and you can skip the following sidebar. If you closed **index.html**, re-open it now (from the **News Website** folder). We recommend you finish the previous exercise (1D) before starting this one. If you haven't finished it, do the following sidebar.

If You Did Not Do the Previous Exercise (1D)

1. Close any files you may have open.
2. In your code editor, open **index-ready-for-styles.html** from the **News Website** folder.
3. Do a **File > Save As** and save the file as **index.html**, replacing the older version in your folder.

2. Preview the file in a browser so you can see how it looks. TIP: If you're using Visual Studio Code with the **open in browser** extension installed, hit **Option–B** (Mac) or **Alt–B** (Windows) and your saved HTML document will open in your default browser.
3. Notice the main heading (h1), three images, and then three stories. Each story has a subheading (h2) and some paragraphs of text.

Tag Selectors: A Way to Set "Default" Appearance

1. Return to **index.html** in your code editor.
2. CSS is a list of stylistic rules for the browser to follow. Because CSS works behind the scenes, it is written in the **head** section rather than in the **body**. Add the following bold code in the **head**, just below the **title** tag:

```
   <title>The Onion - Today's Top Stories</title>
   <style>

   </style>
</head>
```

3. A CSS selector tells the browser where to apply a style. The first type of CSS selector we'll use is called a **tag selector**. It tells the browser to find all instances of a particular HTML tag and style them. Add the following bold **h1** tag selector inside the `<style>` tag:

```
<style>
   h1 {

   }
</style>
```

   This rule will target all the h1's on the page.

4. Inside the h1 tag selector add the following bold code:

```
h1 {
   font-family: Arial, Helvetica, sans-serif;
   font-size: 35px;
   color: #006b3a;
}
```

   A CSS **rule** is the selector (in this case **h1**) through the closing curly brace. Inside the curly braces `{}` we add **property declarations**. For each predefined CSS property (font-size, color, etc.) we set a value (35px, #006b3a, etc.).

   We recommend putting each property on its own indented line, which your code editor may do automatically. It's not required, but makes the code easier to read.

5. Save the file.

6. Return to the browser and reload the page to see that the heading at the top has changed. Awesome!

> **The Font-Family Property**
>
> The **font-family** property is a wish list. The first font will be applied if it's available on the user's computer. The second, third, or any additional fonts will only be used if the preceding font is not available. This list is called the font stack.

> **Hexadecimal Color Codes**
>
> Web colors are commonly coded as a 6-digit hexadecimal value that represents RGB color values: the first 2 digits are red, the next 2 green, and the last 2 are blue. Hexadecimal values must start with a # sign: **`#00ff33`**
>
> The letters in hex values are not case sensitive—they can be written in either upper or lowercase.

7. Return to **index.html** in your code editor.
8. Inside the **style** tags, below the **h1** style, add a rule for the subheadings (h2 tags). Type only the new code that is highlighted in bold:

```
   h1 {
      font-family: Arial, Helvetica, sans-serif;
      font-size: 35px;
      color: #006b3a;
   }
   h2 {
      font-family: Arial, Helvetica, sans-serif;
      font-size: 24px;
      color: #8f6800;
   }
</style>
```

9. Save the file.
10. Return to the browser and reload the page again to see your new heading styles. You have styled all elements in the page that have been tagged as h1 or h2.
11. Let's add another rule. Return to your code editor.

12. Below the **h2** rule, add a rule for all paragraphs (type only the new rule that is highlighted in bold):

```
   h2 {
      font-family: Arial, Helvetica, sans-serif;
      font-size: 24px;
      color: #8f6800;
   }
   p {
      font-family: Verdana, sans-serif;
      font-size: 14px;
      line-height: 25px;
   }
</style>
```

NOTE: We use line-height to adjust the space between lines of text (it's called leading in print design). Line-height defines how tall a line of text is. Characters are vertically centered within this space, so increasing line-height adds space above and below the characters. Readability can be dramatically improved by increasing line-height, particularly if the lines of text are long.

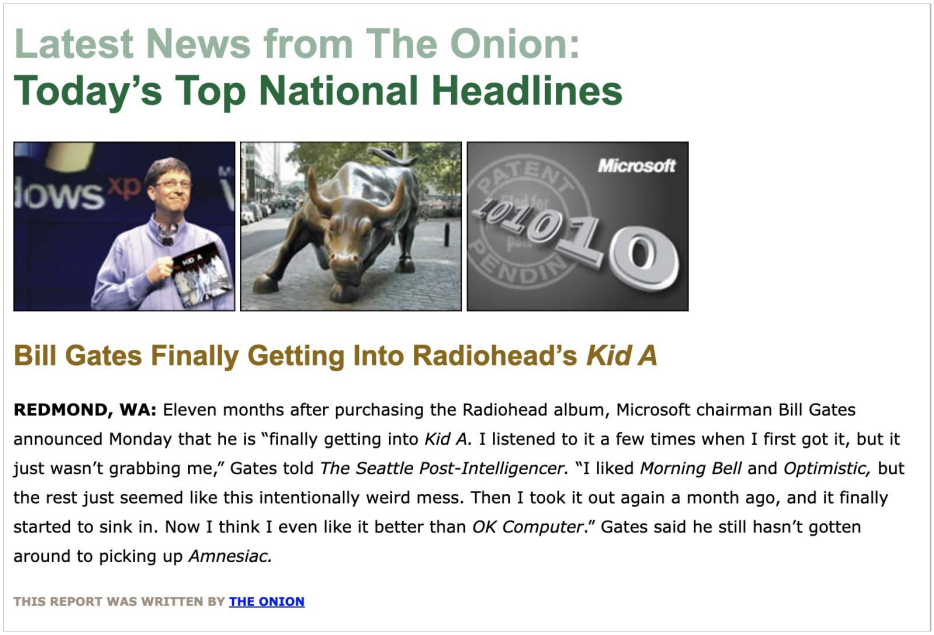
13. Save the file.

14. Return to the browser and reload the page. What a difference!

Units of Measure for Type in CSS

We measure screens in **pixels**, so pixels are commonly used to size type in webpages. There are other types of measurement, such as **ems** and **rems**. They are proportional, and require a bit of math to figure out the type size. We're starting with pixels because they are easier.

Exercise Preview



Exercise Overview

In the previous exercise you learned that CSS tag selectors are a quick way to style all instances of a tag. But what if you want a one paragraph to look different from the rest? In this exercise you'll learn how to use the **class selector** to override a tag selector anywhere you like.

1. If you completed the previous exercise, **index.html** should still be open in your code editor, and you can skip the following sidebar. If you closed **index.html**, re-open it now (from the **News Website** folder). We recommend you finish the previous exercises (1D–2A) before starting this one. If you haven't finished it, do the following sidebar.

If You Did Not Do the Previous Exercises (1D–2A)

1. Close any files you may have open.
2. In your code editor, open **index-ready-for-classes.html** from the **News Website** folder.
3. Do a **File > Save As** and save the file as **index.html**, replacing the older version in your folder.

2. Preview the file in a browser so you can see how it looks.

   TIP: If you're using Visual Studio Code with the **open in browser** extension installed, hit **Option–B** (Mac) or **Alt–B** (Windows) and your HTML file will open in your default browser.

Class Selectors: Making Exceptions to the Defaults

We want to style some paragraphs differently than the rest. To do this we must give them a name (called a **class**) that we'll use to target them for styling. We can add a **class** attribute to any HTML tag.

1. There are three paragraphs that say **This report was written by**… Find the first one (around line 36).

2. On the opening **p** tag, add **`class="author"`** as shown below in bold:

   ```
   <p class="author">This report was written by <a href="https://
   www.theonion.com" target="_blank">The Onion</a></p>
   ```

   NOTE: There are no predefined class names. You choose the name, ideally a name that describes what the element is (such as **author**), rather than how it will look (such as **uppercase**). We don't want to have to change the class name if we later decide to make it look different.

3. Now that we have a way to refer to this specific paragraph, we can define a CSS rule for how it should look. In the **style** tag, under the **p** style, add the following bold code. Be sure to type the period (.) before **author**, which is what tells the browser it's a **class** selector!

   ```
      p {
         font-family: Verdana, sans-serif;
         font-size: 14px;
         line-height: 25px;
      }
      .author {
         font-size: 10px;
         text-transform: uppercase;
         font-weight: bold;
         color: #a18f81;
      }
   </style>
   ```

4. Save the file.

5. Return to the browser and reload the page. Notice that only the first instance of **THIS REPORT WAS WRITTEN BY** should be uppercase and sand brown.

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

7. Classes can be reused as many times as needed. Find the second **This report was written**… paragraph (around line 48) and add **class="author"** as shown below in bold:

```
<p>The new regulation, SEC rule 206(b)-7, will reportedly target Wall Street
executives who accept disgustingly bloated annual payouts, forcing them to
raise and then lower their shoulders in a manner that conveys a mild degree of
humility or a sense of "Aw, shucks. Who? Me?"</p>
<p class="author">This report was written by <a href="https://
www.theonion.com" target="_blank">The Onion</a></p>
```

8. One more to go. Near the bottom, find the last **This report was written**… paragraph and add **class="author"** as shown below in bold:

```
   <p class="author">This report was inspired by, but not written by <a
href="https://www.theonion.com" target="_blank">The Onion</a></p>
</body>
</html>
```

9. Save the file.

10. Return to the browser and reload the page to see that the author style applies to all three author paragraphs.

The Span Tag

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

2. We want to make the first part of the h1 tag a bit lighter. We need an HTML tag to add the **class** attribute, so we'll need to wrap a **** tag around the words we want to group together.

   Find the **h1** near the start of the **body** section, and add the following bold code:

```
<h1>
   <span class="muted">Latest News from The Onion:</span><br>
   Today's Top National Headlines
</h1>
```

   NOTE: Class names are case sensitive. They cannot contain spaces, so use hyphens or underscores instead. They cannot start with a number, and you should avoid using special characters.

3. Now we can make the CSS rule to style our new class. Under the **.author** style, add the following bold code:

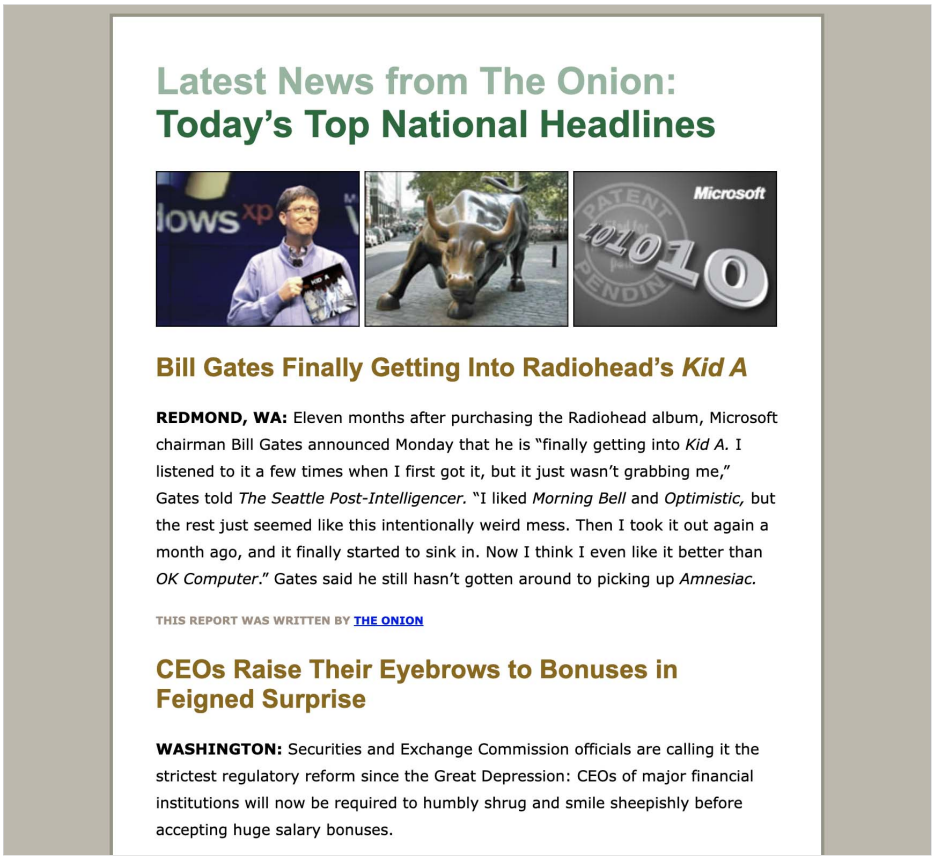
```
   .author {
      font-size: 10px;
      text-transform: uppercase;
      font-weight: bold;
      color:#a18f81;
   }
   .muted {
      opacity: .5;
   }
</style>
```

   NOTE: Opacity takes a value between 0 and 1 (1 is 100% visible, and 0 is invisible). A decimal such as .5 or 0.5 (the preceding 0 is optional) would be 50% transparent.

4. Save the file.

5. Return to the browser and reload the page. You should see that **Latest News from The Onion:** is lighter (because it's partially transparent and showing through to the white background behind the text).

Exercise Preview



Exercise Overview

In this exercise you'll take more control over the layout of the page, using a **div** tag (div is short for division). Wrapping content in div tags lets us create sections of content that are grouped together and can be styled with CSS.

Div versus Span

In the previous exercise we used a **span** tag to do something similar, but here's the main difference:

- **Span** tags are **inline** elements, meaning multiple span tags go next to each other in a line.
- **Div** tags are **block** elements, meaning multiple div tags stack on top of each other.

1. If you completed the previous exercise, **index.html** should still be open in your code editor, and you can skip the following sidebar. If you closed **index.html**, re-open it now (from the **News Website** folder). We recommend you finish the previous exercises (1D–2B) before starting this one. If you haven't finished them, do the following sidebar.

If You Did Not Do the Previous Exercises (1D–2B)

1. Close any files you may have open.
2. In your code editor, open **index-ready-for-divs.html** from the **News Website** folder.
3. Do a **File > Save As** and save the file as **index.html**, replacing the older version in your folder.

Wrapping Content in a Div & Setting a Page Width

1. Preview the file in a browser so you can see how it looks. Take a moment to click and drag the edge of the browser window in and out to resize the window. Notice how all the content, including the images, wrap to the edge of the browser window. This isn't all that problematic for the more narrow window size, but if you make the window rather wide, the content stretches out to a point where it is incredibly hard to read.

   We can use the **`<div>`** tag to get more control over the layout.

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

3. Let's start to wrap a **div** tag around all the content in the page. Type the following opening tag (highlighted in bold) just below the opening **body** tag:

```
</head>
<body>
   <div>
   <h1>
```

4. As shown below in bold, close the **div** tag just above the closing **body** tag (at the bottom of the file):

```
   <p class="author">This report was inspired by, but not written by <a
href="https://www.theonion.com" target="_blank">The Onion</a></p>
   </div>
</body>
</html>
```

5. We recommend indenting the lines in between the opening and closing **div** tags. It's not required for the code to work, but it makes your code vastly more legible.

> **Code Indentation Shortcuts**
>
> Highlight the line(s) of code and try one of these keystrokes:
>
> - **Tab** to indent. **Shift–Tab** to unindent.
> - **Cmd–]** (Mac) or **Ctrl–]** (Windows) to indent.
>   **Cmd–[** (Mac) or **Ctrl–[** (Windows) to unindent.

6. Although we currently have only one **div**, pages typically have multiple divs. Assuming more will be added later, to style this div we should give it a name (either a **class** or **ID**). Add the following ID to the opening **div** tag, as follows:

```
<body>
   <div id="wrapper">
      <h1>
```

We'll use this ID to apply CSS to this specific div. Do not use spaces or special characters in an ID. How is an ID different than a class? You can apply a class name to multiple elements, but an ID can only be applied to one element in a page.

7. In the **style** tag, between the **p** and **.author** rules, add the following new rule (highlighted in bold):

```
p {
    CODE OMITTED TO SAVE SPACE
}
#wrapper {
   width: 580px;
}
.author {
    CODE OMITTED TO SAVE SPACE
}
```

ID selectors use a **hash mark** (#) prior to the ID name, which tells the browser to find/style an element with that ID. Only one element can use an ID within a page, making ID selectors the most specific type of selector.

8. Save the file.

9. Return to the browser and reload the page to see the content reflows to fit within the 580 pixel-wide container.

10. Resize the browser window narrower than the width of the content. Notice that you have to scroll horizontally to see whatever does not fit inside the window. This is not ideal, but we wanted you to see how width works. There's an easy way to change this to a flexible, fluid layout that work a bit better in both large and small windows.
11. Return to **index.html** in your code editor.
12. In the **#wrapper** rule, change **width** to **max-width** as shown below in bold:

```
#wrapper {
   max-width: 580px;
}
```

13. Save the file.
14. Return to the browser and reload the page.
15. Resize the browser window narrower than the width of the content and notice that now the content reflows to stay inside the window and you are no longer forced to scroll horizontally. This will be much better for small screens.

Adding a Background-Color

1. Return to **index.html** in your code editor.
2. In the **style** tag, above all the other rules add a new rule for **body** (as shown below in bold):

```
<style>
   body {
      background-color: #bdb8ad;
   }
   h1 {
      font-family: Arial, Helvetica, sans-serif;
      font-size: 35px;
      color: #006b3a;
   }
```

3. Save the file.
4. Return to the browser and reload the page to see that the color applies to the entire page background (behind all the content).
5. Return to **index.html** in your code editor.

6. Let's add a white background behind only the text area to make it pop off the page's background. In the **style** tag, find the rule for **#wrapper** and add the following property declaration (in bold):

```
#wrapper {
   max-width: 580px;
   background-color: #ffffff;
}
```

7. Save the file.

8. Return to the browser and reload the page to see that the white background is applied only to the content inside the div tag. The body has a different background color that is behind the div.

Adding Padding Inside the Div

1. The content is very close to the edge of the browser and white background. Let's add some **padding** (space between the content and the edge of the containing element) to make it more legible. Return to **index.html** in your code editor and add the following property declaration (in bold) to the rule for **#wrapper**:

```
#wrapper {
   max-width: 580px;
   background-color: #ffffff;
   padding: 40px;
}
```

   This sets 40 pixels of padding on all four sides: top, right, bottom, and left.

2. Save the file.

3. Return to the browser and reload to see more white space around the content.

Centering Content in the Window

1. Return to **index.html** in your code editor and add the following two new property declarations (in bold) to the rule for **#wrapper**:

```
#wrapper {
   max-width: 580px;
   background-color: #ffffff;
   padding: 40px;
   margin-left: auto;
   margin-right: auto;
}
```

   NOTE: Margin is space outside of an element. Padding is space inside an element.

2. Save the file.
3. Return to the browser and reload the page to see how the browser centers the div in the browser window.

   Why does this work? Setting left and right margins (space outside an element) to **auto**, will automatically make them equal to one another. When the width for an element (such as the **div**) is smaller than its container (such as the **body**), the effect horizontally centers the div with its container.

Adding a Border

1. Return to **index.html** in your code editor and add the following three new property declarations (in bold) to the rule for **#wrapper**:

```
#wrapper {
   max-width: 580px;
   background-color: #ffffff;
   padding: 40px;
   margin-left: auto;
   margin-right: auto;
   border-width: 3px;
   border-style: solid;
   border-color: #959485;
}
```

   NOTE: The **border-style** property tells the browser to render the border as a solid line. Other possible values include dashed and dotted.

2. Save the file.
3. Return to the browser and reload the page to see the 3-pixel solid border around the white background div.

CSS Clean-up: Shorthand & "DRY"

Instead of specifying the width, style, and color for the border as three separate property declarations, CSS has a method of "shorthand" we can use to write a border style more elegantly.

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

2. Edit your rule for **#wrapper** as shown below. Note that, instead of three lines of code to describe the border properties, one **border** declaration can be written like so:

```
#wrapper {
   max-width: 580px;
   background-color: #ffffff;
   padding: 40px;
   margin-left: auto;
   margin-right: auto;
   border: 3px solid #959485;
}
```

3. While we're cleaning up our CSS, take note of how often we declare **font-family: Arial, Helvetica, sans-serif;**

   A core principle of coding is **DRY**, which stands for "don't repeat yourself". We try to avoid redundancies and write the most elegant code possible.

   We can declare the default font-family once (in the body rule) and have all elements inherit this property, because they sit inside the body tag.

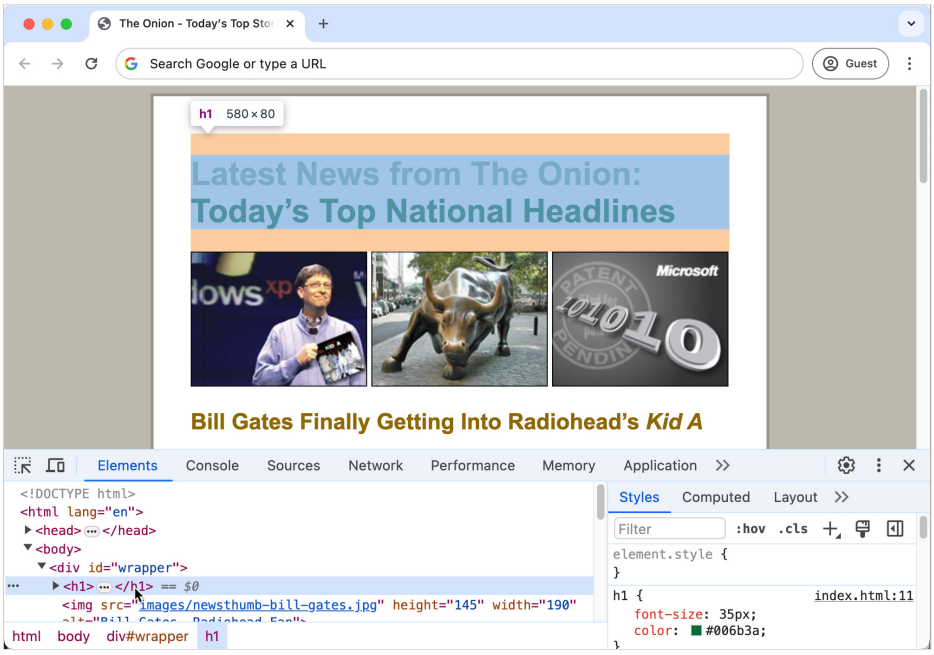
4. Edit your CSS rules as shown below. Delete the **font-family** declaration from **h1** and **h2**, and instead, place it inside the **body** rule:

```
body {
   font-family: Arial, Helvetica, sans-serif;
   background-color: #bdb8ad;
}
h1 {
   font-size: 35px;
   color: #006b3a;
}
h2 {
   font-size: 24px;
   color: #8f6800;
}
```

5. Save the file.

6. Return to the browser and reload the page. It should look the same as it did before the change, but the code is leaner and more elegant.

Exercise Preview



Exercise Overview

All major browsers have built-in tools that let you see and alter HTML and CSS on-the-fly. In this exercise, you'll see how useful they can be when experimenting with code changes.

We'll be using Chrome's DevTools, but the tools in other browsers work similarly.

1. If you completed the previous exercise, **index.html** should still be open, and you can skip the following sidebar. If you closed **index.html**, re-open it now. We recommend you finish the previous exercises (1D–2C) before starting this one. If you haven't finished them, do the following sidebar.

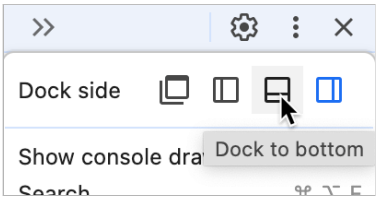
If You Did Not Do the Previous Exercises (1D–2C)

1. Close any files you may have open.
2. In your code editor, open **index-ready-for-dev-tools.html** from the **News Website** folder.
3. Do a **File > Save As** and save the file as **index.html**, replacing the older version in your folder.

Using Chrome's DevTools

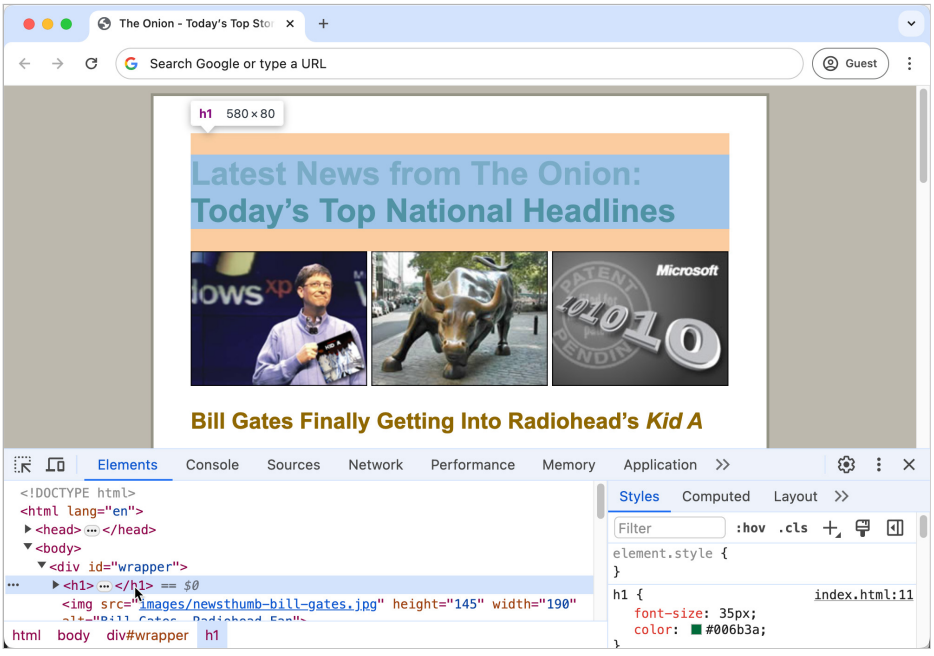
1. Preview index.html in **Chrome**.
2. To open the DevTools, **Ctrl–click** (Mac) or **Right–click** (Windows) on any of the images near the top of the page and choose **Inspect**.

   NOTE: The element you **Ctrl–click** (Mac) or **Right–click** (Windows) on will be initially selected in the DevTools.
3. 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 then choose **Dock to bottom** as shown below:



4. The DevTools are organized into tabbed panels. The **Elements** panel should currently be open on the left side. Here, you can view and edit all elements in the **DOM tree** (DOM is short for Document Object Model), which is a fancy way of saying you can see the structure of the document and the way each HTML element is nested inside another.

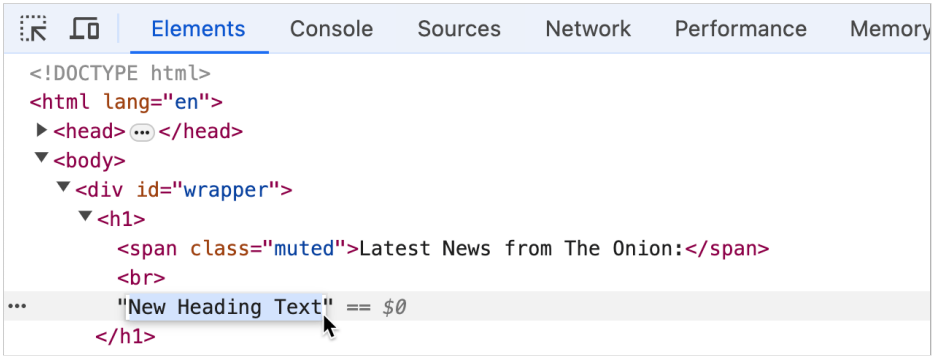
   Hover over some of the tags, noticing that as you do so, the element will be highlighted in the browser window above (along with a tooltip that shows the HTML element's width and height).



5. While you're here, also take note of the arrows that can be opened and closed to show the contents of the HTML elements. Experiment a little by opening up the **<h1>** tag (directly above the three **img** tags) to see the content.

6. Notice the nested **** and **
** tags. It is helpful to know that the <h1> is considered the parent element and the and
 are child elements within the <h1>. Following this logic, the and
 elements are siblings according to the structure of the DOM tree.

7. Double–click on the text inside the **<h1>** element and type out some new text (whatever you want), as shown below:



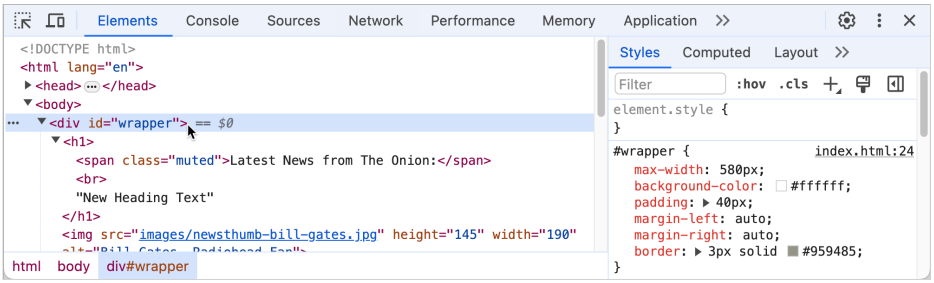
8. Hit **Return** (Mac) or **Enter** (Windows) to see the change in the browser window above the DevTools. Wow, that's cool!

   Play around for a bit and have fun. Don't worry, the changes you make in the DevTools are temporary (they are not saved into the HTML file). You need to modify the content in your actual HTML file to save the changes.

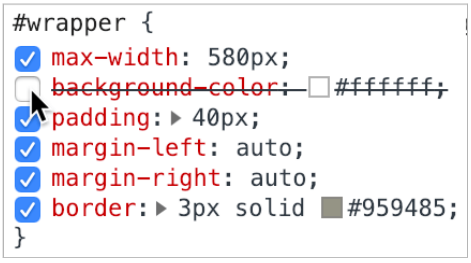
9. Reload the browser to see the original content once again.

10. Click on the ** ** tag in the DevTools, which is just below the **<body>** tag.

11. Look at the **Styles** panel on the right to see the CSS for **#wrapper**



12. Hover over the properties for the **#wrapper** rule and notice that **checkboxes** appear next to each property. Try unchecking and checking the properties to see how you can temporarily disable and re-enable the property in the browser. When a property is disabled, it will appear with a strike-through:



13. Find the **border** property and click on the value that's there (3px solid #959485).

14. Type in **7px dashed #f00** to see how the browser displays the change:

```
#wrapper {
   max-width: 580px;
   background-color: #ffffff;
   padding: 40px;
   margin-left: auto;
   margin-right: auto;
   border: 7px dashed #f00;
}
```

NOTE: Other possible values for border style are **dotted**, **double**, **groove**, **ridge**, **inset**, and **outset**.

Hexadecimal Shorthand

When a CSS color hex value contains **3 identical pairs**, it can be written as a 3-digit shorthand **`#fff`** (instead of the 6-digit **`#ffffff`**). Shorthand can only be used for colors with **3 identical pairs** of characters. Instead of **`#33cc66`** you could write **`#3c6`**. Colors such as **`#0075a4`** cannot be shortened.

15. Play around for a bit and have fun. Remember, the changes you make to the code in the DevTools are temporary. To save the changes you would need to remember the new values and then code them into your CSS file in your code editor.
16. Find the **width** property and click on the value to highlight it.
17. As you do the following, watch how the wrapper div's width is updated live in the browser window:
    - Hit the **Up Arrow** key on your keyboard to increase the value **1** pixel at a time.
    - Hit the **Down Arrow** key to decrease the value.
    - Hold **Shift** while hitting the Arrow keys to change the value **10** pixels at a time.

```
#wrapper {
   max-width: 625px;
   background-color: #ffffff;
   padding: 40px;
   margin-left: auto;
   margin-right: auto;
   border: 7px dashed #f00;
}
```

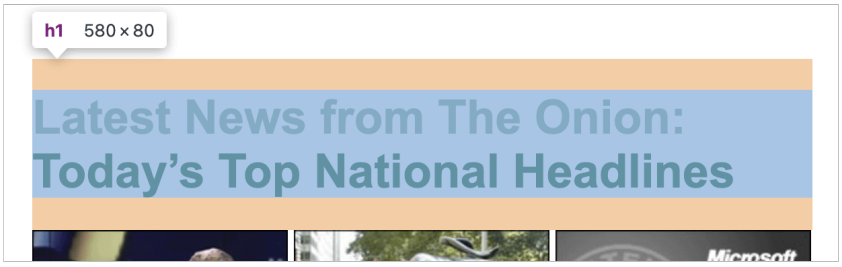
18. Reload the browser to return to the originally assigned values.

    We don't want to actually modify this element's width or border, but this gives you a sense of how a browser's DevTools can help you understand and figure out your CSS by providing immediate visual feedback on your changes.

    Let's put what we've learned to some use in fine-tuning this particular page.

Fine-Tuning Margins for the Main Heading

1. Look at the main heading of the page (Latest News from the Onion: Today's Top National Headlines) and notice that the space around the heading is uneven. There's way more space above the heading than on the left. Let's balance out this space.
2. On the left side of the DevTools, find the **<h1>** tag.
3. Hover over the **<h1>** tag and, as you do so, look at the way it is highlighted in the browser window above. You should see an orange-colored highlight above and below the header. The orange color indicates this element has margin:

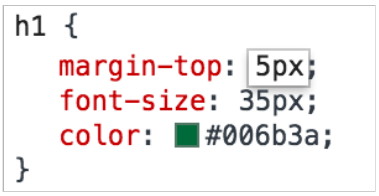


4. Click on the **<h1>** tag in the DevTools to select it. You will now see the **Styles** associated with the **h1** on the right side of the DevTools.

5. In the **Styles** panel of the DevTools, notice that we did not add any margin to **h1** to the CSS in the style tag embedded into **index.html** but, just below that rule, you'll see rules added by the **user agent stylesheet**. This is styling the browser does by default. All browsers add margin to the top and bottom of headings, paragraphs, and lists. Let's override this.

6. In the **Styles** section of DevTools, to edit our rule for **h1**, click once inside the opening curly brace:

7. Type **margin-top** and hit **Tab**. Then type **5px**



8. Use the **Up Arrow** and **Down Arrow** keys on your keyboard to modify the margin. Notice that the space looks more even when **margin-top** is set to **0px**. Let's add this CSS to our actual code to lock it in.

9. You should still have **index.html** open in your code editor. If not, open it now.

10. Find the **h1** rule (near the top of the **style** tag) and add the following code shown below in bold:

```
h1 {
   margin-top: 0;
   font-size: 35px;
   color: #006b3a;
}
```

    NOTE: In CSS, you almost always specify a unit of measurement (such as px) for a numeric value. Because zero is always zero, regardless of the unit of measurement, you do not need to specify px when coding 0 (zero).

11. Save the file.

12. Return to the browser and reload **index.html** to see the final change to the code.

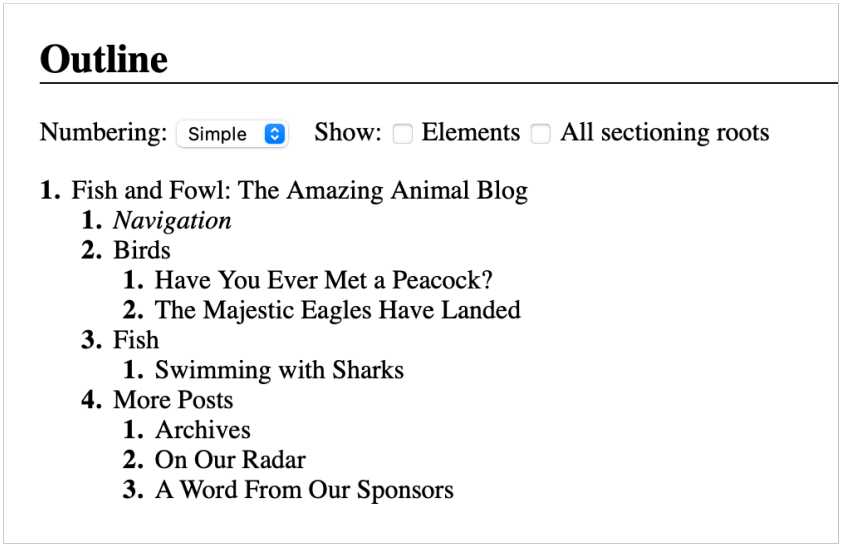
Checking for Errors: Validating Your Code

Validating your code is a way to test for coding mistakes such as unclosed HTML elements, typos, etc. If you have a problem with some code and can't find the mistake, running the code through a validator may help you to find it. Even if your page seems to be working properly, validating it may catch unseen errors. We'll use an online validation tool.

1. Open a browser and go to: **validator.w3.org**
2. Click the **Validate by File Upload** tab.
3. Click the **Choose File** or **Browse** button.
   - Navigate to **Class Files > Web Dev Class > News Website**.
   - Double–click on **index.html** to select it.
4. Click the **Check** button.

   It should say **Document checking completed. No errors or warnings to show.**

Exercise Preview



Exercise Overview

In a previous exercise, we used **span** and **div** tags to group content. These are non-semantic tags, because they say nothing about their content. HTML5 added new semantic tags (such as **article**, **header**, **footer**, **nav**, etc.) that actually describe their content. They are beneficial for developers, screen readers, and search engines.

Getting Started

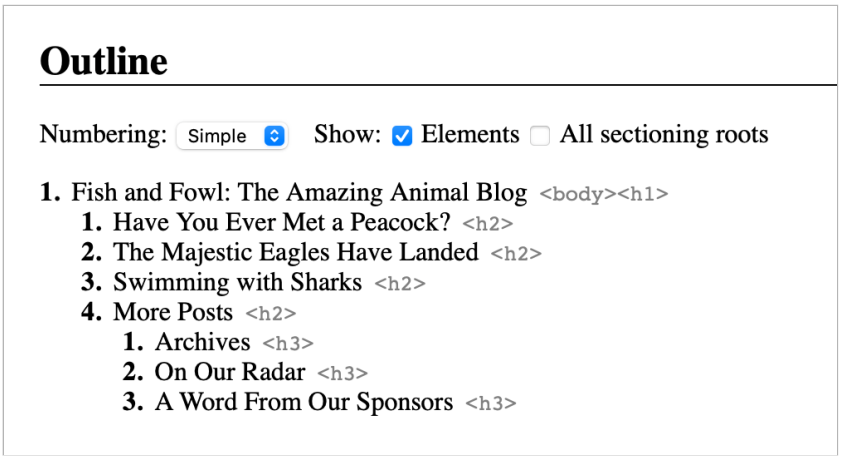
1. In your code editor, hit **Cmd–O** (Mac) or **Ctrl–O** (Windows) to open a file.
2. Navigate to **Desktop > Class Files > Web Dev Class > Structural Semantics**.
3. Double–click on **semantic-elements.html** to open it.
4. Before we start tagging the content, let's get acquainted with it. Preview **semantic-elements.html** in a web browser.
5. Scroll down the page and scan the content. In particular, note the three headings: two about birds (Peacocks and Eagles) and one about fish (Sharks). These three articles are the primary content for the page.
6. Switch back to **semantic-elements.html** in your code editor.
7. Look over the headings in the code, which range from h1 through h3.

The Outline Algorithm

Every webpage has a topical outline (similar to a table of contents) created by the various heading levels (h1, h2, etc). Let's take a look at the document outline and see if (and how) it can be improved.

1. Select all the code and copy it.
2. Open a web browser and go to: **hoyois.github.io/html5outliner**

   NOTE: There are also browser extensions to view the outline, but we'll use this website because it doesn't require any installation.
3. Under **Input HTML**, delete the existing code and paste in yours.
4. Click the **Show outline >** button.
5. Under **Outline**, check on **Elements** (next to Show).



This outline (document structure) is created by the heading tags (currently we have h1, h2, and h3 tags). The top level (main topic) is an h1. That is divided into subtopics with h2 tags. Those subtopics are further divided using h3 tags.

The Header

There are additional semantic (linguistically meaningful) tags that we can use to improve the structure of our HTML.

Let's begin with the **header** element. According to the **HTML spec**, "A header typically contains a group of introductory or navigational aids. A header element is intended to usually contain the section's heading (an h1–h6 element), but this is not required. The header element can also be used to wrap a section's table of contents, a search form, or any relevant logos."

1. Switch back to your code editor and find this line of code just after the start of the body tag:

   ```
   <h1>Fish and Fowl: The Amazing Animal Blog</h1>
   ```

2. As shown below, wrap these two lines of code in a **header** tag.

   TIP: If you're using Visual Studio Code (and set up the wrap keystroke using the instructions in the **Before You Begin** section near the start of this book), you can quickly wrap a tag around a selection using a keystroke. Select the code you want to wrap and hit **Option–W** (Mac) or **Alt–W** (Windows). Then type in the name of the wrapper (which in this case is **header**) and hit **Return** (Mac) or **Enter** (Windows).

   ```
   <header>
      <h1>Fish and Fowl: The Amazing Animal Blog</h1>
   </header>
   ```

   NOTE: There should be only one **h1** per page, which indicates that page's topic.

Proper Semantics for Nav Elements

1. Just below the header are some links to navigate the site. Wrap a **nav** tag around them, as shown below:

   ```
   <nav>
      <a href="#">About Us</a>
      <a href="#">Our Mission</a>
      <a href="#">Contact Us</a>
   </nav>
   ```

   NOTE: These don't link to real pages, yet. Setting the **href** to **#** is a common way to create a placeholder link because it doesn't go anywhere. It allows us create and style the links before we build the rest of the site.

2. While we're here, let's mark up the links with some additional semantic tags. This is essentially a list of links, so we'll mark them up as an unordered list. Wrap a **ul** tag around the links, as follows:

   ```
   <nav>
      <ul>
         <a href="#">About Us</a>
         <a href="#">Our Mission</a>
         <a href="#">Contact Us</a>
      </ul>
   </nav>
   ```

3. Now wrap each link in an **li** (list item) tag, as follows:

```
<ul>
   <li><a href="#">About Us</a></li>
   <li><a href="#">Our Mission</a></li>
   <li><a href="#">Contact Us</a></li>
</ul>
```

4. Save the file.

The Article, Aside, & Footer Elements

1. Below the nav, find the first blog post (an h2 and two paragraphs). The **article** element is ideal for this type of content.

   According to the **HTML spec**, "The article element represents a complete, or self-contained, composition in a document, page, application, or site and that is, in principle, independently distributable or reusable, e.g. in syndication. This could be a forum post, a magazine or newspaper article, a blog entry, a user-submitted comment, an interactive widget or gadget, or any other independent item of content."

2. As shown below, wrap the following lines of code in an **article** tag:

```
<article>
   <h2>Have You Ever Met a Peacock?</h2>
   CODE OMITTED TO SAVE SPACE
   <p>Source: <a href="http://en.wikipedia.org/wiki/Peafowl">Wikipedia</a></p>
</article>
```

3. Wrap the **article** tag around the **two** remaining articles. The code should look like this when you are finished:

```
<article>
   <h2>The Majestic Eagles Have Landed</h2>
   CODE OMITTED TO SAVE SPACE
   <p>Source: <a href="http://en.wikipedia.org/wiki/Eagle">Wikipedia</a></p>
</article>

<article>
   <h2>Swimming with Sharks</h2>
   CODE OMITTED TO SAVE SPACE
   <p>Source: <a href="http://en.wikipedia.org/wiki/
Whale_shark">Wikipedia</a></p>
</article>
```

4. Now let's mark up the additional posts and links. The **aside** element is used to indicate tangential, additional content. The makers of the HTML spec chose "aside" over "sidebar" because it more accurately describes the semantic value of the content rather than the content's position on the page. Sidebars are often placed off to the left or the right but then they are moved to the bottom of the main content for a mobile layout. Aside is a more flexible term.

   As shown below, wrap the following content in an **aside** tag:

```
<aside>
   <h2>More Posts</h2>
   <h3>Archives</h3>

   CODE OMITTED TO SAVE SPACE

   <h3>A Word From Our Sponsors</h3>
   <ul>
      <li>PETA</li>
      <li>ASPCA</li>
   </ul>
</aside>
```

5. The last paragraph contains a copyright notice. It's a perfect **footer** element, which according to the **HTML spec**, "typically contains information about its section such as who wrote it, links to related documents, copyright data, and the like".

   Wrap the **footer** tag around the last paragraph, as shown below:

```
   <footer>
      <p>© fish-and-fowl.org - all rights reserved - all wrongs reversed</p>
   </footer>
</body>
```

The Section Element

According to the **HTML spec**, a **section** element is "a thematic grouping of content, typically with a heading. Examples of sections would be chapters, the various tabbed pages in a tabbed dialog box, or the numbered sections of a thesis. A Web site's home page could be split into sections for an introduction, news items, and contact information." We have two groups of articles (birds and fish), so the section element will be perfect for marking them up.

1. Wrap the two articles about birds (**peacocks** and **eagles**) in one **section** tag, as shown below:

```
<section>
   <article>
      <h2>Have You Ever Met a Peacock?</h2>

      CODE OMITTED TO SAVE SPACE

   </article>

   <article>
      <h2>The Majestic Eagles Have Landed</h2>

      CODE OMITTED TO SAVE SPACE

   </article>
</section>
```

2. Wrap the last article (about **sharks**) in a **section** tag, as shown below:

```
<section>
   <article>
      <h2>Swimming With Sharks</h2>

      CODE OMITTED TO SAVE SPACE

   </article>
</section>
```

3. Save the file.

> **Section vs. Article**
>
> An **article** is perfect for syndication: a discrete piece of content that can stand on its own and be reused elsewhere. Think of a blog post or a news story.
>
> A **section** divides content into different subject areas. Here we grouped articles into sections but, if the articles were longer, we could also have sections nested in articles to further divide the content.

Adding Titles to Sections

According to the **HTML spec for creating an outline**, "Each section can have one heading associated with it".

For the bird and fish sections, we can add a heading to label them. That would be useful info and actually improve the page for users.

1. Return to **semantic-elements.html** in your code editor.

2. Let's give the birds section a logical heading. Directly after the opening **section** tag add the following bold code:

```
<section>
   <h2>Birds</h2>
   <article>
      <h2>Have You Ever Met a Peacock?</h2>
```

3. Let's give the fish section a logical heading too. After the section opening tag type:

```
<section>
   <h2>Fish</h2>
   <article>
      <h2>Swimming With Sharks</h2>
```

4. Now that we the section headings are h2 tags, the article headings should change from h2 to h3. Change the three article headings, as shown below in bold:

```
<section>
   <h2>Birds</h2>
   <article>
      <h3>Have You Ever Met a Peacock?</h3>
      CODE OMITTED TO SAVE SPACE
   </article>

   <article>
      <h3>The Majestic Eagles Have Landed</h3>
      CODE OMITTED TO SAVE SPACE
   </article>
</section>

<section>
   <h2>Fish</h2>
   <article>
      <h3>Swimming with Sharks</h3>
```

5. Save the file.

Rechecking the Outline

1. Select all your code and copy it.
2. In a browser, go back to **hoyois.github.io/html5outliner**
3. Under **Input HTML**, delete the existing code and paste in yours.
4. Click the **Show outline >** button.

5. Under **Outline**, make sure Show: **Elements** is checked on.

Original

1. Fish and Fowl: The Amazing Animal Blog `<body><h1>`
    1. Have You Ever Met a Peacock? `<h2>`
    2. The Majestic Eagles Have Landed `<h2>`
    3. Swimming with Sharks `<h2>`
    4. More Posts `<h2>`
        1. Archives `<h3>`
        2. On Our Radar `<h3>`
        3. A Word From Our Sponsors `<h3>`

Improved

1. Fish and Fowl: The Amazing Animal Blog `<body><h1>`
    1. *Navigation* `<nav>`
    2. Birds `<section><h2>`
        1. Have You Ever Met a Peacock? `<article><h3>`
        2. The Majestic Eagles Have Landed `<article><h3>`
    3. Fish `<section><h2>`
        1. Swimming with Sharks `<article><h3>`
    4. More Posts `<aside><h2>`
        1. Archives `<h3>`
        2. On Our Radar `<h3>`
        3. A Word From Our Sponsors `<h3>`

The Main Element

Let's define the most important content in the document with the **main** tag. It does not affect the document outline but adds semantic value. The **main** element's primary purpose is to improve accessibility. It helps screen readers understand where the main content begins. It can only be used once per document and it must not be nested inside of an article, aside, footer, header, or nav.

1. Return to **semantic-elements.html** in your code editor.

2. Wrap the two section elements in the **main** tag, as shown below.

```
<main>
   <section>
      <h2>Birds</h2>
      <article>
          CODE OMITTED TO SAVE SPACE
      </article>

      <article>
          CODE OMITTED TO SAVE SPACE
      </article>
   </section>

   <section>
      <h2>Fish</h2>
      <article>
          CODE OMITTED TO SAVE SPACE
      </article>
   </section>
</main>
```

3. Save the file.

Optional Bonus: Figure & Figcaption

Let's add an image of an exotic animal to our page.

1. Return to **semantic-elements.html** in your code editor.
2. Below the **Birds** heading, add the following **img**:

```
<section>
   <h2>Birds</h2>

   <img src="img/tawny-frogmouth.jpg" height="346" width="700" alt="Tawny
Frogmouth">

   <article>
```

3. Save the file and preview **semantic-elements.html** in a web browser.

What a cool bird! It would be nice to add a caption to provide details about the bird. A **figure** with a **figcaption** is a semantic way to include an image, chart, or code example accompanied by an explanatory caption. Let's see how it works.

4. Return to **semantic-elements.html** in your code editor.
5. Wrap the **img** tag in the following **figure** element:

```
<figure>
   <img src="img/tawny-frogmouth.jpg" height="346" width="700" alt="Tawny
Frogmouth">
</figure>
```

6. Add the **figcaption** element inside the **figure** tag below the image, as follows:

```
<figure>
   <img src="img/tawny-frogmouth.jpg" height="346" width="700" alt="Tawny
Frogmouth">
   <figcaption></figcaption>
</figure>
```

7. To save you time, we've typed out the caption content. Open **content-figcaption.html** from the **snippets** folder (in the **Structural Semantics** folder).
8. Copy all the text.
9. Close the file and return to **semantic-elements.html**.
10. Paste the code inside the empty **<figcaption></figcaption>** tags, as follows:

```
<figcaption>The Tawny Frogmouth (Podargus strigoides) is a type of bird found
throughout the Australian mainland, Tasmania...frogmouths are more closely
related to nightjars and oilbirds. Source: <a href="http://en.wikipedia.org/
wiki/Tawny_frogmouth">Wikipedia</a></figcaption>
```

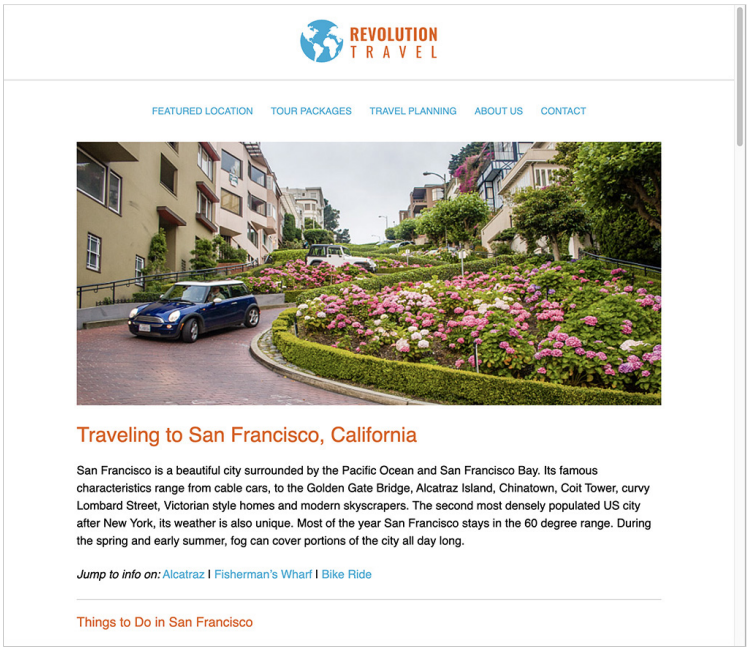
11. Save the file and preview **semantic-elements.html** in a web browser. Interesting!

    Notice that some margin space has been added on the left of the figure. There's also margin on the right, but it's not obvious. These margins could be modified with CSS, but for now we just wanted to focus on the HTML.

Who Decides the HTML Specifications?

The **W3C** (World Wide Web Consortium) is an international community of member organizations led by Web inventor Tim Berners-Lee. Later, the **WHATWG** (Web Hypertext Application Technology Working Group) was formed by Apple, Google, Mozilla (makers of Firefox), and Opera because of some controversial decisions made by the W3C. The two groups have since come to an agreement that the **WHATWG** decides and publishes the HTML Living Standard. You can learn more at **whatwg.org** and **w3.org**. For a brief story about their history, read **en.wikipedia.org/wiki/whatwg**

Exercise Preview



Exercise Overview

This is the first in a series of exercises in which you'll lay out a complete website. In this exercise, you'll begin by laying out the basics of a single page. You will build the structure of the page and place content inside each section.

Getting Started

1. We'll be using a new folder of provided files for this exercise. This website folder contains images, some partially made webpages for you to finish, and a couple of code snippets, one of which contains tagged HTML content for you to copy and paste. Close any files you may have open in your code editor to avoid confusion.

2. We'll be working with files in a folder named **Revolution Travel** located in **Class Files > Web Dev Class**.

   TIP: It's helpful to see the entire website folder as you work. If you're working in Visual Studio Code, you can:

   - Go to **File > Open Folder**.
   - Navigate to **Class Files > Web Dev Class > Revolution Travel** and hit **Open** (Mac) or **Select Folder** (Windows).
   - If you get a message about trusting the author, check on **Trust the authors of all files in the parent folder** and click **Yes, I trust the authors**.

3. In your code editor, open **san-francisco.html** from the **Revolution Travel** folder.
4. Change the title of the document to the following, shown below in bold:

```
<head>
   <meta charset="UTF-8">
   <title>Traveling to San Francisco, CA - Revolution Travel</title>
</head>
```

5. Save the file.

Coding Up the Sections

1. To see a finished version of the page you'll build, navigate to the **Desktop** and go into the **Class Files** folder, then **Web Dev Class** folder, then **Revolution Travel Done**.
2. **Ctrl–click** (Mac) or **Right–click** (Windows) on **san-francisco.html**, go to **Open with** and select your favorite browser.
3. Notice that the page is composed of four main sections: a header (the logo), navigation, main content, and a footer (the copyright) on the bottom. Let's code up each of these sections.
4. Return to **san-francisco.html** in your code editor.
5. Add the following sectioning tags (highlighted in bold) inside the **body** tag:

```
<body>
   <header></header>
   <nav></nav>
   <main></main>
   <footer></footer>
</body>
```

6. If you were to preview the file in a browser, you wouldn't see a thing. By default, section elements have no border, no background, and are only as tall as the content inside of them. Browsers will render these sections as block-level elements.

Block-level Elements

Block-level elements stack on top of one another like a child's building blocks. Each one begins on a new line and takes up 100% of the width of their parent container. These sectioning elements (header, nav, main, etc.) are block-level elements. Paragraphs, headings, divs, and lists are some other block-level elements that we've already covered.

Adding Content to the Header

1. The **header** should feature the company's logo. Add the following bold code:

```
<body>
   <header><img src="images/revolution-logo.svg"></header>
   <nav></nav>
   <main></main>
   <footer></footer>
</body>
```

2. While you're here, don't forget to add the alt text:

```
<img src="images/revolution-logo.svg" alt="Revolution Travel">
```

3. Save the file.
4. Preview the page in a browser to see the logo.

Adding the Main Content

1. To save you some time, we've tagged up most of the main content for the page and saved it into a file. In your code editor, open **page-content.html** from the **snippets** folder (in the **Revolution Travel** folder).
2. Select all the code (**Cmd–A** (Mac) or **Ctrl–A** (Windows)).
3. Copy it (**Cmd–C** (Mac) or **Ctrl–C** (Windows)).
4. Close the file.
5. You should be back in **san-francisco.html**.
6. Place the cursor between the **<main>** opening and closing tags.
7. To make the code more legible, hit **Return** (Mac) or **Enter** (Windows) to place the closing `</main>` tag on its own line like so:

```
<main>

</main>
```

8. Paste (**Cmd–V** (Mac) or **Ctrl–V** (Windows)) the code into the **main** section.
9. Save the file.

Adding the Nav Content

The first five lines of the content you just pasted into the main section is actually the text for our navigation links. Let's move them to the appropriate section.

1. Place your cursor between the **<nav>** opening and closing tags and hit **Return** (Mac) or **Enter** (Windows) to place the closing tag on its own line like so:

```
<nav>

</nav>
```

2. In the **main** section, select the first five lines, starting with **Featured Location** and ending with **Contact**.
3. Cut the text (**Cmd–X** (Mac) or **Ctrl–X** (Windows)).
4. Now paste it into the **nav** like so:

```
<nav>
   Featured Location
   Tour Packages
   Travel Planning
   About Us
   Contact
</nav>
```

5. To clean up the code, delete any leftover **whitespace** (empty lines) at the start of the **main** section.
6. Let's also mark up the nav element with the additional semantic value of an unordered list. Wrap a `<ul>` tag around all the nav content as well as an `<li>` tag around each nav item, as follows:

```
 <nav>
   <ul>
      <li>Featured Location</li>
      <li>Tour Packages</li>
      <li>Travel Planning</li>
      <li>About Us</li>
      <li>Contact</li>
   </ul>
</nav>
```

7. Save the file.

Adding the Footer Content

1. Scroll down to the code at the bottom of the page. Place your cursor between the **<footer>** opening and closing tags and hit **Return** (Mac) or **Enter** (Windows) to place the closing tag on its own line like so:

```
<footer>

</footer>
```

2. Select the last paragraph of the main content (the copyright paragraph) and cut it (**Cmd–X** (Mac) or **Ctrl–X** (Windows)).
3. Paste it into the **footer** like so:

```
<footer>
   <p>© Revolution Travel</p>
</footer>
```

4. To clean up the code, delete any leftover **whitespace** (empty lines) at the end of the **main** section.
5. Save the file.
6. Return to the browser, and reload the page. Nothing has been styled yet, so each section of content stacks on top of each other, and wraps to the edge of the browser window.

Marking Up the Headings

Let's mark up the text to give more structure to the topics on the page.

1. Return to your code, and near the top find the first paragraph in the **main** section.
2. As shown below, change the **p** tags to **h1** tags:

```
<main>
   <h1>Traveling to San Francisco, California</h1>
```

3. A couple paragraphs down, find **Things to Do in San Francisco** and change the **p** tags to **h2** tags:

```
<h2>Things to Do in San Francisco</h2>
<p>San Francisco is filled with culture, fine eating, nightlife and more. From
beaches and parks, museums, bike riding, cable cars, to street performers,
there is always something to do. The hard part is deciding how you'll spend
your time! Here are a few recommendations:</p>
```

4. Several paragraphs below that, find **San Francisco Travel Tips** and again change the **p** tags to **h2** tags:

```
<h2>San Francisco Travel Tips</h2>
<p>San Francisco has pretty mild temperatures all year long. While that means
you won't be too freezing or too hot, don't be fooled by the lore of "sunny
California." San Fran does tend to be on the chilly side, especially at night
or when the wind gets blowing. Bring some extra layers in case you need them.
Most people are surprised to find it's colder than they expected.</p>
```

5. **Things to Do in San Francisco** is broken down into 3 activities, so change those **p** tags to **h3** tags:

```
<h3>Tour Alcatraz Island</h3>
```

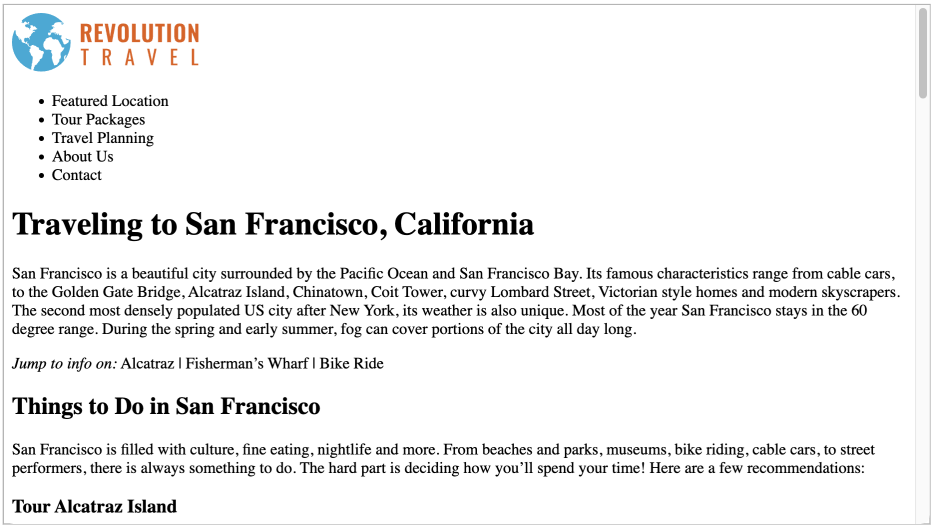
CODE OMITTED TO SAVE SPACE

```
<h3>Visit Fisherman's Wharf</h3>
```

CODE OMITTED TO SAVE SPACE

```
<h3>Ride a Bike around San Francisco</h3>
```

6. Save the file, return to the browser, and reload to see the improved markup.



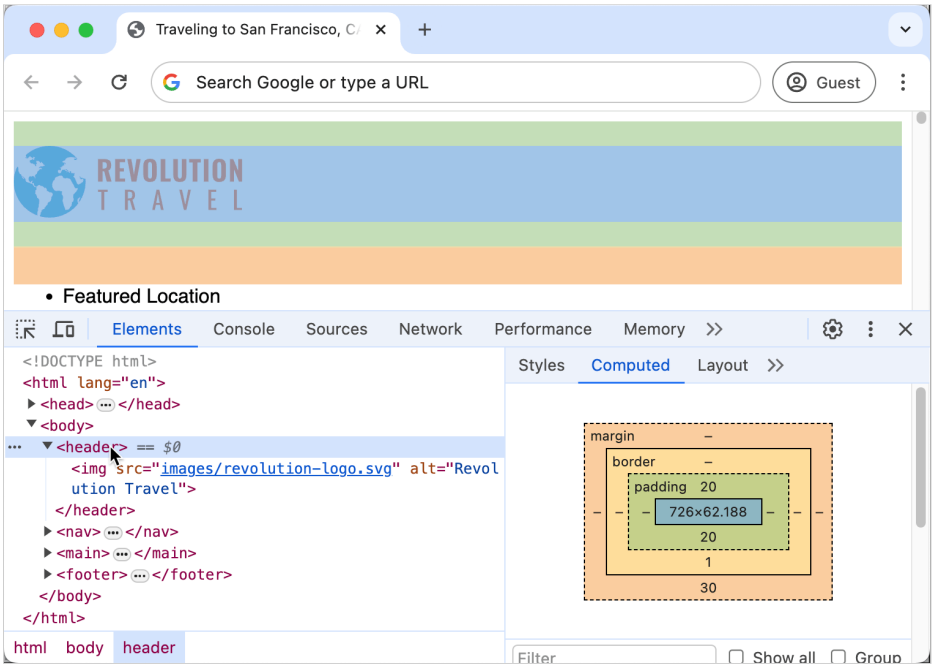
7. The page doesn't have much going on yet, but you now have most of the HTML structure needed to start styling the elements. You can keep **san-francisco.html** open in the browser and the code editor as you'll continue them in the next exercise.

How to Create a Brand New HTML File

In this exercise we provided a starter HTML file with the basic tags. If you don't have that, here's how to create a new HTML file in Visual Studio Code (and most code editors that have Emmet installed):

1. Go to **File > New File**.
2. Save the file as **some-file-name.html**
3. To quickly create all the basic HTML tags, type an **exclamation point** (!) and hit **Tab**. (This step requires Emmet which Visual Studio Code has.)

Exercise Preview



Exercise Overview

HTML elements that hold content can have space both inside and outside of them, so it is helpful to consider them as boxes when thinking about styling these elements with CSS. In this exercise, we'll explore this CSS Box Model to see how we use the width, padding, and margin properties to control the page layout.

1. If you completed the previous exercise, **san-francisco.html** should still be open, and you can skip the following sidebar. If you closed **san-francisco.html**, re-open it now. We recommend you finish the previous exercise (3B) before starting this one. If you haven't finished it, do the following sidebar.

If You Did Not Do the Previous Exercise (3B)

1. Close any files you may have open.
2. On the **Desktop**, go to **Class Files > Web Dev Class**.
3. Delete the **Revolution Travel** folder if it exists.
4. Duplicate the **Revolution Travel Ready for Box Model** folder.
5. Rename the folder to **Revolution Travel**.

Getting Started

1. In **san-francisco.html**, add a **<style>** tag to the head of the document as follows:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Traveling to San Francisco, CA - Revolution Travel</title>
   <style>

   </style>
</head>
```

2. Save the file.

Setting Width

For our design, we want the header and nav to stretch across 100% of the browser window (which they do by default), but the **main** content and **footer** should not fill the page.

1. Add the following code (highlighted in bold) inside the **style** tag:

```
<style>
   main, footer {
      width: 90%;
   }
</style>
```

NOTE: We can style multiple elements with a single CSS rule by listing them (separated by commas). Height is usually not specified, because the content may be edited later, and in this case the height will change as the browser width changes.

2. Save the file.
3. Preview the file in a browser.
   - The main text content is now a bit narrower, so there's some whitespace to the right of the text. It would look better if that space were equally divided on the left and right (so the section was centered in the page).
   - This width is good for small screens, but make the browser window as wide as you can. Notice that the long lines of text become hard to read. We can use max-width to limit the content to a width that is more reader-friendly.

   TIP: Keep this file open in the browser so you can reload the page to test the code as you work.
4. Return to your code editor.

5. Add the following new property declaration to the **main** rule:

```
main, footer {
   width: 90%;
   max-width: 800px;
}
```

6. Save the file.
7. Return to the browser, and reload the page.
   - Resize the browser window to see that the fluid layout adjusts to the window size when it's narrow, but is a fixed 800px width when the window is wide.
   - The images won't scale down, but we'll fix them soon.
   - Now let's make it so the space equally divided on the left and right of the content.
8. Return to **main.css** in your code editor.
9. In the **main, footer** rule, add the following new properties (in bold):

```
main, footer {
   width: 90%;
   max-width: 800px;
   margin-left: auto;
   margin-right: auto;
}
```

10. Save the file and reload any of the site's pages in a browser. Notice that all the main content is centered across the board. Much better.

The Box Model

Specifying an element's width, as you've done for both the main and footer sections, is just about the simplest way you can describe an HTML element's "box," or how much space it takes up in the flow of the document.

Any element can be considered a box, not just section elements and divs. The box model allows you to not only specify the **width** and **height** of content, it allows you to place **padding** within the element's box and **margins** between the element and other content on the page. You will explore padding, margin, and the **border** property (which is also part of the box model) later in this exercise.

Adding a Hero Image

In web design lingo, a hero image is a large banner-style image prominently placed at the top of a webpage. Its purpose is to provide a sort of visual overview of the page's content. Let's add a hero image to our page about visiting San Francisco.

1. In **san-francisco.html**, add an image inside **main**, above the **h1** tag:

```
<main>
   <img src="images/san-fran-lombard.jpg" alt="Lombard Street">
   <h1>Traveling to San Francisco, California</h1>
```

2. Save the file and preview it in a browser to see the image.
3. Looks good! Resize the window from wide to narrow and notice the wide photos do not scale down to fit within the window. Let's fix that.
4. Return to your code editor.
5. In the **style** tag at the top, add the following new **img** rule:

```
<style>
   img {
      max-width: 100%;
   }
   main, footer
```

This max-width ensures images will not be wider than 100% of the width of its parent container. It will not scale images up beyond their native width, but will scale them down to fit within their container. For this to work properly, you should remove the **width** and **height** attributes on any **img** tags that might be scaled.

6. Save the file and preview it in a browser.

Resize the window and make sure the photos scale down to fit within the browser window. Perfect!

Styling the Text

Let's style the text. We'll start by creating a rule for the **body** tag. By setting text styling on a parent tag (in this case the body tag), all the text-based children of that element will inherit the styling.

1. Back in your code editor, add the following code (highlighted in bold) inside the style tag above the **img** rule:

```
body {
   font-family: sans-serif;
}
img {
   max-width: 100%;
}
```

   NOTE: Typically the default sans-serif font on Mac is Helvetica and Windows is Arial. Although rare, users can set their sans-serif font to something else. For more control, use a more specific font-stack such as Arial, Helvetica, sans-serif.

2. Save the file, return to the browser, and reload the page.

   - All the text should be sans-serif, and we didn't have to make a rule for each individual element!
   - Notice the headings are all bold because that's the default browser styling for headings. Let's see how we can remove the bold (and change their color).

3. Back in your code editor, add the following new rule (in bold) below the **img** rule:

```
h1, h2, h3 {
   color: #e45c09;
   font-weight: normal;
}
```

4. Save the file and reload the page in a browser to see the headings are no longer bold and are colored.

5. We want to change the font size of the h1. Back in your code editor, add the following new rule (in bold) below the **h1, h2, h3** rule:

```
h1, h2, h3 {
   color: #e45c09;
   font-weight: normal;
}
h1 {
   font-size: 28px;
}
```

6. Save the file, return to the browser, and reload the page to see the main heading **Traveling to San Francisco, California** is now a little smaller.

   Resize the browser window so the heading wraps. The line-height (called leading in some design apps) is a bit tight, so let's increase it.

1. Back in your code editor, add the following property to the **h1** rule:

```
h1 {
   font-size: 28px;
   line-height: 36px;
}
```

2. While we're here, let's make the other headings (h2 and h3) a bit smaller. Below the **h1** rule, add the following new rules for **h2** and **h3**:

```
h1 {
   font-size: 28px;
   line-height: 36px;
}
h2 {
   font-size: 23px;
}
h3 {
   font-size: 18px;
}
```

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

   - The **Traveling to San Francisco, California** heading has more space between the lines.
   - The subheadings should now be smaller.
   - The paragraph line spacing is a bit tight. Let's fix that.

1. Back in your code editor, below the **h3** rule, add the following new rule for **p**:

```
p {
   line-height: 24px;
}
```

   NOTE: The default font-size is 16px. That's what we want for this design, so we don't have to code it.

2. Save the file, return to the browser, and reload the page. Now the paragraphs have a nice amount of space between the lines.

Margin

While the space between lines in paragraphs is improved, we'd like more space below each paragraph. Margin comes in handy for adjusting the space between elements. Margin is space **outside** an element's "box". For text, this is similar to space after or before a paragraph in design apps like Photoshop, InDesign, etc.

1. Back in your code editor, add the **margin-bottom** (below in bold) to the **p** rule:

```
p {
   line-height: 24px;
   margin-bottom: 22px;
}
```

2. Save the file, return to the browser, and reload the page to see a subtle change in the space between paragraphs.

3. Let's add space between the main sections of the page. Back in your code editor, above the **main, footer** rule, add the following new rule:

```
header, nav, main, footer  {
   margin-bottom: 30px;
}
main, footer {
```

4. Save the file, return to the browser, and reload the page to see there is now more space below the header (logo), below the nav, and the other main sections.

Padding & Border

Margin is space **outside** an element. Padding is space **inside** an element. A border will appear at the edge of the element (between the padding and margin).

1. Return to your code editor.

2. Let's create a visual separation between the header (logo) and the nav with a border. Below the **main, footer** rule, add the following new rule:

```
header {
   border-bottom: 1px solid #ccc;
}
```

3. Save the file, return to the browser, and reload the page. There should now be a gray border below the logo. The space below the border (margin-bottom) is fine, but we want more space above the border (between the logo and the border). This is where padding comes in handy.

4. Back in your code editor, add the following new property to the **header** rule:

```
header {
   border-bottom: 1px solid #ccc;
   padding-top: 20px;
   padding-bottom: 20px;
}
```

NOTE: The order of the properties does not matter.

5. Save the file, return to the browser, and reload the page to see the improved spacing between the logo and the border.
6. You can keep **san-francisco.html** open in the browser and the code editor. You'll continue with this file in the next exercise.

Margin vs. Padding

The margin and padding properties can be used to fine-tune the space between elements in a layout, though each adds space a bit differently. When thinking of a box, **padding** is **inside** the box. Padding actually opens up an element's box by making it appear wider and/or taller. For instance, if the element has a background color or border, the padding stretches the background and pushes the border farther away from the content.

Margin, on the other hand, is **outside** the element's box. The space is transparent.

For more information and to see a graphic that illustrates the difference between these two properties, see the **Box Model** reference at the back of the workbook.

Exercise Preview

Traveling to San Francisco, California

San Francisco is a beautiful city surrounded by the Pacific Ocean and San Francisco Bay. Its famous characteristics range from cable cars, to the Golden Gate Bridge, Alcatraz Island, Chinatown, Coit Tower, curvy Lombard Street, Victorian style homes and modern skyscrapers. The second most densely populated US city after New York, its weather is also unique. Most of the year San Francisco stays in the 60 degree range. During the spring and early summer, fog can cover portions of the city all day long.

Jump to info on: Alcatraz | Fisherman's Wharf | Bike Ride

Exercise Overview

This exercise will refresh you on how to link to other pages within a site and how to code external links that open in a new browser tab or window. You will also beef up your linking skills by creating links to different sections within a single webpage and wrapping links around images.

1. If you completed the previous exercise, **san-francisco.html** should still be open, and you can skip the following sidebar. If you closed **san-francisco.html**, re-open it now. We recommend you finish the previous exercises (3B–3C) before starting this one. If you haven't finished them, do the following sidebar.

If You Did Not Do the Previous Exercises (3B–3C)

1. Close any files you may have open.
2. On the **Desktop**, go to **Class Files > Web Dev Class**.
3. Delete the **Revolution Travel** folder if it exists.
4. Duplicate the **Revolution Travel Ready for Links** folder.
5. Rename the folder to **Revolution Travel**.

Coding Links to Pages Within a Site

Before you start coding the links, here is a tip that you should keep in mind. If you're using Visual Studio Code (and set up the wrap keystroke using the instructions in the **Before You Begin** section near the start of this book), you can quickly wrap a tag around a selection using a keystroke. Select the code you want to wrap and hit **Option–W** (Mac) or **Alt–W** (Windows). Then type in the name of the tag you want to wrap (for links that is **a**) and hit **Return** (Mac) or **Enter** (Windows).

1. Let's begin by coding the navigation links that lead to other pages in this site. In **san-francisco.html**, find the **nav** (just after the start of the **body** tag). Wrap the following anchors and hrefs (in bold) around the text:

```
<nav>
   <ul>
      <li><a href="san-francisco.html">Featured Location</a></li>
      <li><a href="tour-packages.html">Tour Packages</a></li>
      <li><a href="travel-planning.html">Travel Planning</a></li>
      <li><a href="about.html">About Us</a></li>
      <li><a href="contact.html">Contact</a></li>
   </ul>
</nav>
```

   NOTE: When linking from one file to another in a site, we use **relative links**. The links are relative to the location of the file in which you are coding. If both the file you are in and the one you are linking to are in the same folder (like ours are), all you have to type for the value of the **href** (hyperlink reference) is the file name.

2. Save the file.
3. Preview **san-francisco.html** in a browser to test a link. (These pages have been provided for you and are not styled yet. We'll style them in a later exercise.)
4. Hit the browser's **Back** button to return to **san-francisco.html**. Leave this page open in the browser so you can reload the page as you work.

Wrapping Links Around Images

1. Let's add a link around the logo image. Traditionally, the logo image links to the homepage (index.html). Return to **san-francisco.html** in your code editor.
2. Above the **nav**, there's a logo image inside the **header**. Wrap the image in an anchor tag with an href to the index page, as shown below in bold:

```
<header><a href="index.html"><img src="images/revolution-logo.svg"
alt="Revolution Travel"></a></header>
```

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

4. Click on the logo and you should be taken to the homepage. (This is another unstyled file we have provided for you.)
5. Hit the browser's **Back** button to return to **san-francisco.html**.

Creating Links to Other Websites

In the list of **Things to Do in San Francisco** there are 3 **Learn More** bits of text that each need to be wrapped in an anchor tag with an href to the appropriate website.

1. Back in your code editor, scroll down to the **main** section and find the **Tour Alcatraz Island** heading.

   A couple paragraphs below that wrap an anchor tag around Learn More as shown below in bold:

   ```
   <p>
      <a href="https://www.alcatrazcruises.com">Learn More →</a>
   ```

   When linking to another website you must always use **http://** (hypertext transfer protocol) or **https://** (HTTP secure), depending on which protocol the site you are linking to uses. It's best to copy the URL directly from the browser's address bar.

2. Save the file and reload your page in a browser to test out the link. Awesome.
3. Hit the browser's **Back** button to return to **san-francisco.html**.

   What if you prefer to open the link in another browser tab or window? Let's see how to do that.

4. Back in your code editor, add the **target** attribute to anchor tag and a **class** so we can style it later:

   ```
   <a href="https://www.alcatrazcruises.com" target="_blank" class="learn-more">Learn More →</a>
   ```

5. Save the file and reload your page in a browser to test out the link once more.

   This time the link will open in a new tab. (If a user has set preferences for links to open in windows rather than tabs, a new window will open.)

   Let's use the same method for linking up the other two images in this section.

6. Back in your code editor, find the **Visit Fisherman's Wharf** heading, and wrap an anchor tag around its **Learn More** (don't forget the **target** and **class**):

   ```
   <a href="https://www.fishermanswharf.org" target="_blank" class="learn-more">Learn More →</a>
   ```

7. Find the **Ride a Bike around San Francisco** heading, and wrap an anchor tag around its **Learn More** (don't forget the **target** and **class**):

```
<a href="https://www.blazingsaddles.com" target="_blank" class="learn-more">Learn More →</a>
```

8. Save the file and reload your page in a browser to test out the two new links. Each should open in its own new tab (or window).

Links Within a Page

In order to make navigating the content of a page easier, **IDs** can be coded into sections of the document and then links to those IDs can be provided from a "Table of Contents" of sorts. A visitor can click a link to jump down the page and view a section.

We already have content set up to for these types of "page jumps". Let's code the links to make it work.

1. Add an **ID** to the heading **Tour Alcatraz Island**, as shown below in bold:

```
<h3 id="alcatraz">Tour Alcatraz Island</h3>
```

2. Add an ID to the heading **Visit Fisherman's Wharf**:

```
<h3 id="wharf">Visit Fisherman's Wharf</h3>
```

3. Add one more ID to the heading **Ride a Bike around San Francisco**:

```
<h3 id="biking">Ride a Bike around San Francisco</h3>
```

4. You can link to any ID within the page, by making a link's href equal to the ID. Above the **Things to Do in San Francisco** heading, next to **Jump to info on**, wrap each item with an anchor tag and href for the ID of the section we want to jump to:

```
<p><em>Jump to info on::</em> <a href="#alcatraz">Alcatraz</a> | <a href="#wharf">Fisherman's Wharf</a> | <a href="#biking">Bike Ride</a></p>
```

The **#** in the href tells the browser to find an element with that ID and scroll up or down to it.

5. Save the file and preview **san-francisco.html** in a browser.

6. Above the **Things to Do in San Francisco** heading, click the links to the right of **Jump to info on:** to test them out. If the links do not work as expected, double-check your code. Make sure the **anchor's href** and the **ID** are typed the same and have no spaces or special characters.

7. The page jumps down to specific content, but it would also be nice to be able to jump back up to the top of the page. Let's code that. Return to **san-francisco.html** in your code editor.

8. Below the **Tour Alcatraz Island** section's **Learn More** link you should see **Back to Top**.

   Wrap that text in an anchor tag with an href that points to **#** like so (also add a class so we can style it later):

   ```
   <a href="#" class="back-to-top">Back to Top</a>
   ```

   You just created a link to an unnamed ID. When the browser encounters this, it jumps to the top of the document by default. Pretty neat trick.

9. Wrap the following three remaining **Back to Top** text phrases with the same exact `<a href="#" class="back-to-top">Back to Top</a>` code:

   - At the end of the **Fisherman's Wharf** content.
   - At the end of the **Blazing Saddles** content.
   - And all the way down at the end of the **San Francisco Travel Tips** (just above the **footer**).

10. Save the file and reload your page in a browser to test out all the **Back to Top** links!

Styling the "Learn More" & "Back to Top" Links

1. We added classes to our links, so let's style them. At the bottom of the **style** tag add these new rules:

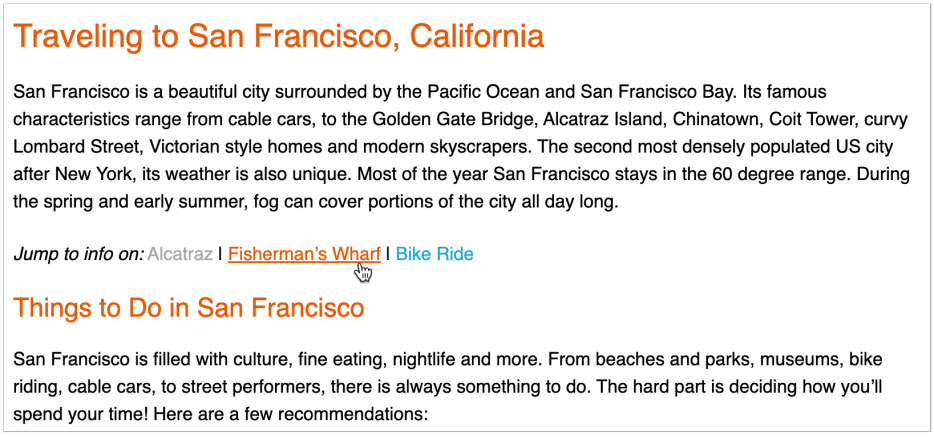
   ```
      .learn-more {
         margin-right: 20px;
      }
      .back-to-top {
         color: #888;
       }
   </style>
   ```

2. Save the file and reload your page in a browser to see:

   - The **Learn More** link has space to the right so **Back to Top** isn't so close.
   - THe **Back to Top** is now gray.
   - In the next exercise we'll finish styling links, which is why we didn't set a color on the **Learn More** link (and we'll remove underlines from all links).

3. If you finish early, check out the **Spambot-Resistant Email Link** bonus exercise at the end of this book for more fun with links.

Exercise Preview



Exercise Overview

In this exercise you'll learn how to style links. The anchor tag has built-in style and functionality. When you code a link in HTML, the browser will render it with blue, underlined text. As a user clicks a link (for the brief moment the pointer is pressed down) the color will change to red. After a link has been visited, the browser changes the color to purple. These styles offer usability clues: "I'm a link, you can click on me" or "You already visited that page". The browser's default colors and underline do not suit every design, so we can change them.

1. If you completed the previous exercise, **san-francisco.html** should still be open, and you can skip the following sidebar. If you closed **san-francisco.html**, re-open it now from the **Revolution Travel** folder. We recommend you finish the previous exercises (3B–3D) before starting this one. If you haven't finished them, do the following sidebar.

If You Did Not Do the Previous Exercises (3B–3D)

1. Close any files you may have open.
2. On the **Desktop**, go to **Class Files > Web Dev Class**.
3. Delete the **Revolution Travel** folder if it exists.
4. Duplicate the **Revolution Travel Ready for Styling Links** folder.
5. Rename the folder to **Revolution Travel**.

Creating a New Rule for the Anchor Tag

Let's begin by creating a new, default style for all links on the page. Links are coded with the anchor tag <a>, so we can target them with a tag selector.

1. In the styles at the top of **san-francisco.html** add the following new rule just below the rule for **p**, like so:

```
p {
   line-height: 24px;
   margin-bottom: 22px;
}
a {
   color: #13aad7;
   text-decoration: none;
}
```

2. Save the file and preview it in a browser to see that the links should now be a lighter blue that matches the logo. We've also removed the underlines that links have by default.

3. Click on a couple of the navigation links, like **Tour Packages** or **Travel Planning**. Hit the browser's **Back** button each time to return to the San Francisco page.

4. Notice that the link style on **san-francisco.html** is always blue with no underline. Let's see how to style specific states of the anchor tag. Leave this page open in the browser so you can reload the page as you work.

Pseudo-Classes

1. Return to **san-francisco.html** in your code editor.

2. As shown below, add the **:link** pseudo-class to the anchor tag rule you just wrote:

```
a:link {
   color: #13aad7;
   text-decoration: none;
}
```

   NOTE: We added a **pseudo-class** to the **a** tag, targeting specific functionality of that element. Pseudo-classes begin with a **colon** (:) and are written next to the element you want to target, with no space between the element and the pseudo-class.

3. Save the file and preview it in a browser.

4. Notice that links you already visited are now purple (the default color for visited links). That's because the **:link** pseudo-class tells the browser to only style links before they have been visited. Let's create our own style for visited links to override the default purple color.

5. Return to **san-francisco.html** in your code editor.
6. Just below the **a:link** rule, add the following bold rule:

```
a:link {
   color: #13aad7;
   text-decoration: none;
}
a:visited {
   color: #aaa;
}
```

7. While we're here, let's add another rule to set the style of the link when a visitor presses down on the link. Below the **a:visited** rule, add the following bold rule:

```
a:visited {
   color: #aaa;
}
a:active {
   color: #2d4;
}
```

8. Save the file and preview it in a browser.
9. The visited links should now be gray. Click and hold on any link to see that the active color changes to green.
10. If you were taken to another page, hit the browser's **Back** button to return to the San Francisco page.
11. Let's create a style for when a user is hovering over a link. Return to **san-francisco.html** in your code editor.
12. Below the **a:visited** rule, add the following bold rule:

```
a:visited {
   color: #aaa;
}
a:hover {
   color: #e45c09;
   text-decoration: underline;
}
a:active {
   color: #2d4;
}
```

13. Save the file and preview it in a browser.
14. Hover over the links to see they become orange (which matches the logo) and underlined.

Order Matters

We had you write the anchor rules in a specific order, because that order is important. Let's investigate.

1. Return to **san-francisco.html** in your code editor.
2. Cut and paste the **a:hover** rule above the **a:visited** rule, so you end up with the following:

```
a:link {
   color: #13aad7;
   text-decoration: none;
}
a:hover {
   color: #e45c09;
   text-decoration: underline;
}
a:visited {
   color: #aaa;
}
a:active {
   color: #2d4;
}
```

3. Save the file and preview it in a browser.
4. Hover over some links and notice that when hovering over **visited** links, the underline appears, but the text remains gray instead of turning orange. Why is this?

   Browsers read rules from the top down. All of these rules have the same level of specificity, so the color for :visited is overriding the color for :hover. The order of link styles in the code matters! When out of order they may not work properly. The correct order is:

   - a:link
   - a:visited
   - a:hover
   - a:focus (We'll cover this a little later in this exercise.)
   - a:active

   A mnemonic that may help you to remember the order is **Lord Vader Hates Furry Animals**. Some people use LoVe HAte, although that doesn't include focus.

5. Return to your code editor and undo the change, so your stack of link styles looks like this:

```
a:link {
   color: #13aad7;
   text-decoration: none;
}
a:visited {
   color: #aaa;
}
a:hover {
   color: #e45c09;
   text-decoration: underline;
}
a:active {
   color: #2d4;
}
```

6. Let's make the code a little simpler. The :link pseudo-class does not really need to be stated directly. We can get the same results if we target all links/states and then add the rules for the specific :visited, :active, and :hover states. Streamline the first link style by deleting the **:link** part, as shown below:

```
a {
   color: #13aad7;
   text-decoration: none;
}
a:visited {
   color: #aaa;
}
a:hover {
   color: #e45c09;
   text-decoration: underline;
}
a:active {
   color: #2d4;
}
```

7. We don't actually want to change the active color to green. Instead, let's just remove the underline that is added by the hover state. Delete color from the **a:active** rule and add the following code shown in bold:

```
a:active {
   text-decoration: none;
}
```

8. Save the file and preview it in Chrome. We want to use Chrome so we can test something specific.

9. In Chrome, do the following:
   - Hover over some links (visited or not) and notice the orange hover color is working properly again.
   - Press down on a link to see the active appearance (the underline should be removed). Then click the browser's **Back** button to go back to the San Fran page.
   - Reload the page (san-francisco.html).
   - Press the **Tab** key and notice that an outline will appear around a link (which should be the logo image).
   - Hit **Tab** again several times to see the outline appear around the text links. Notice that the link with the outline doesn't look any different, except for the outline. This is called the **focus** state, and we can style it as well.

The Focus Pseudo-Class & Grouped Selectors

1. Return to your code editor.
2. We could make a whole new appearance and new CSS rule for the **focus** state (like we have for the visited, hover, etc. states), but we typically don't want to have to design yet another state for such a fleeting appearance most people barely notice. So let's just make the focus state look the same as the hover state. In CSS you can apply a rule to multiple elements or states. For example if you want to style h1 and h2 tags, you'd write the selector as **h1, h2** (just like you write lists in English). After **a:hover**, add a **comma** and **a:focus** as shown below in bold:

```
a:hover, a:focus {
   color: #e45c09;
   text-decoration: underline;
}
```

3. Make sure you did not miss the comma between **a:hover** and **a:focus** in the code you just edited!
4. Save the file and preview it in Chrome.
5. Hit **Tab** several times until the outline appears around the text links.
   - Now the text link that has focus (the one with the outline) should also be orange with an underline. This helps draw even more attention to it.
   - The outline color doesn't match our design, and it's too close to the text for our liking. Luckily we can style it.

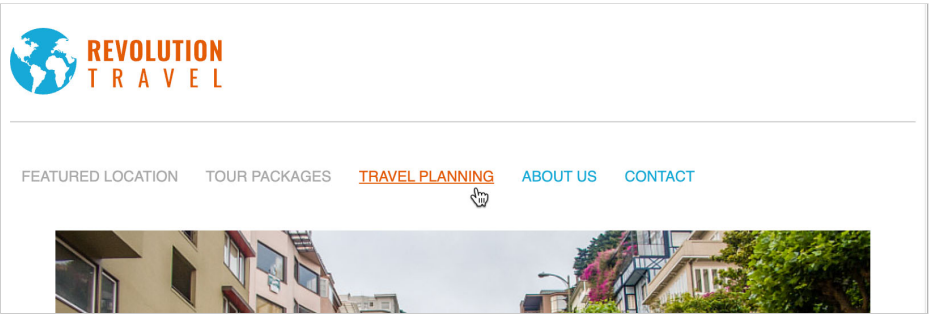
Styling the Link Outline

1. Return to your code editor.
2. In the main link style, add the following properties shown in bold:

```
a {
   color: #13aad7;
   text-decoration: none;
   outline-color: #13aad7;
   outline-offset: 4px;
}
```

3. Save the file and preview it in Chrome.
4. Hit **Tab** several times until the outline appears around the links. Now the link outline color is the same blue as our design, and isn't so close to the text. Better!
5. You can keep **san-francisco.html** open in the browser and the code editor. You'll continue with this file in the next exercise.

Exercise Preview



Exercise Overview

In this exercise you'll start styling the list of links in the nav section so they look like a streamlined navigation bar instead of a clunky list of links. You'll also learn about descendant selectors: a way to target elements based on how elements are nested inside one another.

1. If you completed the previous exercise, **san-francisco.html** should still be open, and you can skip the following sidebar. If you closed **san-francisco.html**, re-open it now. We recommend you finish the previous exercises (3B–4A) before starting this one. If you haven't finished them, do the following sidebar.

If You Did Not Do the Previous Exercises (3B–4A)

1. Close any files you may have open.
2. On the **Desktop**, go to **Class Files > Web Dev Class**.
3. Delete the **Revolution Travel** folder if it exists.
4. Duplicate the **Revolution Travel Ready for Nav Styles** folder.
5. Rename the folder to **Revolution Travel**.

Overriding Default List Styles

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

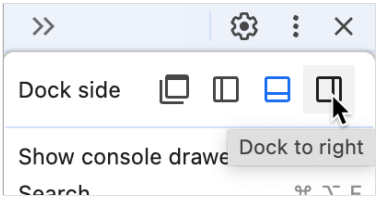
1. In **san-francisco.html**, add the following new rule below the rule for **header**:

```
ul {
   list-style-type: none;
}
```

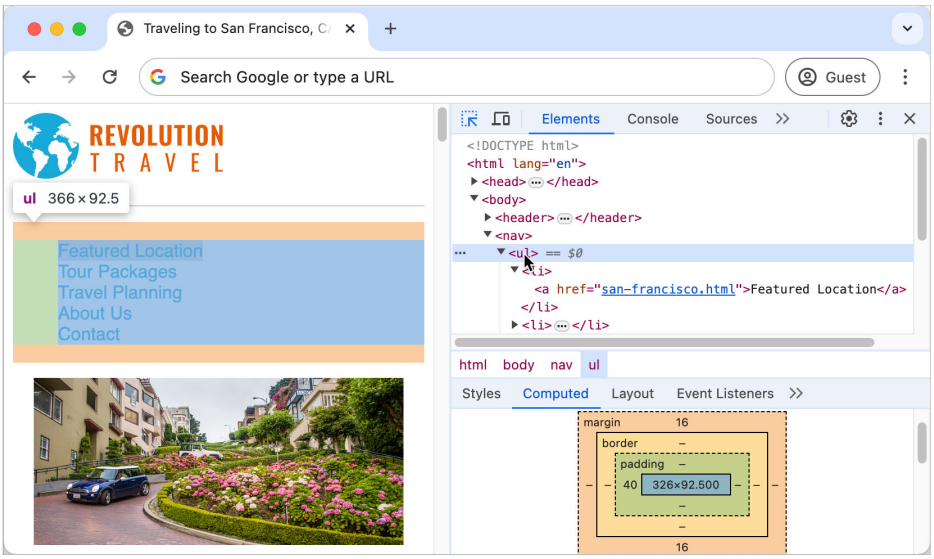
2. Save the file and preview **san-francisco.html** in Chrome. (We'll use Chrome's DevTools.)

3. The nav links no longer have bullets, but let's investigate more. **Ctrl–click** (Mac) or **Right–click** (Windows) on any link in the nav and choose **Inspect**.

4. If the DevTools are docked to the bottom of the browser window, at the top right of the DevTools panel click the ⋮ button and choose **Dock to right** as shown below:



5. In the DevTools, click on (and stay hovering over) the **`<ul>`** tag. As you do so, look at the colored highlights over the content. You should see an orange highlight above and below the list and a green highlight to the left. **Orange** indicates the **margins** of the element, while **green** indicates **padding**. Let's remove all of this space.



6. Leave this page open in Chrome, and return to **san-francisco.html** in your code editor.

7. Add the following property declarations to the **ul** rule:

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

8. Save the file, return to Chrome, and reload the page. Next, we want the links in a horizontal row, rather than the vertical stack we have now. By default, each list item display as a stacked element (called **block**), but we can change that.
9. Return to **san-francisco.html** in your code editor.
10. Add the following new rule below the rule for **ul**:

```
ul {
   list-style-type: none;
   margin: 0;
   padding: 0;
}
li {
   display: inline;
}
```

> **Inline versus Block**
>
> **Block** elements stack on top of one another like a child's building blocks. Paragraphs, headings, section elements, divs, and lists are some common block-level elements that we use on a regular basis.
>
> **Inline** elements are rendered next to one another in a row. Anchors (links), images, and spans are some of the inline elements we've used so far in class.
>
> The difference goes a bit further. Manipulating the box-model is more limited for inline elements. While block-level elements can be controlled by changing the width, height, padding, margin, and border of the element, inline elements are only as wide as their content. They do not respond to CSS changes to their width or height. Margin and padding are also handled differently: margin and padding will only work horizontally on inline elements (padding on the top and bottom of inline elements will be present, but often bleed into other lines above and below these elements).

11. Save the file, return to Chrome, and reload the page. We need more space between the links, but first let's take care of one other thing.
12. Close the DevTools panel by clicking the **X** at its top right.

Using Descendant Selectors

The rules we just wrote (**ul** and **li**) currently work fine for this page because it contains only one list. But if we added another unordered list, it would also be affected by these rules (which are really only appropriate with the site's navigation list, not all lists). Let's change these rules so they only apply to the nav.

1. Return to **san-francisco.html** in your code editor.
2. Scroll down to the **nav** (just after the start of the **body** tag):

```
<nav>
   <ul>
      <li><a href="san-francisco.html">Featured Location</a></li>
      CODE OMITTED TO SAVE SPACE
   </ul>
</nav>
```

   Notice that the `<nav>` contains a `<ul>`, which in turn contains numerous `<li>`. We can use **nav** in our selectors to make sure our **ul** and **li** rules will only affect the ul and li tags within this section of the page.

3. Scroll up to the styles and add **nav** before the **ul** and **li** selectors, as shown below in bold. (Be sure to type a space after nav!)

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

   These selectors are officially known as **descendant selectors**. The **ul** sits inside the **nav**, so the nav is a parent (or ancestor), and the ul is its child (or descendant). The logic holds true for the **li** as well. The **li** is the grandchild of the **nav** (a more distant descendant, but a descendant nonetheless). A space between the parent and the descendant is required.

4. Save the file, and reload the page in Chrome to see nothing should have changed (the code still works). This code change was important to make, because we will have normal lists on other pages and we want those to look like normal bulleted lists.
5. Hover over the links, paying attention to how close you have to be to the text until you see the pointer cursor which indicates the link is clickable. The clickable area is defined by the text in the anchor, but we can make the clickable area larger so these links will be even easier to to click.

Increasing the Clickable Area of the Navigation Links

1. Return to **san-francisco.html** in your code editor.

2. Let's temporarily add a background color to the links so we'll be able to better see what's going on. Below the **nav li** rule, add the following new rule:

```
nav a {
   background-color: #000;
}
```

3. Save the file and reload the page in Chrome to see the black background behind the navigation links. This is the clickable area!

4. Return to **san-francisco.html** in your code editor.

5. Add the some padding, as follows:

```
nav a {
   background-color: #000;
   padding: 10px;
}
```

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

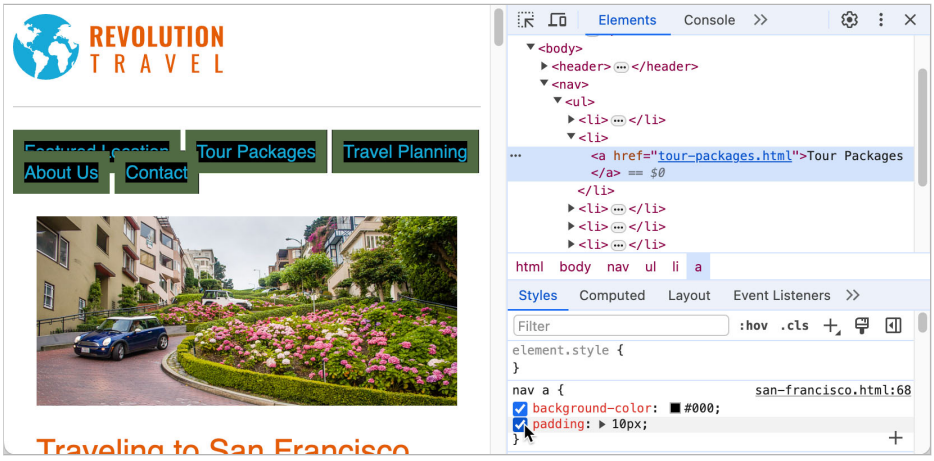
   - Notice there is more space within the black background of each nav link.
   - Hover over the links in the nav to see that the clickable area is bigger as well.

7. There's a CSS rendering issue we need to investigate using the DevTools. **Ctrl–click** (Mac) or **Right–click** (Windows) on a link in the nav and choose **Inspect**.

8. Resize the window (or the DevTools panel) to force the line of navigation links to break onto two lines.

9. Notice how the bottom line of links covers over the top row, which is not good!

10. In the DevTools, click on an **<a>** tag in the nav, and take a look at the **Styles** tab. Click the checkbox next to **padding** to disable and enable this property a few times:



The top and bottom padding may not work as you'd expect. Adding padding makes the links get farther apart horizontally, but not vertically.

11. Make sure **padding** is enabled (checked on) before continuing.

12. Also in the DevTools styles, click the checkbox next to **background-color** to disable and enable this property a few times. Notice that the color of the links in the second line covers over the links in the top line.

    The links are acting this way because anchor elements are displayed inline. Inline elements are meant to be unobtrusive in the layout and do not respond to height or vertical modifications in padding or margin.

13. Return to **san-francisco.html** in your code editor.

14. In the **nav a** style, add the **display** property as shown below:

```
nav a {
   background-color: #000;
   padding: 10px;
   display: inline-block;
}
```

    Using **inline-block** is a way to combine the best of both worlds: have block-level control over the padding (so it works more as you'd expect) but have the links display inline, horizontally across the top of the page.

15. Save the file and reload the page in Chrome. That's better! The second line of links no longer covers over the top line.

16. Return to **san-francisco.html** in your code editor.

17. Now that we've seen how inline-block and padding work we can remove the temporary background color. Delete **background-color** from the **nav a** style, so you end up with the following:

```
nav a {
   padding: 10px;
   display: inline-block;
}
```

18. While we're here, let's also make the text smaller and uppercase. Add the following code shown in bold:

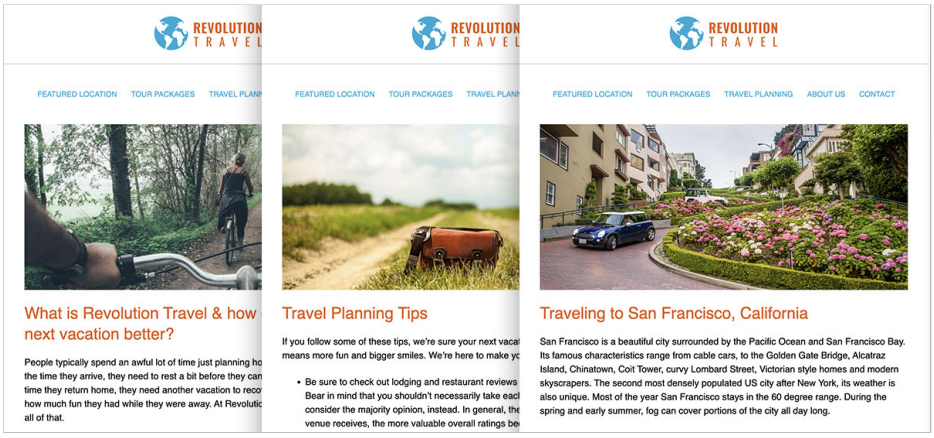
```
nav a {
   padding: 10px;
   display: inline-block;
   font-size: 13px;
   text-transform: uppercase;
}
```

19. Save the file and reload Chrome to see the updated nav styling.

    There's still more we want to do to the nav, such as making the color of all links the same (we don't need to indicate which pages have been visited), but we'll do that in the next exercise.

20. You can keep **san-francisco.html** open in the browser and the code editor. You'll continue with this file in the next exercise.

Exercise Preview



Exercise Overview

In this exercise you'll finish styling the Revolution Travel site and learn about the following concepts:

- As you finish styling the :visited and :hover states on the navigation, you'll learn how specific CSS styles override general CSS styles… a concept called specificity.
- CSS styles can be embedded into individual pages (as we've been doing) or shared across multiple pages (as you'll learn about in this exercise). Styles are often shared across pages to ensure consistent styling and to changes easier (one change updates all linked pages). It also reuses the code, making the website load faster.

1. If you completed the previous exercise, **san-francisco.html** should still be open, and you can skip the following sidebar. If you closed **san-francisco.html**, re-open it now. We recommend you finish the previous exercises (3B–4B) before starting this one. If you haven't finished them, do the following sidebar.

If You Did Not Do the Previous Exercises (3B–4B)

1. Close any files you may have open.
2. On the **Desktop**, go to **Class Files > Web Dev Class**.
3. Delete the **Revolution Travel** folder if it exists.
4. Duplicate the **Revolution Travel Ready for Shared CSS** folder.
5. Rename the folder to **Revolution Travel**.

Customizing the Visited & Hover States of the Main Nav

1. Preview **san-francisco.html** in a browser and notice the following:

   We like the link, visited, and hover styles, but we want to make some minor tweaks specifically to the links in the main site navigation. If you've visited other pages, those links in the main navbar (at the top of the page) should be gray. If you haven't, click a link and then use the back button to return to **san-francisco.html**. Now you should see a gray link.

   The links for pages you have not visited should be blue. We don't think it's necessary to mark visited links in the main site nav, so let's set the nav links to always be blue.

2. Return to **san-francisco.html** in your code editor.

3. In the previous exercise we saw that we can style just the elements that are within a parent element. Let's style the link's visited state, but only for links in the nav section. Below the rule for **nav a** add the following new rule:

```
nav a {
    CODE OMITTED TO SAVE SPACE
}
nav a:visited {
   color: #13aad7;
}
```

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

   - All the nav links should now be blue (as we want).
   - Hover over each link in the nav, and notice that the orange hover color does not work on the visited links. Why is that?

     The **a:hover** rule (which makes hovers orange) targets all links. The new **nav a:visited** rule is more specific because it only targets links in the **nav**. CSS specificity is how browsers decide which rule to apply. More specific rules override less specific rules (regardless of their order in the code). Adding the additional tag selector (**nav**) increases the specificity, so our **nav a:visited** rule overrides the **a:hover** rule. To style the hover state of links in the nav, we'll need to make a rule with a greater specificity than the **a:hover** rule.

More About CSS Specificity

- How specificity is calculated: **css-tricks.com/specifics-on-css-specificity**
- Specificity calculator: **codecaptain.io/tools/css-specificity-calculator**

5. Below the **nav a:visited** rule, add the following new rule:

```
nav a:hover {
   color: #e45c09;
}
```

6. Save the file and reload the page in the browser.
7. Hover over each link in the nav and notice the orange hover works again!
8. Return to **san-francisco.html** in your code editor.
9. We don't want the underline, so to finish our link styling, add the following property to the **nav a:hover** rule:

```
nav a:hover {
   color: #e45c09;
   text-decoration: none;
}
```

10. While we're here, we also want this to be the appearance of the focus state. Add a comma and **nav a:focus** to the selector. Do not miss the comma!

```
nav a:hover, nav a:focus {
   color: #e45c09;
   text-decoration: none;
}
```

11. Save the file and reload the page in the browser.
12. Hover over the links in the nav and notice that only the color changes. Simple and clean, just what we wanted.

    If you want to test the focus state, press **Tab** a few times until a nav link is highlighted and you should see the text turn orange.

Creating & Linking to an External Style Sheet

Let's move these embedded styles into an external style sheet so they can be shared across the entire site.

1. Return to **san-francisco.html** in your code editor.
2. In **san-francisco.html** select **all** the rules inside the `<style></style>` tag. Start with the **body** rule and go all the way down to—and include—the closing curly brace of the **.back-to-top** rule.
3. Cut the rules (**Cmd–X** (Mac) or **Ctrl–X** (Windows)).
4. Open **main.css** in your code editor (it's in the **Revolution Travel** folder). This is a blank file we're providing that should not have any code yet.

5. Paste (**Cmd–V** (Mac) or **Ctrl–V** (Windows)) the rules.
6. Save **main.css** and keep it open.
7. Return to **san-francisco.html** in your code editor. Save the file.
8. Take a moment to preview **san-francisco.html** (NOT main.css) in a browser to note what it looks like without its CSS.

   NOTE: Previewing a CSS file only shows the lines of code—with none of the HTML content it styles. So you don't accidentally preview main.css, we recommend leaving san-francisco.html open in your browser so you can reload the page to test the code as you continue.
9. Return to **san-francisco.html** in your code editor and delete the empty style tags:

   ```
   <style>

   </style>
   ```
10. Where the style tags used to be, type the following line of code:

    ```
    <link rel="stylesheet" href="main.css">
    ```
11. Note the required attributes of the **link** tag:
    - The **rel** attribute states the relationship of the linked document… it's a style sheet!
    - The **href** provides the hyperlink reference to the style sheet file.
12. Save the file and reload your page in the browser to see how that one little line of code pulls in all the styles in the linked file. Your styles can now be shared across the other pages in the site as well.

    NOTE: Like images, the CSS file must be uploaded along with the HTML pages to your remote web server in order for visitors to see the styles.

Linking Styles to Other Pages

Now that we have styles in an external style file, we can share them across the entire site. Each page will have to be linked to the CSS file, but then we'll be able to edit styles for the entire site in one convenient file!

1. Return to your code editor.
2. Copy (**Cmd–C** (Mac) or **Ctrl–C** (Windows)) the code you just wrote in **san-francisco.html**:

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

3. There should be six provided HTML files located in whichever Revolution Travel folder you are working in. (You may have already opened and edited the contact page in an earlier exercise.) There should be no styles—embedded or linked—in these files as of yet.
4. Open **about.html** in the code editor.
5. Paste the link code into the **head** tag as shown:

```
<head>
   <meta charset="UTF-8">
   <title>About Revolution Travel - Revolution Travel</title>
   <link rel="stylesheet" href="main.css">
</head>
```

6. Save the file and then close the file.
7. With the code still copied on your clipboard, paste the code into the following files. Be certain to paste the link code after the **title** and above the **</head>** tag, as shown in the example above.
   - contact.html
   - index.html
   - travel-planning.html
8. Make sure you saved all the files and closed them.
9. Open **tour-packages.html**

   Notice this file has some embedded styles for things that only appear on this page.

   Paste the link code above the **<style>** as shown below:

```
<head>
   <meta charset="UTF-8">
   <title>Tour Packages - Revolution Travel</title>
   <link rel="stylesheet" href="main.css">
   <style>
      .featured-package {
```

10. Save the file and then close it.
11. Go ahead and preview any HTML page (NOT main.css) in the browser and test out the links in the main site navigation. Enjoy the look and feel of the shared style sheet. Keep the browser open to any of these pages so you can reload the browser to test the code as you continue.

Linked Styles vs. Embedded Styles

Embedded styles can happily coexist with linked styles from an external style sheet. It's not an either/or proposition.

Linked style sheets allow you to maintain a cohesive appearance for elements across a site, but there may be elements on a single page that are not repeated elsewhere (so there's no need to share those styles).

Embedded styles may also be used to override rules from linked styles on a page-by-page basis. To ensure that embedded styles will override rules in a shared style sheet, place the embedded styles below the link to the shared style sheet.

Centering the Logo & Nav Content

1. Let's center the logo in the header. Open **main.css** in your code editor (if it's not already open).
2. Find the rule for **header** and add the following property declaration (in bold):

```
header {
   border-bottom: 1px solid #ccc;
   padding-top: 20px;
   padding-bottom: 20px;
   text-align: center;
}
```

The Text-Align Property

Although images are obviously not text, they are **inline** elements. Inline elements like text, anchor tags, and images will be placed horizontally in a line, rather than stacking like blocks (as paragraphs, headings, and divs do). All inline elements can be aligned horizontally using CSS's text-align property.

3. Save the file.
4. Reload any of the site's pages in the browser to see the centered logo.

   One change to our CSS file is shared across all the HTML pages that link up to it. Let's center the content in the navigation the same way.

5. Return to **main.css**.

6. Below the **header** rule, add the following new rule:

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

7. Save the file, return to the browser, and reload the page to see how the logo and your nav center up in the browser window.

Fine-Tuning Margins & Padding

We want the border below the header section to go all the way to the edge of the browser window. The default of all major browsers is to render the body element with about 8px of margin. This puts space between the edge of the browser window and the content of the page. Let's remove that space.

1. Return to **main.css** in your code editor.
2. Go to the very top and edit the rule for **body** as follows:

```
body {
   font-family: sans-serif;
   margin: 0;
}
```

NOTE: Are you surprised that we're adjusting margin instead of padding? Typically padding is space inside an element, and margin is space outside. By default the body element has some margin, but no padding. Because the body fills the browser window it's unique. Its margins and padding both have to fit inside the window.

3. Save the file and reload any of the site's pages in a browser to see the changes to the margin around the content of the page.

   The border now touches the edges of the browser window.

4. Go to the **Travel Planning** page and notice the list items don't have nice spacing like paragraphs, because we have not styled them.
5. Return to your code editor and find the **p** rule. We want to use the same styling for list items, so add a comma and **li** as shown below. Do not miss the comma!

```
p, li {
   line-height: 24px;
   margin-bottom: 22px;
}
```

6. Save the file and reload the **Travel Planning** page in a browser to see the list items now look nicer.

Optional Bonus: Styling the Footer Content

Let's make the copyright less prominent than the other text. We'll write a style to mute the copyright text color and also move it away from the other page content.

1. Below the **header** rule, add the following new rule:

```
footer {
   color: #999;
   font-size: 12px;
   text-transform: uppercase;
}
```

2. Save the file and reload the **About Us** page to see the newly styled copyright text in the footer. Let's place a subtle border between the copyright content in the footer and the page content.

3. Return to **main.css** in your code editor and continue to edit the rule for footer to add the following new border property:

```
footer {
   CODE OMITTED TO SAVE SPACE
   text-transform: uppercase;
   border-top: 1px solid #ccc;
}
```

4. Save the file and reload the **About Us** page to see the footer's border. Let's add more space between the border and the footer's copyright text.

5. Return to **main.css** in your code editor and edit the footer to add the following padding property:

```
footer {
   CODE OMITTED TO SAVE SPACE
   border-top: 1px solid #ccc;
   padding-top: 20px;
}
```

6. Save the file and reload the **About Us** page to see the enhanced separation of the footer content. Let's add the same border between page topics.

7. Edit the **h2** rule (near the top) to add the following properties (in bold):

```
h2 {
   font-size: 23px;
   border-top: 1px solid #ccc;
   padding-top: 20px;
   margin-top: 40px;
}
```

8. Let's also add more space above the h3:

```
h3 {
   font-size: 18px;
   margin-top: 40px;
}
```

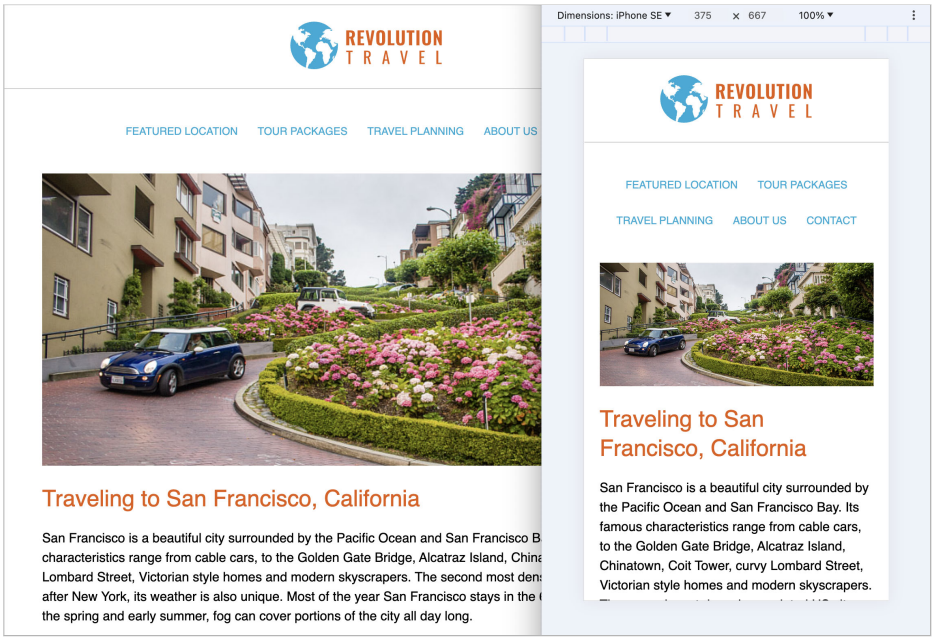
9. Save the file and reload the **About Us** page.

10. You can keep these files open in the browser and the code editor. You'll continue with this site in the next exercise.

How to Create a Brand New CSS File In Most Code Editors:

1. Go to **File > New File**.
2. Save the file as **main.css** (or another name of your choosing), and you're all set to start coding styles.

Exercise Preview



Exercise Overview

The viewport meta tag controls how a webpage is displayed on a mobile device. Without the viewport set, mobile devices will render the page at a typical desktop screen width, scaled to fit the screen. Setting a viewport gives control over the page's width and scaling on different devices.

1. If you completed the previous exercise, you can skip the following sidebar. If you closed **san-francisco.html**, re-open it now. We recommend you finish the previous exercises (3B–4C) before starting this one. If you haven't finished them, do the following sidebar.

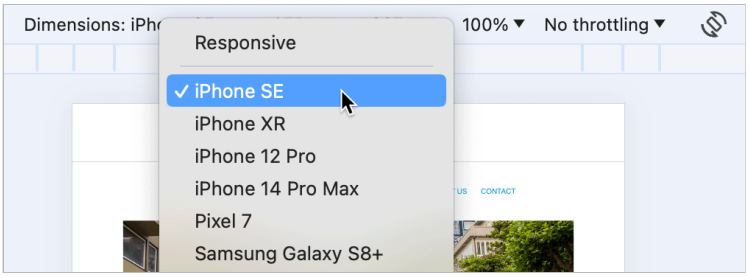
If You Did Not Do the Previous Exercises (3B–4C)

1. Close any files you may have open.
2. On the **Desktop**, go to **Class Files > Web Dev Class**.
3. Delete the **Revolution Travel** folder if it exists.
4. Duplicate the **Revolution Travel Ready for Viewport** folder.
5. Rename the folder to **Revolution Travel**.

Disabling Mobile Browser Text Size Adjustment

While the webpage responds properly in a desktop browser, we need to test how it will work on a mobile device. Thankfully, Chrome DevTools has a mobile simulator.

1. Preview **san-francisco.html** in Chrome.
2. To open the DevTools, **Ctrl–click** (Mac) or **Right–click** (Windows) on the page and choose **Inspect**.
3. At the top left of the DevTools panel, click the **Toggle device toolbar** button to open the mobile simulator.
4. Above the webpage preview, select a device such as the **iPhone SE**:



5. The desktop layout has been scaled down (as you can see in the navigation at the top), but text in the main column is large so what's going on? Some mobile browsers enlarge text they think is too small (if they think it doesn't break the layout). We don't want browsers arbitrarily overriding some font sizes, so let's disable that.
6. Switch back to your code editor.
7. Go to **Revolution Travel > snippets** and open a code snippet we prepared for you, **text-size-adjust.css**.
8. Hit **Cmd–A** (Mac) or **Ctrl–A** (Windows) to select all the code.
9. Hit **Cmd–C** (Mac) or **Ctrl–C** (Windows) to copy it.
10. Close the file.
11. At the top of **main.css**, paste the new code above the **body** rule:

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

12. Save the file.
13. Switch back to Chrome and reload the page.

14. The text is no longer being enlarged, so it looks like the desktop layout scaled down to fit a mobile phone. Now that the browser won't be deciding what text it might enlarge, we must get it to display the layout appropriately for small screens, instead of scaling down the desktop layout.

> **Vendor Prefixes**
>
> The **-moz** and **-webkit** properties are CSS vendor prefixed properties (for Mozilla Firefox and Webkit-based browsers such as Safari) while **text-size-adjust** is the standard CSS property.
>
> Vendor prefixes are non-standard CSS properties that ensure compatibility with a browser during experimental stages of a new CSS feature. They help us use new CSS features before they are fully standardized and supported by all browsers. Eventually we should be able to use only the standard property, but sometimes that can take a long time or it's possible a browser maker might decide to not support the standard CSS property.
>
> The practice of using vendor prefixes is less common today, but a few uses still remain.

The Viewport Meta Tag

Mobile devices assume websites were designed for desktops. They render the site on a large viewport (980px) and scale it down to fit their smaller screen. For responsive sites we must add a **meta** tag to tell it to render the page at the actual pixel width of the device, instead of the default 980px.

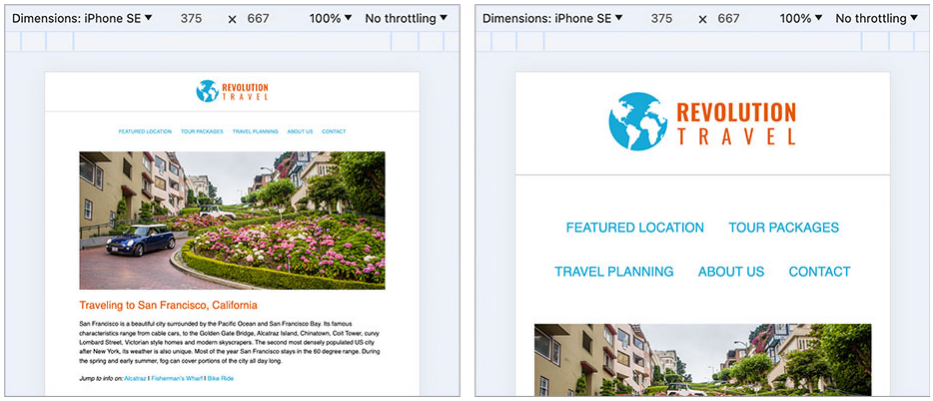
1. Keep the browser open in device mode so we can come back to preview the changes we're about to make.
2. In your code editor, switch to **san-francisco.html**.
3. In the **<head>** tag, add the following bold code:

```
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width">
   <title>Traveling to San Francisco, CA - Revolution Travel</title>
```

This tells mobile browsers to make the width of the **viewport** equal to the width of the **device**. Like other meta tags, it goes in the head tag.

4. Save the file and reload the page in Chrome (which should still be in device mode).

   The layout and text size should now look a lot more appropriate for a small mobile phone! The viewport meta tag does not affect **desktop** browsers, only **mobile** browsers which employ the scaling behavior. As shown below, on the left is how the page looked before the meta tag, and on the right we see it with the meta tag!



5. What happens when we rotate the device to landscape? In Chrome, above the webpage preview, click the **Rotate** button to change the orientation.

   That looks good. The portrait layout is recalculated to show more content in the wider, landscape preview.

6. At the top left of the DevTools panel, click the **Toggle device toolbar** button to close the mobile simulator.

7. Mobile users can pinch to zoom in on the webpage (scale it up). While it's unlikely for mobile browsers to automatically apply a zoom, let's ensure they won't by adding some more code to the viewport meta tag.

   Back in **san-francisco.html**, add the following bold code and don't miss the comma!

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

More About Initial-Scale

Setting **initial-scale=1** sets the page zoom to 100%, or in other words… do not scale the page. For example, 1.5 would zoom in to 150%.

Most mobile browsers set **initial-scale** to **1** by default once you set the viewport meta tag width to **device-width**, but it's a best practice to also specify it in code just in case.

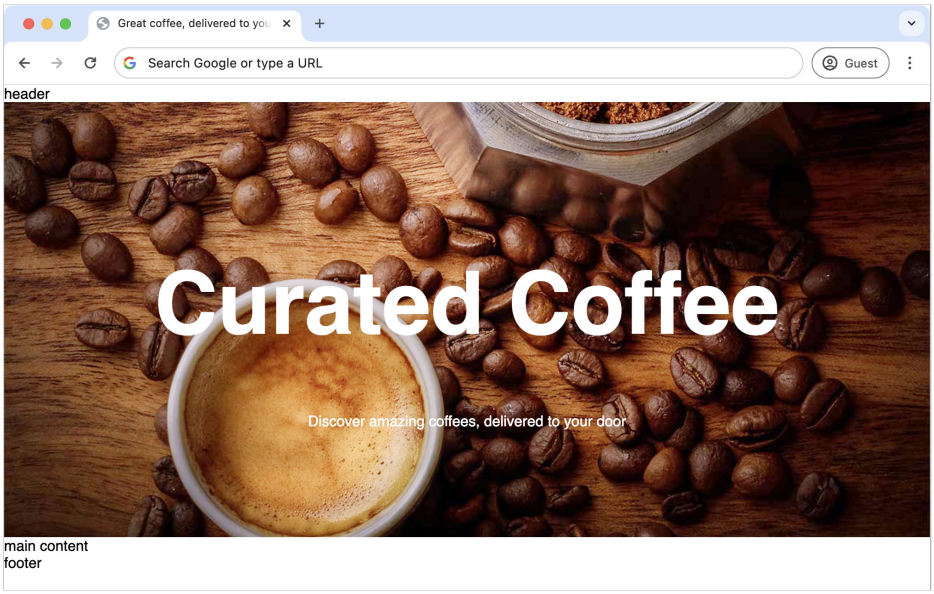
8. Save the file. You shouldn't see a change, but having this code is a best practice.

   Please note that the viewport meta tag needs to be in every HTML file in a website. If you wanted to complete this site, you'd have to copy and paste it into the other .html files. We're done with this site so we're not going to make you do that busy work, but keep that in mind in the future. Make sure you have the viewport meta tag before making all the pages of your site!

   Disabling Zoom

   Mobile browser users can zoom in on a page. Adding **maximum-scale=1** to the viewport meta tag limits how far a user can zoom in. When the maximum-scale and the initial-scale are both set to 1, the user will be unable to zoom. Because this is bad for usability, Safari on iOS ignores this code so users can still zoom. While it's possible to disable page zooming in some browsers, we do not recommend it.

Exercise Preview



Exercise Overview

This is the first in a series of exercises where you'll create a small coffee subscription site. In this exercise you'll start by building one part of the home page. You'll style the hero section where headings overlay an image background, allowing us to take a deeper dive into the CSS background-image property.

Getting Started

1. In your code editor, close any files you may have open.
2. For this exercise we'll be working with the **Hipstirred** folder located in **Desktop > Class Files > Web Dev Class**.

   TIP: It's beneficial to see the entire website folder as you work. Many code editors allow you to do so. If you're in Visual Studio Code, go to **File > Open Folder**, navigate to **Class Files > Web Dev Class > Hipstirred** and hit **Open** (Mac) or **Select Folder** (Windows).
3. In your code editor, open **index.html** from the **Hipstirred** folder.
4. Title the page by editing the code as follows:

```
<title>Great coffee, delivered to your door - Hipstirred</title>
```

Coding Up the Sections

The home page will feature a header, a prominent hero section where the headings will overlay an image background, a main content section that describes the product, and a footer that will contain the copyright and social media links.

A hero section of a webpage is the most prominent content a user will first see. A hero image often consists of a large image and text, and its purpose is to give an overview of what the page/website is about.

1. Inside the body tag, add the following sectioning tags with placeholder content (highlighted in bold):

```
<body>
   <header>header</header>
   <div class="hero">
      <h1>Curated Coffee</h1>
      <p>Discover amazing coffees, delivered to your door</p>
   </div>
   <main>main content</main>
   <footer>footer</footer>
</body>
```

   NOTE: We're using a **div** element for the hero section. There's no semantic tag to describe this kind of content, so a div the most appropriate.

2. Save the file and preview **index.html** in Chrome (we'll be using Chrome's DevTools). There's not much to see, but now we can start adding real content and styling these sections, beginning with the hero.

   Leave **index.html** open in Chrome, so you can reload the page to see the changes you make in the code.

3. Return to your code editor.

4. Open **main.css** from the **Hipstirred** folder. The provided file is completely empty. Let's start by declaring some basic font defaults for the page.

5. Create the following new rule in **main.css**:

```
body {
   font-family: sans-serif;
}
```

6. The default of most browsers is to render the body element with 8px of margin. This puts space between the edge of the browser window and the content of the page. Let's remove that, so we'll only have space we choose to add. Add the following new property to the **body** rule:

```
body {
   font-family: sans-serif;
   margin: 0;
}
```

7. Save **main.css**.

8. Return to **index.html** and add a link to **main.css**, as follows:

```
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1">
   <title>Great coffee, delivered to your door - Hipstirred</title>
   <link rel="stylesheet" href="main.css">
</head>
```

9. Save the file and reload the page in Chrome. You should see the text in a sans-serif font.

Creating a Hero Style

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

2. We want to center the text in the hero, make the heading bigger (so we can see more of the background image behind it), add some space inside the div, and change the background color so we can see the size of the section we're working with. Create the following new rules below the **body** rule:

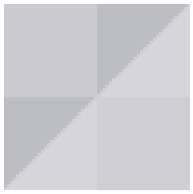
```
h1 {
   font-size: 95px;
}
.hero {
   text-align: center;
   padding: 30px;
   color: #fff;
   background-color: #520;
}
```

3. Save **main.css**.

4. Return to Chrome and reload the page. You should see the headings on a background color. Later we'll need to adjust the space around and between the headings, but for now let's add a background graphic to replace the solid background color.

Adding a CSS Background-Image

Before we add our hero photo, let's see how CSS background images can be used to create a repeating pattern. We'll start with a graphic (shown below) that is a single 60px by 60px square image.



1. Return to **main.css** in your code editor.
2. Add the following new property to the **.hero** rule:

```
.hero {
    CODE OMITTED TO SAVE SPACE
    background-color: #520;
    background-image: url(images/pattern.png);
}
```

NOTE: Notice that we kept the background color? The background image will display on top of the background color so we usually won't see the color. While users are waiting for the image to load, the darker background color will allow people to read the white text.

3. Save **main.css**.
4. Return to Chrome and reload the page. Notice the square graphic appears at its native size, and then repeats (in rows/columns) to fill the entire space of the div.
5. Resize the browser window to see the pattern is cropped by the containing div. The pattern size remains constant, but how many rows and columns of repeats you see is determined by the size of the containing div.
6. Let's switch to our hero image. Return to **main.css** in your code editor.
7. In the **.hero** rule, change the background image to the hero photo:

```
.hero {
    CODE OMITTED TO SAVE SPACE
    background-image: url(images/hero.jpg);
}
```

8. Save **main.css**.

9. Return to Chrome, reload the page, and note the following:

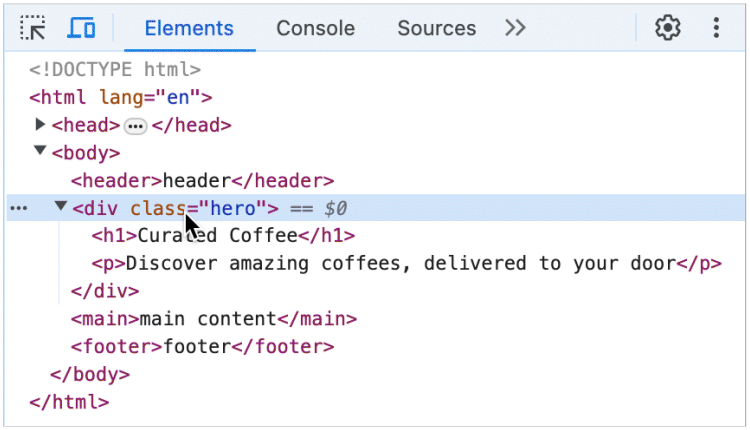
   This photo is large, so only the top left of the photo is visible. Here's what the entire image looks like, so you can keep it in mind as you work through this exercise:

   The photo currently appears at its native size and then repeats, but you'll only be able to see the repeat (on the right side) if your monitor is wide enough.

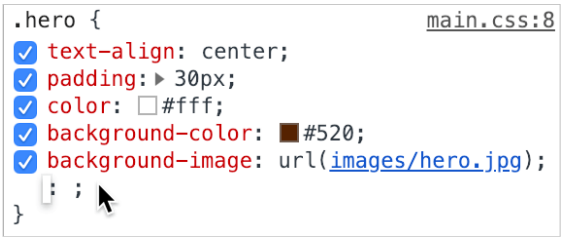
Modifying Background-Position

1. We can adjust the position of the background image within the hero div. Let's use DevTools to see the changes as we make them. Still in Chrome, **Ctrl–click** (Mac) or **Right–click** (Windows) on the hero section and choose **Inspect**.

2. In the Elements panel, select the **hero** div (you'll see it highlighted in the browser):



3. You should now see the **.hero** rule in the **Styles** tab on the right side of the DevTools window. Click at the bottom of the stack of property declarations for the **.hero** rule. The cursor should be blinking, indicating that you're ready to start typing to add a new property declaration:



4. Type **background-position: right bottom** into the DevTools:

5. You can see the image move behind the headings in the hero div as you modify the CSS in the DevTools. Excellent!

   NOTE: The background-position property takes two values: the X position (horizontal offset) and the Y position (vertical offset) separated by a space. The default value is **left top**. Values can be keywords **top**, **bottom**, **left**, **right**, and **center**, but you can also use absolute pixel values or percentages (which we'll experiment with a bit later).

6. Click on the **right bottom** value of the **background-position** property and change it to **center** (which will center the image horizontally and vertically).

Proportional Sizing for the Hero Section

The size of the hero div defines the size of the "window" through which we can see the photo. With a fixed 30px of padding, the amount of padding is not proportional to the screen size. 30px is a lot on a mobile phone that's 320px wide, but not much on a desktop that's 1440px wide. Let's switch to a flexible amount instead.

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

2. In the **.hero** rule, change the padding to **15%**. By using a percent instead of pixels, it will better adapt to different size screens.

```
.hero {
   text-align: center;
   padding: 15%;
   color: #fff;
   background-color: #520;
   background-image: url(images/hero.jpg);
}
```

3. While we're here, set the background-position to **center** as follows:

```
   background-image: url(images/hero.jpg);
   background-position: center;
}
```

4. Save **main.css**.

5. Return to Chrome, reload the page, and resize the browser window (making it narrow and wide) and notice that the padding gets smaller when the window is narrow, and bigger when the window is wide.

   This size and position looks pretty good, but the background image still repeats on wide monitors.

Background-Repeat

1. In Chrome, **Ctrl–click** (Mac) or **Right–click** (Windows) on the hero section and choose **Inspect**.

2. In the Elements panel, select the **hero** div.

3. In the **Styles** tab, find the **.hero** rule.

4. Click on the **center** value of the **background-position** property and change it to **400px 150px**

   In case you couldn't see the background repeating before, now that we've offset the X and Y position, you can certainly see that the background-image is repeating (like a pattern). We do not want the photo to repeat, so let's modify this in our CSS file. We'll edit the position a bit later.

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

6. Add the following new property declaration to the **.hero** rule:

```
    background-image: url(images/hero.jpg);
    background-position: center;
    background-repeat: no-repeat;
}
```

NOTE: You can also set **background-repeat** to **repeat-x** (creates a row of repeating images horizontally) or **repeat-y** (creates a column of repeating images vertically).

7. Save **main.css**.

Setting the Background-Size

Currently the background image does not scale. When the window is wider than the native 1280 pixels of our image, we run out of image and see the solid background color of the hero div. Let's use background-size to tell the image to scale up or down to always fill the hero div.

2. Add the following new property declaration to **.hero**:

```
    background-image: url(images/hero.jpg);
    background-position: center;
    background-repeat: no-repeat;
    background-size: 100% auto;
}
```

Background-size accepts a **width** (first value) and **height** (second value) separated by a space. We can use fixed pixel sizes for the image or percentages that correspond to the containing element. Here, we're setting the background image to be 100% of the width of the hero div and **auto** for the height to maintain the aspect ratio.

3. Save **main.css**.

4. Return to Chrome and reload the page. Resize the browser window to see how the background scales at different widths. It looks great on a wide window, but when the browser window is narrow, the image isn't tall enough to fill the hero area, so we see some of the solid background color at the top and bottom.

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

6. Let's try to fill the height of the hero area. Edit the background-size value:

```
    background-size: 100% 100%;
}
```

7. Save **main.css**.

8. Return to Chrome and reload the page. Resize the browser window again. Yikes, the image now squishes to fit the hero area! Not the look we want.

9. Return to **main.css** in your code editor.
10. Let's try one of the background size keyword values. Change the background-size value to cover:

```
    background-size: cover;
}
```

11. Because of this sizing, we'll never see the background image repeat. So delete the **background-repeat** property as it's no longer needed. Your code should now be:

```
.hero {
    text-align: center;
    padding: 15%;
    color: #fff;
    background-color: #520;
    background-image: url(images/hero.jpg);
    background-position: center;
    background-size: cover;
}
```

12. Save **main.css**.
13. Return to Chrome and reload the page. Resize the browser window. This looks nice!

    The **cover** value scales the image up or down (maintaining the aspect ratio) and if necessary it crops off the top/bottom or left/right.

Perfecting the Background-Position

On wide screens we'd like to see more of the cup, so let's tweak the position.

1. Make the browser window wide.
2. **Ctrl–click** (Mac) or **Right–click** (Windows) on the hero section and choose **Inspect**.
3. In the Elements panel, select the **hero** div.
4. Find the **.hero** rule in the **Styles** tab.
5. Click on the **center** value of the **background-position** property and change it to **center 0%**
6. Now highlight just the **0%** value.
7. Hold **Shift** and press the **Up Arrow** key on your keyboard to increase the value **10** percent at a time. Find a value that positions the image nicely behind the header text. Around 70–80% seems to work well.

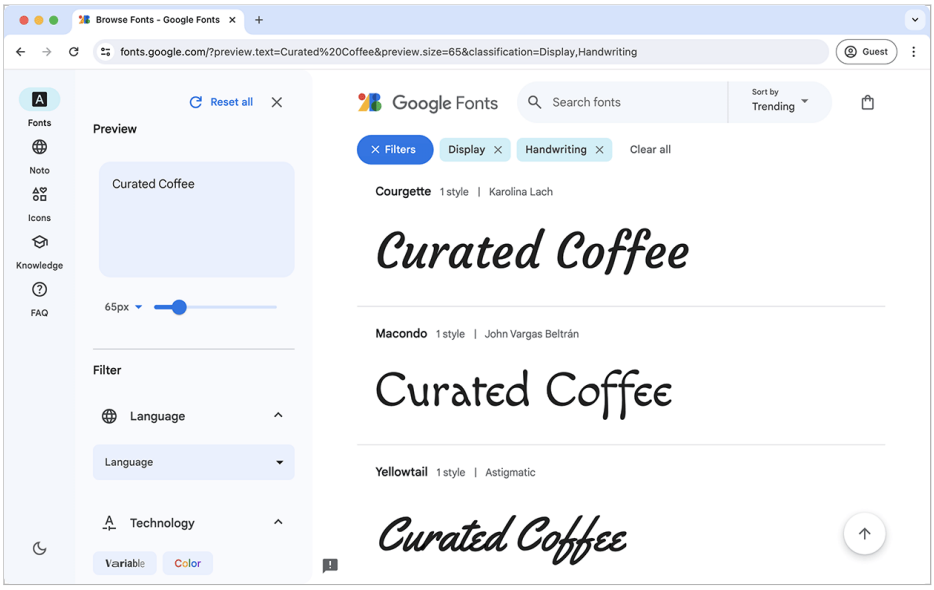
   NOTE: Percentage values are relative to the background positioning area. 0% 0% is left top. 100% 100% is right bottom. 50% 50% is center. We set the Y offset to 75% to move the image up, so we see more of the bottom part of the image.

8. Return to **main.css** in your code editor to lock this change into the final code.
9. Edit the background-position property value for the **.hero** rule as follows:

```
    background-position: center 75%;
    background-repeat: no-repeat;
    background-size: cover;
}
```

10. Save **main.css**.
11. Return to the browser, reload the page, and enjoy your nice looking hero image!

Exercise Preview



Exercise Overview

In this exercise, we'll continue with the Hipstirred site. We'll give the page a more polished, professional look by embedding custom fonts rather than sticking to the safe, bland list of fonts that ship with most operating systems. We'll use **Google Fonts**, a popular source of free web fonts. Google hosts many fonts and provides the code to make it all work.

1. We'll be using a new folder of provided files for this exercise. Close any files you may have open in your code editor to avoid confusion.
2. For this exercise we'll be working with the **Hipstirred Custom Fonts** folder located in **Desktop > Class Files > Web Dev Class**. You should open that folder in your code editor if it allows you to (like Visual Studio Code does).

Limiting the Text Content's Width

1. In your code editor, open **index.html** from the **Hipstirred Custom Fonts** folder.
2. Preview **index.html** in a web browser and notice:
   - We've added some text content into the main tag, and a copyright line into the page's footer.
   - The text touches the edge of the page and the long lines of text are hard to read. Before we experiment with fonts, let's limit the width of the main content and add some space around it.

3. Return to your code editor.
4. Open **main.css** from the **Hipstirred Custom Fonts** folder.
5. Below the **.hero** rule, add the following new rule:

```
main {
   max-width: 850px;
   padding: 25px;
}
```

6. Save **main.css**.
7. Return to the browser and reload **index.html**. The text is far more legible at this max-width but the main content should be centered in the page.
8. Return to **main.css** in your code editor.
9. Add the following new properties to the rule for main:

```
main {
   max-width: 850px;
   padding: 25px;
   margin: auto;
}
```

10. Save **main.css**.
11. Return to the browser and reload **index.html**.

    The layout is better, so now we can turn our attention to the fonts! Let's find a better font to use instead of the default.

    NOTE: Don't worry about the footer's copyright text being aligned to the far left. We'll finish the footer in a later exercise.

Experimenting with Google Fonts

1. In a new browser tab, go to **fonts.google.com**

   This service from Google provides free fonts that you can use for any commercial project (websites, apps, etc.).

2. We want to preview our specific **Curated Coffee** text. Click where it says **Type something** and type **Curated Coffee**
3. Take a moment to scroll through some of the available fonts.
4. We have three fonts specific fonts we want to try. In the search field at the top of the page, search for **Medula**.
5. Click on **Medula One**

6. Notice that this font has only one style: **Regular 400**.

   Font weights for the web are numbered. Weights such as 100, 200, 300, 400, 500, 600, 700 go from thin (lower numbers) to bolder (higher numbers).

7. Click on **Get font**.

   A shopping bag icon will appear at the top right. We have more fonts to add, so we're not going to get the code yet.

8. Let's find a second font:

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

9. Let's find a third font:

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

Adding Fonts to Our Page

1. At the top right of the page click the shopping bag icon.
2. You should see 3 fonts listed: **Abel**, **Medula One**, and **Rancho**.
3. Click **Get embed code**

   NOTE: These 3 fonts only have regular versions, but other fonts may have bold, italic, light, black, etc. When deciding on which styles to include, keep in mind that each style increases the file size and slightly slow down the loading of the page. Only load styles you will actually be using.

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

```
<link rel="preconnect" href="https://fonts.googleapis.com">
<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
<link href="https://fonts.googleapis.com/css2?
family=Abel&family=Medula+One&family=Rancho&display=swap" rel="stylesheet">
```

5. Keep this page open in your browser, you'll need to come back to it later.
6. Return to **index.html** in your code editor.

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

```
<title>Great coffee, delivered to your door - Hipstirred</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=Abel&family=Medula+One&family=Rancho&display=swap" rel="stylesheet">
<link rel="stylesheet" href="main.css">
```

   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.

8. Save the file.

Applying the Fonts to Content

Now that the fonts are accessible to use in this page, let's use them!

1. Return to the Google Fonts website in your browser.
2. Copy the code for the **Abel** font:

```
font-family: "Abel", sans-serif;
```

3. Return to **main.css** in your code editor.
4. Edit the rule for **body** as follows:

```
body {
   font-family: "Abel", sans-serif;
   margin: 0;
}
```

5. Save **main.css**.
6. Return to the browser and reload **index.html**. All the text should now be the new font. It's nice, but a bit too small.
7. Return to **main.css** in your code editor.
8. Edit the rule for **body** as follows:

```
body {
   font-family: "Abel", sans-serif;
   font-size: 19px;
   margin: 0;
}
```

9. Save **main.css**.
10. Return to the browser and reload **index.html**. That's better. Now let's choose another font for the main heading.
11. Return to Google Fonts in your browser.
12. Copy the code for the **Medula One** font:

```
font-family: "Medula One", system-ui;
```

13. Return to **main.css** in your code editor.
14. Edit the rule for **h1** as follows:

```
h1 {
  font-family: "Medula One", system-ui;
  font-size: 95px;
}
```

15. System UI fonts are native to each operating system, so they will vary from one platform to another (Mac, Windows, iOS, Android). We don't want that, so change **system-ui** to **sans-serif** as shown here:

```
h1 {
  font-family: "Medula One", sans-serif;
  font-size: 95px;
}
```

16. Save the file and reload **index.html** in the browser to see the new font. The heading should now be in a new font, but there's an issue we need to fix.

    Browsers make headings bold by default. This font does not have a bold version, so the browser is artificially making it bold. Let's change it to the normal weight (400) so we can truly see how this font looks.

1. Return to **main.css** in your code editor.
2. Add the following new property declaration to the **h1** rule:

```
h1 {
   font-family: "Medula One", system-ui;
   font-size: 95px;
   font-weight: 400;
}
```

   NOTE: We could have also used **font-weight: normal;** because 400 equates to normal. We used 400 in this case because we saw the 400 weight listed on Google Fonts and wanted to make the connect to that.

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

   Notice that the heading font is thinner. This is how this font was designed. This font is interesting, but too similar to the other font we're using. Let's try a different font.

4. Return to Google Fonts in your browser.
5. There are no good cursive fallback fonts common to all platforms (Mac, PC, iOS, etc.), so just copy the **Rancho** font name.
6. Return to **main.css** in your code editor.
7. Edit the **h1** rule as follows:

```
h1 {
   font-family: "Rancho", sans-serif;
   font-size: 95px;
   font-weight: 400;
}
```

8. Save **main.css**.
9. Return to the browser and reload **index.html**. Nice, we like this font!

Removing an Unused Font

We're no longer using the **Medula One** font, but the page is still downloading it. That makes the page load a bit slower and wastes mobile users' data! Let's fix that.

1. Return to **index.html** in your code editor.
2. In the link tag for Google fonts, remove **&family=Medula+One** so you end up with:

```
<link href="https://fonts.googleapis.com/css2?
family=Abel&family=Rancho&display=swap" rel="stylesheet">
```

3. Save the file and reload the page in the browser to make sure you didn't break the fonts from loading.

Styling the Hero Text Below the Heading

1. Return to **main.css** in your code editor.
2. Let's style the paragraph in the hero section (under the h1 **Curated Coffee**). Add the following new rule below the **h1** rule:

```
.hero p {
   text-transform: uppercase;
}
```

3. Save the file and reload the page in the browser to preview the change. This looks good, except there's too much space between the heading and paragraph.

Improving Margins & Line-Height

1. Return to **main.css** in your code editor.
2. Add the following new property declaration to the **.hero p** rule:

```
.hero p {
   text-transform: uppercase;
   margin: 0;
}
```

3. Add the following new property declarations to the **h1** rule:

```
h1 {
   font-family: "Rancho", sans-serif;
   font-size: 95px;
   font-weight: 400;
   margin-top: 0;
   margin-bottom: 10px;
}
```

4. Save the file and reload **index.html** in the browser and:
   - Make your browser window wide and notice the headings look good.
   - Make the browser window narrow enough so **Curated** and **Coffee** each wrap onto a separate line and you'll be able to see there's too much line space:

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

6. Add the following new property declaration to the **h1** rule:

```
h1 {
   font-family: "Rancho", sans-serif;
   font-size: 95px;
   line-height: 85px;
   font-weight: 400;
   margin-top: 0;
   margin-bottom: 10px;
}
```

7. Save the file and reload **index.html** in the browser to preview the change. Resize the browser window to fully test the new line-height. Looks good.

8. Now let's style the h2 tags (the headings in the text below the hero image) to create more obvious topic breaks. Return to **main.css** in your code editor.

9. Add the following new rule below the **h1** rule:

```
h2 {
   font-size: 28px;
   border-bottom: 1px solid #d17e32;
   padding-bottom: 10px;
   margin-top: 40px;
   margin-bottom: 10px;
}
```

10. Save the file and reload **index.html** in the browser to preview the refined headings.

11. The paragraphs could also use refinement. Return to **main.css** in your code editor.

12. Let's increase the paragraph margin and line-height. Add the following new rule for **p** below the **h2** rule:

```
p {
   line-height: 32px;
   margin-top: 0;
   margin-bottom: 20px;
}
```

13. Save the file.

14. Return to the browser to reload **index.html**. The type is looking good now that we have custom fonts and better spacing! We just have one last little tweak.

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

16. Let's soften the contrast a little by making all the text a dark gray:

```
body {
   font-family: "Abel", sans-serif;
   font-size: 19px;
   color: #555;
   margin: 0;
}
```

17. Save the file.

18. Return to the browser to reload **index.html**.

Testing Custom Web Fonts (Such as Google Fonts)

Test your site on other computers/devices to make sure your custom web fonts load properly. Why? When designing the site you often install the font on your Mac/PC. If the embed code for the webpage doesn't work, you wouldn't know because the web browser can use the font installed in your system. You'd only see the problem when testing on another computer, phone, or tablet that does not have the font installed. It's also good to test in various browsers/platforms as fonts may render a bit differently.

Using Google Fonts in Design Apps

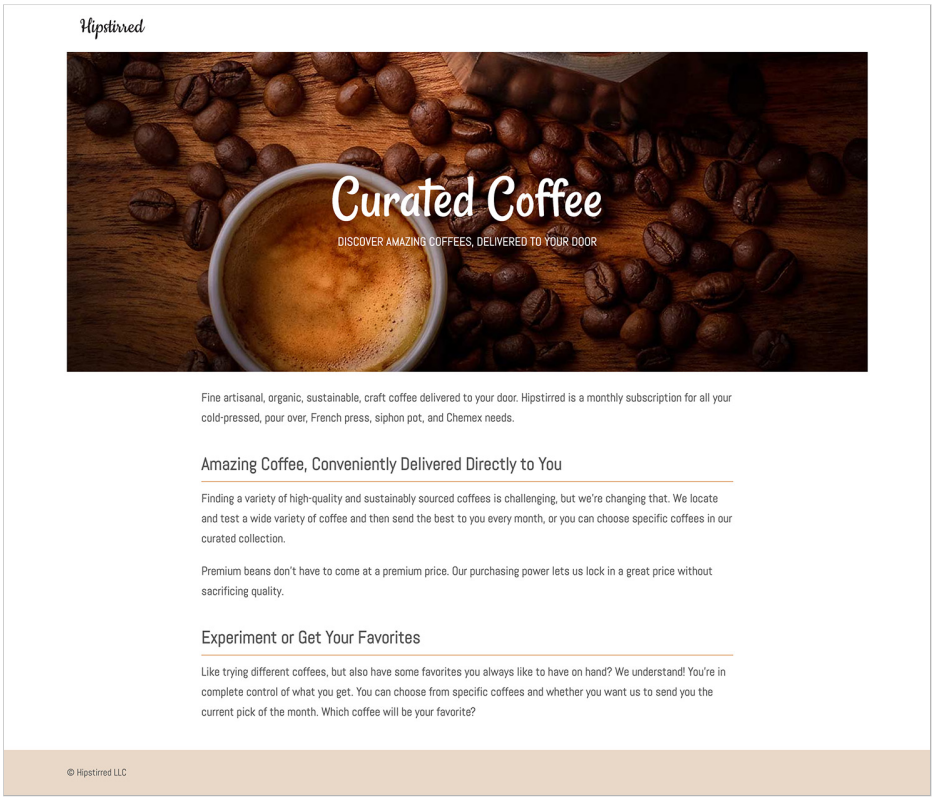
If you're designing a website in Figma, Figma loads Google Fonts automatically, making it easy to use them in your designs.

If you need to use Google web fonts while designing in desktop apps like Photoshop or Sketch, you'll need to download Google fonts and install them on your Mac or PC. One way you could do this more easily (instead of manually getting them from the Google Font website) is using **FontBase**. FontBase is a free app that allows you to quickly install Google fonts (and more) onto your Mac or PC. Find out more at **https://fontba.se**

Google Fonts May Be Blocked

Corporate, campus, or even government firewalls may block Google Fonts. For instance, people in China will have to enjoy Hipstirred in Helvetica or Arial because Google Fonts are blocked there. While most users will see the page with your Google Fonts, it's possible not everyone will.

Exercise Preview



Exercise Overview

In this exercise, you will refine the spacing and width of the layout for the Hipstirred site. Along the way, you'll investigate wrapping content in container divs and firm up your understanding of how an ID selector differs from a class selector.

1. We'll be using a new folder of provided files for this exercise. Close any files you may have open in your code editor to avoid confusion.
2. For this exercise we'll be working with the **Hipstirred Box Model** folder located in **Desktop > Class Files > Web Dev Class**. You should open that folder in your code editor if it allows you to (like Visual Studio Code does).
3. Open **index.html** from the **Hipstirred Box Model** folder.
4. Preview **index.html** in Chrome to see what we have so far. (We will be using Chrome's DevTools.)

   Let's start by adding the logo to the header, and then styling the footer.

   NOTE: We recommend leaving **index.html** open in Chrome as you work, so you can reload the page to see the changes you make in the code.

Adding the Logo to the Header

1. Return to **index.html** in your code editor.
2. In the **header** add the following image:

```
<header>
   <img src="images/logo.svg" height="35" width="105" alt="Hipstirred">
</header>
```

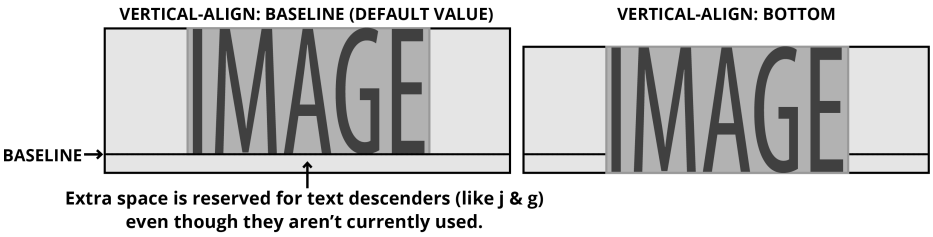
   NOTE: We're coding a width and height for this logo because it's always going to be a fixed size in the layout. If the size was going to be flexible, we'd omit the height and width.

3. Save the file.
4. Return to the browser to reload the page. The logo is way too tight against the top-left edge, but before we add space around it, notice that there's a little extra space below the logo. Why is that?

Removing the Extra Space Below an Image

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. Open **main.css** from the **Hipstirred Box Model** folder.
2. Add the following new rule below the **body** rule:

```
img {
   vertical-align: bottom;
}
```

3. Save **main.css**.

4. Switch back to Chrome and reload **index.html** to see that the extra space below the logo is now gone.
5. Now let's add space around the logo. Switch to **main.css** in your code editor.
6. Add the following new rule below the **p** rule:

```
header {
   padding: 20px;
}
```

7. Save **main.css**.
8. Return to the browser and reload **index.html**. The header looks better with space around the logo.

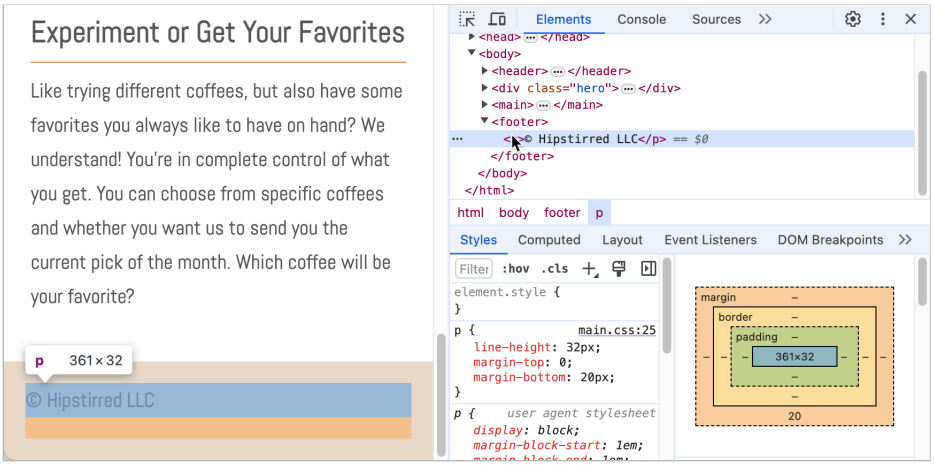
Styling the Footer

1. Switch to **main.css** in your code editor.
2. Let's add a background color to the footer to help separate it from the rest of the page. At the bottom of the file, just below the **main** rule, add the following new rule:

```
footer {
   background-color: #e9d8c8;
   padding: 20px;
}
```

3. Save **main.css**.
4. Return to Chrome and reload **index.html**. We like how the footer is looking with the background color, but the copyright paragraph doesn't look centered vertically. What's going on? Let's inspect.
5. **Ctrl–click** (Mac) or **Right–click** (Windows) on the footer and choose **Inspect**.

6. In the Elements panel, select the **<p>** tag in the footer. You'll see it highlighted in the browser and you should also see the **p** rule in the **Styles** tab of the DevTools window:



Bottom margin is good for typical paragraphs (like those in the main section of the page), but not when we only have this one paragraph in the footer. Luckily descendant selectors allow us to target an element when it's inside of another element. We want to style the **p** tags only inside the **footer**.

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

8. Let's remove the space below the copyright, and make it smaller while we're at it. At the bottom of the file, just below the **footer** rule, add the following new rule:

```
footer p {
   margin: 0;
   font-size: 15px;
}
```

9. Save **main.css**.

10. Return to the browser and reload **index.html**. Now the footer and copyright text look better, and copyright text is vertically centered in the footer's background color.

11. Keep the browser window open, but close the DevTools.

Setting Limits with an Outer-Wrapper & Max-Width

1. Resize your browser window as wide as it will go. This hero image looks good at 1280 pixels or less. If you make the window wider than 1280 pixels, the hero image starts to become pixelated. We could use a bigger image, but another option is to limit the width of the hero area.

   To control the overall width of the content on the page, we can wrap a `<div>` around all the content and then set limits to its width in the CSS.

2. Return to your code editor and switch to **index.html**.
3. Wrap a **div** tag around all the content of the document inside the **body** tag.

   TIP: If you're using Visual Studio Code (and set up the wrap keystroke using the instructions in the **Before You Begin** section near the start of this book), you can wrap a tag around a selection by pressing **Option–W** (Mac) or **Alt–W** (Windows). Then type in the wrapper (**div** in this case) and hit **Return** (Mac) or **Enter** (Windows).
4. When you're finished, check that you have an opening ** ** (highlighted below in bold) just below the opening **body** tag.

```
<body>
   <div>
      <header>
```

5. Check that you have a closing ** ** (highlighted below in bold) just above the closing **body** tag.

```
      </footer>
   </div>
</body>
</html>
```

6. Give the new an ID of **wrapper**:

```
<body>
   <div id="wrapper">
      <header>
         <img src="images/logo.svg" height="35" width="105" alt="Hipstirred">
```

7. Save **index.html**.
8. Switch to **main.css**.
9. Below the **p** rule, add the following new rule:

```
#wrapper {
   max-width: 1100px;
}
```

10. Save **main.css**.
11. Return to the browser and reload **index.html**. Resize your browser to make it narrow and wide. You'll see the content stop scaling up when you get to 1101 pixels or wider. Our max-width works, but the content isn't centered.
12. Return to **main.css** in your code editor.

13. Add the following new properties to the #wrapper rule:

```
#wrapper {
   max-width: 1100px;
   margin: auto;
}
```

14. Save **main.css**.

15. Return to the browser and reload **index.html**. Now the hero does not get too wide, and when when the window is 1101 pixels or wider, the hero area is centered within the page.

    The top part of the page looks, but scroll down to the footer. Although we'd like the copyright content to have a max-width, the background color would look nicer stretching edge-to-edge.

Setting an Inner-Wrapper

1. Return to your code editor and switch to **index.html**.

2. What we need to do is end the current wrapper's `</div>` just after the `</main>` section and leave the footer out of the wrapper. Move the **`</div>`** as follows:

```
      </main>
   </div>
   <footer>
      <p>© Hipstirred LLC</p>
   </footer>
</body>
```

3. Save the file, return to the browser and reload the page to see that the footer is no longer constrained.

   We'd still like to have the copyright align on the left with the Hipstirred logo. To accomplish this, we need to place a wrapper around the content **inside** of the footer.

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

5. As shown in bold, wrap a `<div>` tag around the **`<p>`** inside the `<footer>` and give it an ID of **wrapper**:

```
<footer>
   <div id="wrapper">
      <p>© Hipstirred LLC</p>
   </div>
</footer>
```

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

   This worked, but only because the browser is giving us a pass right now. We used the same ID twice, which goes against spec. We're seeing a "silent failure", which means that we have written invalid code, but the browser will try to guess what our intent was and handle it accordingly.

ID Selectors vs. Class Selectors

IDs signify uniqueness. They are meant to be used just once, per element, per page. Although the browser fails "silently" right now, this issue can come back to cause us problems down the road. Since the class selector was created specifically to style multiple elements the same way, let's use a class for wrapper instead of an ID.

1. Return to **index.html** in your code editor.
2. Scroll up to the first **wrapper** and edit the attribute as follows:

```
<body>
   <div class="wrapper">
      <header>
```

3. Down in the footer, change **id** to **class** as well:

```
<footer>
   <div class="wrapper">
      <p>© Hipstirred LLC</p>
   </div>
</footer>
```

4. Save the file.
5. Switch to **main.css**.
6. Change **#wrapper** to **.wrapper** (which changes it from an ID selector to a class selector), so you end up with the following:

```
.wrapper {
   max-width: 1100px;
   margin: auto;
}
```

7. Save the file, return to the browser, and reload the page. If all goes well, you won't see any change but you can bask in the glow of your valid HTML and CSS.

Real-World Widths

We set the max-width value to 1100 pixels to make it easy to see max-width in action, even if you are working with a smaller resolution. For real-world design, though, we recommend setting a max-width to a more standard resolution, like 1280 pixels. This gives these users the edge-to-edge look we desire but also caps off the content when it gets too wide for users who have a higher resolution.

1. Return to **main.css** in your code editor. We'll set the max-width to be a more generous size.
2. Edit the value for the **max-width** property for **.wrapper** as follows:

```
.wrapper {
   max-width: 1280px;
   margin: auto;
}
```

3. Save **main.css**.
4. Return to the browser and reload **index.html**. If your screen is 1280px wide or less, the hero image should fill the page. If your screen is wider than 1280px, test out the new max-width value. Nice.

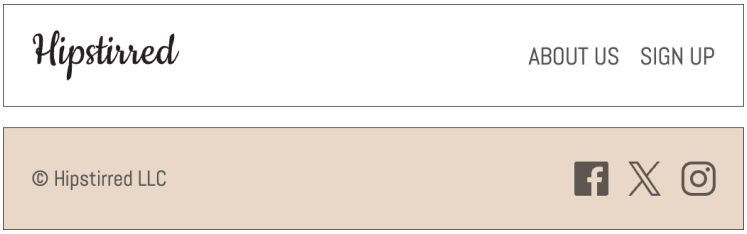
   NOTE: We're not saying you always have to limit the width of your hero images, but in case you do, we wanted to show you how.

Screen Resolution Statistics

gs.statcounter.com is a good resource for web analytics. It's free and gives you a global picture of what's happening on the web.

The best way to find out who's looking at your site, what browser they're using, what their screen resolution is, where they live, and almost anything else you'd like to know, is to use Google Analytics, the industry standard for data. Google Analytics is free, though they have tiered pay plans for "enterprise-scale" data. Find out more at google.com/analytics

Exercise Preview



Exercise Overview

In this exercise, you'll review semantically correct markup for navigation and how to style it. You'll also learn how to adjust the layout, using flexbox to pull elements to the far left and far right of the layout.

1. We'll be using a new folder of provided files for this exercise. Close any files you may have open in your code editor to avoid confusion.
2. For this exercise we'll be working with the **Hipstirred Navbar Layout** folder located in **Desktop > Class Files > Web Dev Class**. You should open that folder in your code editor if it allows you to (like Visual Studio Code does).
3. Open **index.html** from the **Hipstirred Navbar Layout** folder.
4. Preview **index.html** in a web browser.

   NOTE: We recommend leaving **index.html** open in the browser as you work, so you can reload the page to see the changes you make in the code.

Adding Links to Other Pages

1. Return to **index.html** in your code editor.
2. As shown below, add a **nav** tag below the **img** inside the **header**:

```
<header>
   <img src="images/logo.svg" height="35" width="105" alt="Hipstirred">
   <nav>

   </nav>
</header>
```

3. Add an unordered list with two links, as follows:

```
<nav>
   <ul>
      <li><a href="about.html">About Us</a></li>
      <li><a href="signup.html">Sign Up</a></li>
   </ul>
</nav>
```

   NOTE: The pages we're linking to do not exist yet, but we know the names we want to use for them. We may as well code the links now and create those pages later.

4. Save the file, return to the browser, and reload the page. While it's semantically correct to mark up a list of links as a list, the list styling is not what we want. Let's create a better look for the navigation.

5. Return to your code editor and open up **main.css**.

6. To remove the default bullets, margin, and padding from the unordered list, we should write a descendant selector. This way we know we'll specifically target unordered lists in the navigation and nowhere else on the site. Add the following new rule below the rule for **header**, as follows:

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

7. Save the file.

8. Return to the browser and reload **index.html**. That's better without the bullets and extra space, but let's put the list items next to each other in the same line.

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

10. Add the following new rule below the **nav ul** rule:

```
nav li {
   display: inline;
   margin-left: 10px;
}
```

11. Save the file.

12. Return to the browser and reload **index.html**. Now the two nav links should be next to each other. Let's finish styling the links.

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

14. We should write another descendant selector that specifically targets links in the navigation. This will ensure that link styles elsewhere won't be affected. Add the following new rule below the **nav li** rule:

```
nav a {
   color: #555;
   text-decoration: none;
   font-size: 17px;
   text-transform: uppercase;
}
```

15. Save the file.

16. Return to the browser and reload **index.html**. That's looking better, but let's put the links to the far right of the logo (on the same line).

    To do this we'll use a CSS layout feature called flexbox.

Aligning the Nav Links to the Right Side of the Page

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

2. Flexbox lets you align/distribute child elements nested inside of a parent element.

   Notice that the logo **img** and **nav** are contained in the **header**. Flexbox must be applied to the parent element (in this case the **header**).

3. Switch to **main.css** in your code editor and in the **header** rule add this new property:

```
header {
   padding: 20px;
   display: flex;
}
```

4. Save the file.

5. Return to the browser and reload **index.html**. Now the navigation links should in the same line as the logo, but we want them vertically aligned center with the logo.

6. Return to **main.css** in your code editor and in the **header** rule add this new property:

```
header {
   padding: 20px;
   display: flex;
   align-items: center;
}
```

   NOTE: For a complete guide to flexbox properties, and whether they are applied to the child or parent elements, refer to **css-tricks.com/a-guide-to-flexbox**

7. Return to the browser and reload **index.html**.
   - Now the navigation links should in the same line as the logo and vertically centered.
   - We also want to move the links to the right (with the logo remaining on the left). We want to put the space (margin) to be between the logo and nav.
8. Return to **main.css** in your code editor and above the **nav ul** rule add this new rule:

```
nav {
   margin-left: auto;
}
```

   NOTE: We could also put margin-right: auto; on the logo image and it would work the same way.
9. Save the file.
10. Return to the browser and reload **index.html**. Now the nav links should be aligned to the right!

Aligning the Social Media Links in the Footer

1. Still looking at the page in your browser, scroll down to the footer. Here we have social media icons, but they have not been styled:
   - There are tiny underlines between the social icons (because links normally have underlines).
   - The icons are too close together.
   - The black is a bit harsh compared to the gray of the text.
   - We also want to align the social icons to the right, as we did with the nav links.
2. Return to **main.css** in your code editor.
3. Below all the other rules, add the following new rule:

```
.social a {
   text-decoration: none;
   margin-left: 10px;
   opacity: .6;
}
```

4. Save the file.
5. Return to the browser to reload **index.html**. The icons look better, but we see one last issue. There's some extra space at the bottom of the footer that we want to remove.
6. Return to **index.html** in your code editor.

7. Find the **footer** (near the bottom) and notice:
   - All the footer content is contained in a **wrapper** div.
   - A **social** div contains the linked social media images.
8. Switch to **main.css** and below the **footer** rule, and this new rule:

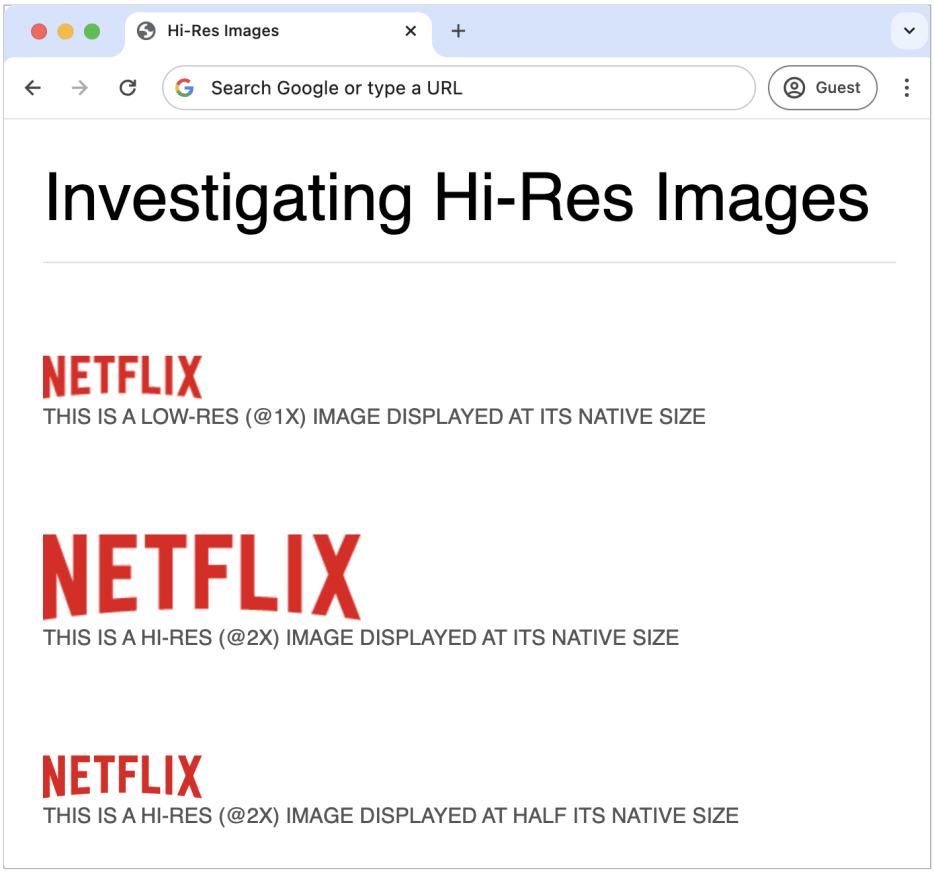
```
footer .wrapper {
   display: flex;
   align-items: center;
}
```

9. Save the file.
10. Return to the browser and reload **index.html**. The copyright and social icons should all be on the same line now.
11. Switch to **main.css** and above the **.social a** rule, and this new rule:

```
.social {
   margin-left: auto;
}
```

12. Save the file.
13. Return to the browser to reload **index.html**. The footer is now looking good!

Exercise Preview



Exercise Overview

Some screens have a higher pixel density (smaller, more tightly packed pixels) than others. These screens—which Apple calls **Retina**, but are also known as **HiDPI** (high dots per inch)—pack at least twice the number of pixels into a given screen size (compared to low-res screens). To provide crisp images for HiDPI/Retina devices, we have to use higher-resolution images.

1. We'll be using a new folder of provided files for this exercise. Close any files you may have open in your code editor to avoid confusion.
2. For this exercise we'll be working with the **Hipstirred Hi-Res Images** folder located in **Desktop > Class Files > Web Dev Class**. You should open that folder in your code editor if it allows you to (like Visual Studio Code does).
3. Open **hi-res-demo.html** from the **Hipstirred Hi-Res Images** folder.
4. Preview **hi-res-demo.html** in a browser. As the headings explain, we'll be placing both a low-res (@1x) and a high-res (@2x) version of an image in the browser to do some comparison testing.

Sizing Things Up

1. Return to **hi-res-demo.html** in your code editor.
2. Add the following image just above the first **h2**, as follows:

```
<img src="images/company-logo.png" width="120">
<h2>This is a low-res (@1x) image displayed at its native size</h2>
```

   NOTE: Whenever possible, we'd prefer to use an SVG version. We can't always get vector versions of all graphics though, so it's still important to know how to properly display pixel-based graphics so they look good on high-res screens.

3. Save the file and preview it in a browser. You are viewing this logo at its native width of 120 pixels.
   - If you're on a standard/low-res screen, it looks fine.
   - If you're on a Retina/HiDPI screen, you may notice that it's a bit fuzzy and/or pixelated (if your eyesight is good enough).

   To create sharper images for users with hi-res screens, web designers create images at twice the size at which they are intended to be displayed. Because these images have 2 pixels for every 1 pixel in a low-res image, they are called **2x** images. A hi-res **2x** image occupies the same amount of physical screen space as a low-res **1x** image, but because there are more, smaller pixels packed into the same space it contains more details and appears sharper. As an industry standard practice, hi-res images are saved with **@2x** at the end of their file name so developers know they are hi-res. Let's take a look at a @2x version of the logo image.

4. Return to your code editor.
5. Add the following @2x image just above the second **h2**, as follows:

```
<img src="images/company-logo@2x.png" width="240">
<h2>This is a hi-res (@2x) image displayed at its native size</h2>
```

6. Save the file and preview it in a browser. It's twice as big, but still fuzzy and/or pixelated. We actually want the 2x image to take up half as much space, so it ends up being the same size as the 1x above.

Code Pixels vs. Hardware Pixels

Not all pixels are the same size. While measurements like inches and millimeters are always the same physical size, a pixel can be any physical size (depending on the pixel density of the screen).

When HTML and CSS were invented, 1 pixel in code equaled 1 pixel of hardware. With hi-res screens, 1 pixel in code may equal 2 (or more) pixels of hardware. Luckily we don't have to worry about how many hardware pixels a user's screen has. The web browser automatically translates coded pixels into hardware pixels. For example, the original iPhone was 320px wide in both code and hardware. The first Retina iPhone was still 320px wide in code (because the screen was the same physical size), but 640px of hardware.

1. Let's see this in action. Return to your code editor.
2. As shown below, add the @2x image again just above the third **h2**, but this time with the same size as the 1x:

```
<img src="images/company-logo@2x.png" width="120">
<h2>This is a hi-res (@2x) image displayed at half its native size</h2>
```

We set the pixel size in the code to half the image file's native size. This will end up displaying at the size we want, but at high resolution (more pixels are packed into the space). The logo will appear the same size on all screens, but it will look sharper on HiDPI screens.

3. Save the file and preview it in a browser.
   - If you're on a hi-res screen: The bottom hi-res @2x logo will look sharper than the other logos. Success!
   - If you're on a low-res screen: The bottom hi-res @2x logo will not appear sharper, because of the limitations of your screen. (Think of it like trying to view a color photo on a black and white screen. The black and white screen is simply not capable of displaying color.)

NOTE: You could also use CSS to set the size of a Retina or HiDPI image with something like this:

```
.logo {
   width: 120px;
}
```

Replacing the Low-Res Photo with a @2x Version in Hipstirred

1. Return to your code editor.
2. Open **index.html** from the **Hipstirred Hi-Res Images** folder.
3. Preview **index.html** in a browser.

Notice the background image for the hero section doesn't look bad, but its not a high resolution as it can be. Let's replace it with a provided @2x photo.

4. Open **main.css** from the **Hipstirred Hi-Res Images** folder.
5. Find the rule for **.hero** (near the bottom) and edit the **background-image** value as follows:

   ```
   background-image: url(images/hero@2x.jpg);
   ```

6. Save the file.
7. Return to **index.html** in the browser and reload the page.

   Nice! But how do you know this is actually a retina image, particularly if you're working on a standard/low-res screen? It works too seamlessly. Let's investigate.

8. Return to **main.css** in your code editor.
9. In the rule for **.hero**, comment out the **background-size** property as follows:

   ```
   /*background-size: cover;*/
   ```

   NOTE: A comment in HTML or CSS is ignored by the web browser. We often use it to leave a note or providing information about the code to other developers (or ourselves). It can also be used to temporarily disable code for testing purposes.

   Comments in CSS are written like this:

   ```
   /* this is a comment */
   ```

   Comments in HTML are written like this:

   ```
   <!-- this is a comment -->
   ```

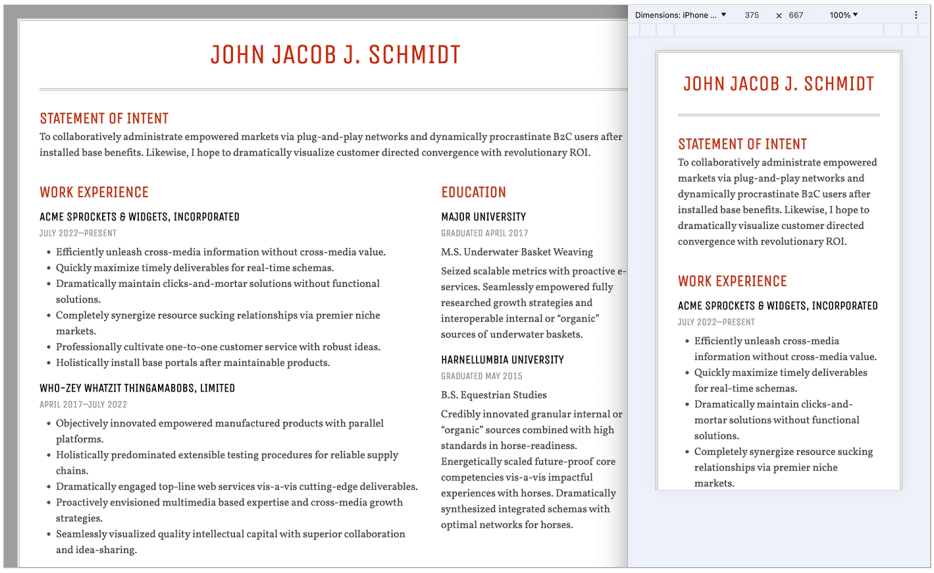
   In most code editors (like Visual Studio Code), you can select code you wish to comment and hit **Cmd–/** (Mac) or **Ctrl–/** (Windows) to toggle comments on or off. The editor will know if you're in CSS or HTML and write the appropriate comment.

10. Save the file.
11. Return to **index.html** in the browser and reload the page. Woah, that's a big hero image… it must be the 2x size.
12. Return to **main.css** in your code editor.
13. In the rule for **.hero**, uncomment the **background-size** property to make the browser render it properly again:

    ```
    background-size: cover;
    ```

14. Save the file.
15. Return to **index.html** in the browser and reload the page. This page is Retina optimized!

Exercise Preview



Exercise Overview

In this exercise, you'll use CSS to define a grid of columns and rows to layout the page. You'll also learn how to change the layout based on screen size, making the design switch from a 1-column layout on small screens, to a 2-column layout on larger screens.

Getting Started

1. In your code editor, close any files you may have open.
2. For this exercise we'll be working with the **Resume** folder located in **Desktop > Class Files > Web Dev Class**. You should open that folder in your code editor if it allows you to (like Visual Studio Code does).
3. Open **resume.html** from the **Resume** folder.
4. Preview **resume.html** in Chrome. (We're using Chrome because we'll be using its DevTools later.)
5. Notice that the page is mostly styled. We're going to focus on putting the **Work Experience** section on the left and the **Education** section on the right so we have a 2-column layout.

   NOTE: We recommend leaving **resume.html** open in the browser as you work, so you can reload the page to see the changes you make in the code.
6. Let's take a quick look at the HTML markup. Switch back to your code editor.

7. Take a look at the code to familiarize yourself with the basic page structure, focusing on the following elements:
   - On line 11 notice there is **wrapper** div. This is the parent container for all the content on the page.
   - Inside **main** (starting on line 15) are 3 **primary sections**, each with a unique **id**.
   - We want the **statement** section (on line 16) to span 2 columns, above the **experience** section (on line 20) as one column on the left, and the **education** section (on line 55) as another column on the right.
8. Let's start styling. Open **main.css** from the **Resume** folder.

Creating a 2-Column Layout with CSS Grid

To make columns we'll define a grid which indicates how many columns we want and how wide we want them to be. We define the grid on the parent element which contains the elements we want to be become columns and rows in the grid.

1. At the bottom below all the other rules, add the following new rule:

```
main {
   display: grid;
   grid-template-columns: 2fr 1fr;
}
```

   NOTE: The **fr** unit is a **fr**actional unit. According to the spec **tinyurl.com/fr-unit** an **fr** unit "represents a fraction of the leftover space in the grid container". So another way to think of **fr** is that it stands for "free space". First the browser calculates the size of grid items with specific widths (such as pixels), then takes the leftover free space and distributes it between the grid tracks with fr units.

2. Save **main.css**.
3. Switch to the browser and reload **resume.html**.

   While the **Statement of Intent** currently looks weird where it is (we'll fix that next), we do have 2 columns, with the left column being twice the width as the right column. There is also a second row down below, which contains the **Education** section (in the left column of that second row).

   Let's make the **Statement of Intent** span across the 2 columns.

4. Return to **main.css** in your code editor.
5. At the bottom below all the other rules, add the following new rule:

```
#statement {
   grid-column: span 2;
}
```

6. Save **main.css** and reload **resume.html** in the browser.

   Much better! It could use some space between the columns and rows though.

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

8. In the **main** rule, add the **gap** property (highlighted below in bold):

```
main {
   display: grid;
   grid-template-columns: 2fr 1fr;
   gap: 30px;
}
```

9. Save **main.css** and reload **resume.html** in the browser to see that the columns now have space between them.

   This looks great on a wide screen, but small screens don't have enough space for multiple columns.

Finding an Appropriate Breakpoint

Let's see how to change the layout across screen sizes.

1. To simulate a small screen, make the browser window as narrow as possible and notice two problems:

   - The 2-column layout looks fine when the window is wider, but they're very hard to read when the window is narrow (like a phone). We should switch to a single-column layout for small screens.
   - When the window is narrow, it would be better if we didn't have the gray area around the page, so more of the screen space would be used for content.

2. Make the window wide again.

   We need to find a good **breakpoint**, a point at which the layout needs to change. Let's open the DevTools so we can see the browser's width and height—a feature that's unique to Chrome's DevTools.

3. Open Chrome's DevTools using the following keystroke:

   - Mac: Hit **Cmd–Opt–I** (that's a letter i, not the number 1) or **F12** (you may have to hold the **fn** key when hitting **F12** on some keyboards).
   - Windows: Hit **Ctrl–Shift–I** (that's a letter i, not the number 1) or **F12** (you may have to hold the **fn** key when hitting **F12** on some keyboards).

4. Slowly resize the browser window while looking at the **top-right corner** of the preview area to see the pixel width and height of the visible area.

5. Find a width where you think the 2-column layout starts looking cramped and we should switch to 1-column. It's a matter of opinion, but we think that this point is around **700 pixels wide** so we'll use this as a breakpoint.
6. For best results the next time we preview the page:
   - Make sure the browser window is **narrower** than our breakpoint of **700 pixels**.
   - Leave both the page and the DevTools open in Chrome so we can come back to them later.

Using a Media Query to Change the Layout at a Specific Screen Size

To change the layout at our 700 pixel breakpoint, we will use a CSS **media query**. CSS media queries let us apply have CSS that only works if certain conditions are true, such as a specific screen size, screen resolution, type of device, etc. If these conditions are not met, the CSS rules are ignored.

1. Switch back to **main.css** in your code editor.
2. Let's start by removing the gray space around the page on small screens, so we can add it back on large screens. This space was created using margin on the **body** tag. At the very top of the file, change **margin** to **0** as shown below:

```
body {
    CODE OMITTED TO SAVE SPACE
    background: #9e9ea0;
    margin: 0;
}
```

3. Save **main.css**.
4. Switch to Chrome and reload **resume.html** to see the gray around the page should be gone.
5. Switch back to **main.css** in your code editor.

6. At the bottom of the file below all the other rules, write the media query shown below in bold:

```
#statement {
   grid-column: span 2;
}

@media (min-width: 700px) {

}
```

Let's break down that code:

- The code in parentheses is the query (the condition to test).
- In the curly braces we'll add CSS rules that we want to apply if the query is true.
- For querying browser/screen width we can use **min-width** and **max-width**. We used **min-width**, so the CSS we'll add inside the media query will only apply to screens that are 700px or wider.

7. Now that we have a way to target large screens, we can add the body margin back in (to create the gray space around the page). Add the following bold code:

```
@media (min-width: 700px) {
   body {
      margin: 20px;
   }
}
```

NOTE: This new rule in the media query will override the **body** rule at the top of the file, but only when the browser is 700px or wider.

> **Media Queries & CSS Style Order**
>
> If two rules have the same specificity, the latter will win. This is why media queries are put below the general styles. The rules in a media query will only apply if the condition is met, and will then override the general styles.

8. Save **main.css**.
9. Switch to Chrome and reload **resume.html**. Because you left the window narrower than 700px, the gray space around the page should NOT be visible.
10. Slowly resize the browser window to be wider. BOOM! Once we hit the breakpoint of 700 pixels, the gray space around the page should all of a sudden appear!

11. Resize the browser window to be narrower than 700px and notice the gray space around the page disappears (so there's more room for the content). As you can see, the rule inside our media query only applies when the condition is met. Otherwise the browser ignores the rule.

> **Mobile First**
>
> CSS at the top (not in a media query) applies to all screen sizes (mobile and larger). The min-width media query only applies to larger screens. The approach of starting with small screens and then larger screens is called "mobile first".

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

13. One smaller screens (like phones) we want a single column layout. Then on large screens (like tablets and desktop) we want a 2 column layout. In the **main** rule change the **grid-template-columns** property to **1fr** (highlighted below in bold):

```
main {
   display: grid;
   grid-template-columns: 1fr;
   gap: 30px;
}
```

    NOTE: This makes it one column that takes up all the free space.

14. Save **main.css**.

15. Switch back to Chrome and reload **resume.html**.

    The span on the **Statement of Intent** is still forcing 2 columns to be made, so we'll need to move that code into the media query for large screens.

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

17. Select and cut the **#statement** rule.

18. At the end of the media query, paste it so you get the following:

```
@media (min-width: 700px) {
   body {
      margin: 20px;
   }
   #statement {
      grid-column: span 2;
   }
}
```

19. Save **main.css**.

20. Switch back to Chrome and reload **resume.html**.

    Now the layout only has one column. That's good, this applies to all screen sizes, not only phones. Let's make the larger screens have 2 columns again.

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

22. In the media query, add a new rule for **main** as shown below in bold:

```
@media (min-width: 700px) {
   body {
      margin: 20px;
   }
   main {
      grid-template-columns: 2fr 1fr;
   }
   #statement {
      grid-column: span 2;
   }
}
```

23. Save **main.css**.

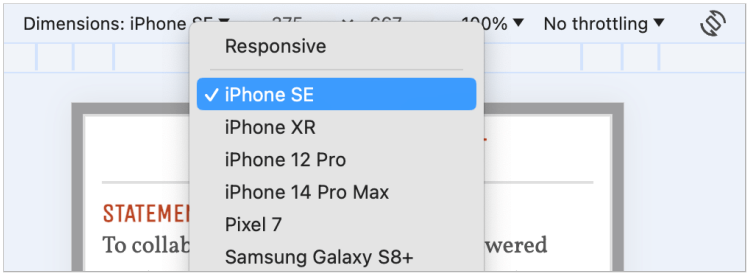
24. Switch back to Chrome and reload **resume.html**.

25. Resize the browser window in and out to see that all the elements are displayed in a single column until it reaches 700 pixels or wider, when the rules that comprise our 2-column layout kick in.

Disabling Mobile Browser Text Size Adjustment

1. On the upper left of the DevTools panel, click the **Toggle device toolbar** button to open the mobile simulator.

2. Above the webpage preview, select a device such as the **iPhone SE**:



3. Notice that for the most part it looks like the desktop layout has been scaled down. It's not the one column layout we want for mobile devices. That's one issue we need to fix. But first, notice that some text (such as the heading and first paragraph) has not been reduced as much as other text (such as the two columns). That's because some mobile browsers enlarge text they think is too small (if they think it doesn't break the layout). We don't want mobile browsers arbitrarily overriding some our font sizes, so let's disable that.
4. Switch back to your code editor.
5. Go to **Resume > snippets** and open a code snippet we prepared for you, **text-size-adjust.css**.
6. Hit **Cmd–A** (Mac) or **Ctrl–A** (Windows) to select all the code.
7. Hit **Cmd–C** (Mac) or **Ctrl–C** (Windows) to copy it.
8. Close the file.
9. At the top of **main.css**, paste the new code above the **body** rule:

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

10. Save the file.
11. Switch back to Chrome and reload the page.
12. The text is no longer being enlarged, so it looks like the desktop layout scaled down to fit a mobile phone. Now that the browser won't be deciding what text it might enlarge, we must get it to display the layout appropriately for small screens, instead of scaling down the desktop layout.

The Viewport Meta Tag

Mobile devices assume websites were designed for desktops. They render the site on a large viewport (980px) and scale it down to fit their smaller screen. For responsive sites we must add a **meta** tag to tell it to render the page at the actual pixel width of the device, instead of the default 980px.

1. Keep the browser open in device mode so we can come back to preview the changes we're about to make.
2. In your code editor, switch to **resume.html**.

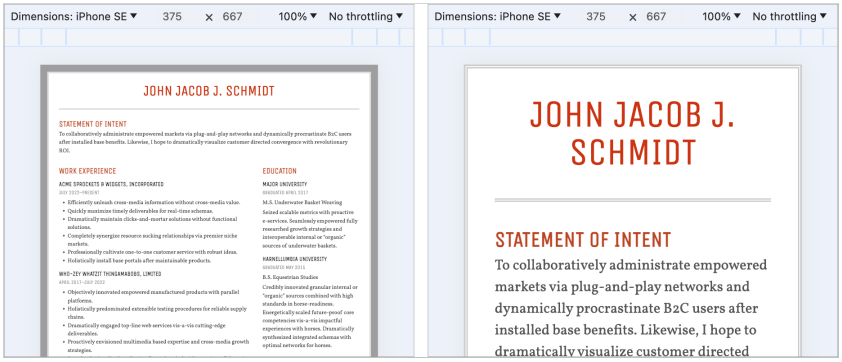
3. In the **<head>** tag, add the following bold code:

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>John Jacob J. Schmidt's Resume</title>
```

This tells mobile browsers to make the width of the **viewport** equal to the width of the **device**. Like other meta tags, it goes in the head tag.

4. Save the file and reload the page in Chrome (which should still be in device mode).

Now it should be the correct one column mobile appropriate layout! The viewport meta tag does not affect **desktop** browsers, only **mobile** browsers which employ the scaling behavior. As shown below, on the left is how the page looked before the meta tag, and on the right we see it with the meta tag!



Optional Bonus: Adjusting Text Size Across Screens

The **John Jacob J. Schmidt** heading (an h1) looks good on desktops, but is too big on phones. Let's adjust that.

1. Switch back to **main.css** in your code editor.
2. In the **h1** rule, change **font-size** to **32px**.
3. To make the text larger on desktops only, inside the media query add the following new rule (shown below in bold):

```
@media (min-width: 700px) {
    body {
        margin: 20px;
    }
    h1 {
        font-size: 42px;
    }
```

4. Save **main.css** and reload **resume.html** in Chrome.

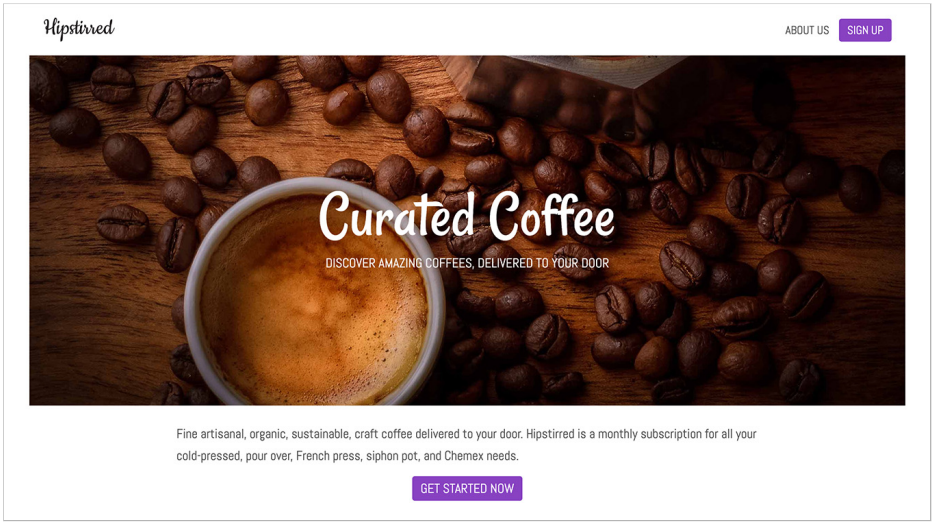
   The phone layout now looks perfect, but let's check the desktop size one last time.

5. In the upper left of the DevTools panel, click the **Toggle device toolbar** button to close the mobile simulator.

6. Close the DevTools panel.

   The desktop layout also looks great!

Exercise Preview



Exercise Overview

In this exercise, you'll learn how to round corners with CSS to create a button-like appearance on text links.

1. We'll be using a new folder of provided files for this exercise. Close any files you may have open in your code editor to avoid confusion.
2. For this exercise we'll be working with the **Hipstirred CSS Buttons** folder located in **Desktop > Class Files > Web Dev Class**. You should open that folder in your code editor if it allows you to (like Visual Studio Code does).
3. Open **index.html** from the **Hipstirred CSS Buttons** folder.
4. Preview **index.html** in a web browser.

   NOTE: We recommend leaving **index.html** open in the browser as you work, so you can reload the page to see the changes you make in the code.

Simple CSS Buttons

We'd like the sign up link in the nav to be a prominent call-to-action. Ideally, the link should look and feel like a bright, eye-catching button. With the power of CSS background-color, padding, and border-radius, we can get the job done nicely.

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

2. Add the following new rule below the **.wrapper** rule:

```
.button {
   background-color: #8842c2;
}
```

3. Save the file.
4. Switch to **index.html** in your code editor.
5. Add the **button** class to the sign up link, as follows:

```
<ul>
   <li><a href="about.html">About Us</a></li>
   <li><a href="signup.html" class="button">Sign Up</a></li>
</ul>
```

6. Save the file.
7. Return to the browser and reload **index.html**. This needs a bit of work. The text is too dark for the background and the button needs padding.
8. Return to your code editor and switch to **main.css**.
9. Add the following new properties to the rule for **.button**:

```
.button {
   background-color: #8842c2;
   color: #fff;
   padding: 10px;
}
```

10. Save the file.
11. Return to the browser and reload **index.html**. This is starting to really look like something. Let's round the button's corners.
12. Return to your code editor and add the following to the **.button** rule:

```
.button {
   CODE OMITTED TO SAVE SPACE
   padding: 10px;
   border-radius: 4px;
}
```

13. Save the file.
14. Go to the browser and reload **index.html**. Rounded, but let's fine-tune the padding.

15. Return to your code editor and update the padding for **.button** as follows:

```
.button {
   background-color: #8842c2;
   color: #fff;
   padding-top: 6px;
   padding-bottom: 6px;
   padding-right: 12px;
   padding-left: 12px;
   border-radius: 4px;
}
```

16. Save the file.

17. Return to the browser and reload **index.html**. Very button-like!

Optional Bonus: Creating a Second Call to Action

Let's reuse the button class for another link just under the hero section.

1. Return to your code editor and switch to **index.html**.

2. Near the start of the **main** section, add the a new div tag with a link inside as shown below in bold:

```
<p>Fine artisanal, organic, sustainable, craft coffee delivered to your door.
Hipstirred is a monthly subscription for all your cold-pressed, pour over,
French press, siphon pot, and Chemex needs.</p>

<div class="call-to-action">
   <a href="signup.html" class="button">Get Started Now</a>
</div>

<h2>Amazing Coffee, Conveniently Delivered Directly to You</h2>
```

3. Save the file.

4. Return to the browser and reload **index.html**.

5. The **Get Started Now** link is there and looks somewhat like the other button, but it still has the default underline and is not all caps like the other button (in the header). It would also look nicer if it were horizontally centered in the page.

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

7. Add the following new rule below the rule for **.button**:

```
.call-to-action {
   text-align: center;
}
```

8. Next let's remove the underline and make it all caps. Add the following new properties to the rule for **.button**:

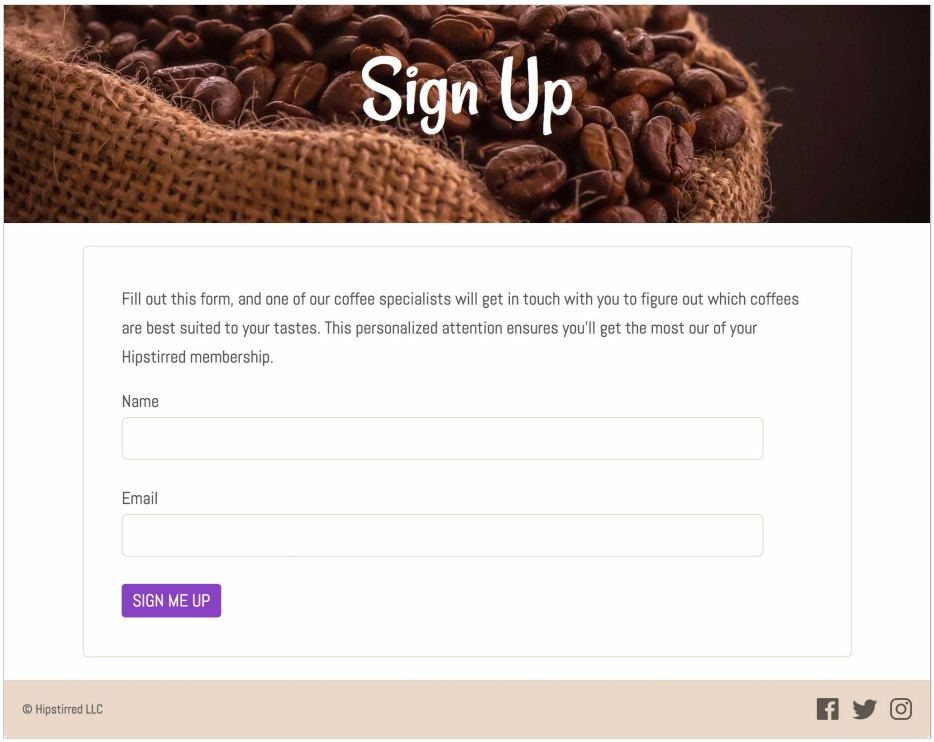
```
.button {
    CODE OMITTED TO SAVE SPACE
    border-radius: 4px;
    text-transform: uppercase;
    text-decoration: none;
}
```

9. Save the file.
10. Return to the browser and reload **index.html**. Good, but the call to action button could use more space below it.
11. Return to **main.css** in your code editor and add the following property to the rule for **.call-to-action**:

```
.call-to-action {
    text-align: center;
    margin-bottom: 60px;
}
```

12. Save the file.
13. Return to the browser and reload **index.html**. Looking good!

Exercise Preview



Exercise Overview

Forms allow you to collect information about your visitors so you can better serve their needs. In this exercise, you'll learn to code and style a basic form to collect a user's name and email.

Getting Started: The Form Tag

1. We'll be using a new folder of provided files for this exercise. Close any files you may have open in your code editor to avoid confusion.
2. Navigate to the **Desktop** and go into the **Class Files** folder, then **Web Dev Class** folder, and find **Hipstirred Form Style**. Open the whole folder in your code editor if it allows you to (like Visual Studio Code does).
3. Open **signup.html** from the **Hipstirred Form Style** folder.
4. Preview the file in a browser. Between the paragraph and the footer we'll add form fields to ask for the user's name and email.

   NOTE: Leave the page open in the browser as you work, so you can reload the page to see the changes you make.

5. Return to **signup.html** in your code editor.
6. Around line 26, wrap a form tag around the paragraph:

```
<main>
   <form action="">
      <p>Fill out this form, and one of our coffee specialists will get in
touch with you to figure out which coffees are best suited to your tastes.
This personalized attention ensures you'll get the most our of your Hipstirred
membership.</p>
   </form>
</main>
```

NOTE: For the form to be processed correctly, you must make sure to place all form elements inside the **form** tag. The **action** attribute specifies where to send the form data when the user hits the submit button. We'll leave it blank for now.

Adding Name & Email Inputs to the Form

We create a text field with the input element. Let's add a couple to our form so we can request a person's name and email.

1. Below the paragraph inside the form tag, add the following code:

```
<form action="">
   <p>Fill out this form, and one of our coffee specialists will get in touch
with you to figure out which coffees are best suited to your tastes. This
personalized attention ensures you'll get the most our of your Hipstirred
membership.</p>

   <input type="text">
</form>
```

NOTE: The **input** element creates something a person can use to enter data. The **type** attribute specifies what kind of control the browser will create. In this case we want a **text** field, but here are a few other examples of other input types: checkbox, radio, submit, hidden, tel, time, and date.

2. Save the file, return to the browser, and reload the page.

You should see a small text field, but how is the user to know what they should type here? Let's add a label.

3. Return to **signup.html** in your code editor.

4. Add the following code above the input element, as shown in bold below:

```
<form action="">
   <p>Fill out this form, and one of our coffee specialists will get in touch
with you to figure out which coffees are best suited to your tastes. This
personalized attention ensures you'll get the most our of your Hipstirred
membership.</p>

   <label>Name</label>
   <input type="text">
</form>
```

5. Save the file, return to the browser, and reload the page.

6. Looks better, but click on the label that says **Name**. Nothing happens. For proper accessibility, the label should be associated with its input.

7. Return to the code and make the following changes:

```
   <label for="nameField">Name</label>
   <input type="text" id="nameField">
</form>
```

8. Save the file, return to the browser, and reload the page.

9. Click on the label that says **Name** and the cursor should automatically be placed in the input field! This is a nice usability improvement overall, but especially important for users with visual impairments so their screen readers will now know which text to announce for which input.

10. Return to **signup.html** in your code editor.

11. Add the following new attribute to the **input** element:

```
<input type="text" id="nameField" name="nameField">
```

    While the ID you added earlier allows us to bind the label to the input, the **name** attribute will be used when sending the form data to a script that will process the form.

    Now that we have this label and input functioning the way we want them to, let's copy the code and create a second input for email.

12. Select the following two lines of code:

```
<label for="nameField">Name</label>
<input type="text" id="nameField" name="nameField">
```

13. Copy the code.

14. Paste the code directly underneath like so:

```
<label for="nameField">Name</label>
<input type="text" id="nameField" name="nameField">

<label for="nameField">Name</label>
<input type="text" id="nameField" name="nameField">
```

15. Edit the code you just pasted to make the following changes (in bold):

```
<label for="nameField">Name</label>
<input type="text" id="nameField" name="nameField">

<label for="emailField">Email</label>
<input type="email" id="emailField" name="emailField">
```

NOTE: As we mentioned earlier, there are numerous values you can use for an input's **type**. We're using **email** here to improve the usability of this form. On a mobile device, you'll see an on screen keyboard more appropriate for entering emails, and it will be easier to validate this field.

16. Save the file, return to the browser, and reload the page.
    - You should have **Name** and **Email** labels, each with a text field.
    - When you click on the **Email** label, the cursor should be placed into its appropriate text field.
    - The labels and text fields are all on one line, but we'd rather have them stack. We'll do that, and improve their appearance in a little bit.

Adding a Submit Button

We need an input that will allow a user to submit their info.

1. Return to **signup.html**.
2. Inside the form, below the email input, add the following code shown in bold:

```
    <input type="email" id="emailField" name="emailField">

    <button>Sign Me Up</button>
</form>
```

NOTE: We can use **<input type="submit" value="Sign Me Up">** instead of the button. Both will submit the form, but a **button** is more flexible because it may contain HTML content. That gives us more options should we want to add **img**, **em**, **strong**, or other tags.

3. Save the file, return to the browser, and reload the page. You should now have a **Sign Me Up** button.

Styling the Form, Input, & Label

Let's make this form look a bit nicer.

1. Return to your code editor and open **main.css** from the **Hipstirred Form Style** folder.
2. We'll start by creating a visual delineation for the form, so it stands out a bit more. Below all the styles, at the very bottom add the following new rule (with a comment to separate the form styles from the rest):

```
/* form styles */
form {
   border: 1px solid #e9d8c8;
   border-radius: 6px;
   padding: 5%;
}
```

3. Save the file, return to the browser, and reload **signup.html**. Now the form has a slightly rounded corner border around it.

   Next let's make the Name and Email fields stack on top of each other. Currently they are inline elements, but we can change them to block and they will stack!

4. Return to **main.css** in your code editor.
5. Add the following new rule below the **form** rule:

```
label, input {
   display: block;
}
```

   NOTE: This rule applies the same styling to both the **label** and **input** elements.

6. Save the file, return to the browser, and reload **signup.html**. All the labels and inputs in the form now stack. There are some issues though.
7. Click into one of the text inputs and type a few characters to see:
   - The text is small and touches the edge of the input. It would look much better with some padding inside the element.
   - The text you just typed is not the same font as the rest of the page.
   - The **Sign Me Up** button is the wrong font as well.
8. Return to **main.css** in your code editor.
9. Add the following new rule below the **label, input** rule:

```
input, button {
   font-family: "Abel", sans-serif;
   font-size: 19px;
}
```

10. Let's make the text inputs wider, add some padding inside them, add some extra space around them, and apply a border that matches our form's border. Add the following new rule below the **input, button** rule:

```
input {
   width: 90%;
   padding: 10px;
   margin-top: 5px;
   margin-bottom: 30px;
   border: 1px solid #e9d8c8;
   border-radius: 6px;
}
```

11. Save the file, return to the browser, and reload **signup.html**.
12. Click into one of the text inputs and type a few characters to see:
    - The text is now bigger and the font matches the rest of the page.
    - The padding opens up the text field so it's easier to click/tap on, and the type is easier to read when it does not touch the edge.
    - The wider text fields provide more space to see what you type.
    - The bottom margin separates the text field from the element below.

    Much better!

Styling the Button

Let's make the button look better. We already have a **.button** class we've used for the main call to action on the home page and the Sign Up button in the navigation. Let's reuse that for the form's button.

1. Return to your code editor and switch to **signup.html**.
2. Find the **button** element and edit it as follows:

```
<button class="button">Sign Me Up</button>
```

3. Save the file, return to the browser, and reload **signup.html**.
    - Look closely and you'll probably see a border of some kind. That's the browser's default styling (which varies across browsers).
    - Click on the button and in some browsers (such as Chrome) you may see an outline appear on the button. That's more browser default styling.
    - Notice that when you hover over the button, the cursor does not change to a pointer. Once again that's browser default styling.

    Luckily we can change all these things with CSS.

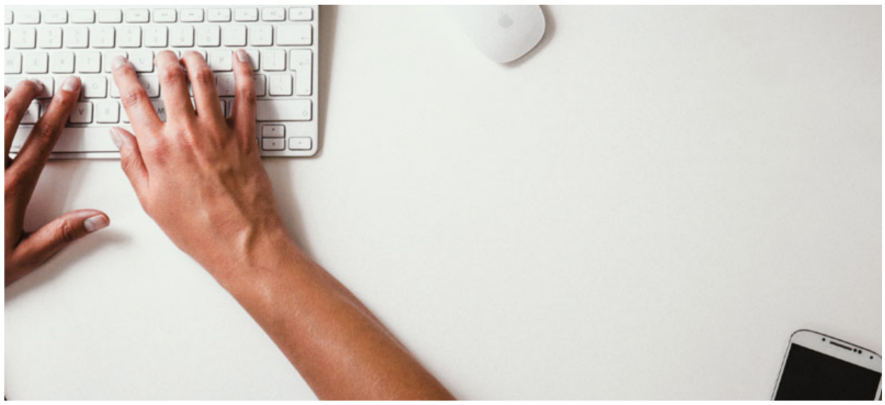
4. Return to your code editor and switch to **main.css**.
5. Add the following new rules below the **input** rule:

```
button {
   border: 0;
   cursor: pointer;
}
button:hover, button:focus {
    background-color: #9c5bd1;
}
```

6. Save the file.
7. Return to the browser to reload **signup.html** and enjoy the look and feel of your button and styled form.

   NOTE: To make the form function would require some back-end coding such as back-end JavaScript, PHP, Ruby on Rails, etc. which is beyond the scope of this book.

Exercise Preview



Contact Revolution Travel

Email Us
hello@revolutiontravel.com

Exercise Overview

In this exercise, you'll learn how to create an email link so visitors can contact you. You'll also learn a best practice for hiding your email address from spammers.

Creating an Email Link Using the Mailto Protocol

1. We'll be using a new folder of provided files for this exercise. Close any files you may have open in your code editor to avoid confusion.
2. For this exercise we'll be working with the **Revolution Travel Contact** folder located in **Desktop > Class Files > Web Dev Class**. You should open that folder in your code editor if it allows you to (like Visual Studio Code does).
3. Open **contact.html** from the **Revolution Travel Contact** folder.
4. Preview **contact.html** in a browser to see the contact info we have so far. Let's add their email address.
5. Add the email for our visitors to see:

```
<p>
   <strong>Email Us</strong><br>
   hello@revolutiontravel.com
</p>
```

While this will display the email address for people to see, we want to make this into a link people can click that will open an email pre-addressed to this email.

6. There is a **mailto** protocol for launching an email application from a link. Add the following code to make the text a functional email link:

```
<p>
   <strong>Email Us</strong><br>
   <a href="mailto:hello@revolutiontravel.com">hello@revolutiontravel.com</a>
</p>
```

7. Save the file and preview **contact.html** in a browser.

8. Click on the email link. If the computer is set up properly, it should launch an email program. If people visiting the website don't have an email program set up, or if they use web-based email like Gmail, they'll still know the address by looking at the webpage.

Creating a Spambot-Resistant Email Link

There's a catch to using the mailto protocol. Spammers have email harvesters that scan websites and find these addresses. Luckily we can use JavaScript (a scripting language used for interactive content on websites) to obfuscate the email link so spammers can't read it.

We can write a script that essentially breaks the **mailto** into pieces in the code and assigns nonsense variables to each piece. Email harvesters are rarely smart enough to understand the script and piece it back together, effectively hiding the address from spammers. If you were to Google **spambot-resistant email link**, you'll see that there are a ton of resources for you about writing your own script. There are even stock scripts you can edit and use, as well as online tools for generating a script.

1. We have provided some JavaScript code that you can easily edit and use in any project. Return to your code editor and open **spam-proof-email.html** from the **snippets** folder (in the **Revolution Travel Contact** folder).

2. Select all the code (**Cmd–A** (Mac) or **Ctrl–A** (Windows)).

3. Copy it (**Cmd–C** (Mac) or **Ctrl–C** (Windows)).

4. Close the file.

5. You should be back in **contact.html**.

6. Select the email link you just wrote and paste the JavaScript over it.

7. Edit the script by changing your-name to **hello** and your-domain to **revolutiontravel** as shown below in bold:

```
<p>
   <strong>Email:</strong><br>
   <script>
      var eDone = 'hello'+'@'+'revolutiontravel'+'.'+'com';
      var eSubject = '';  // optional

      document.write('<a '+'hre'+'f="mai'+'lto:'+eDone+'?
sub'+'ject='+eSubject+'">'+eDone+'</'+'a>');
   </script>
</p>
```

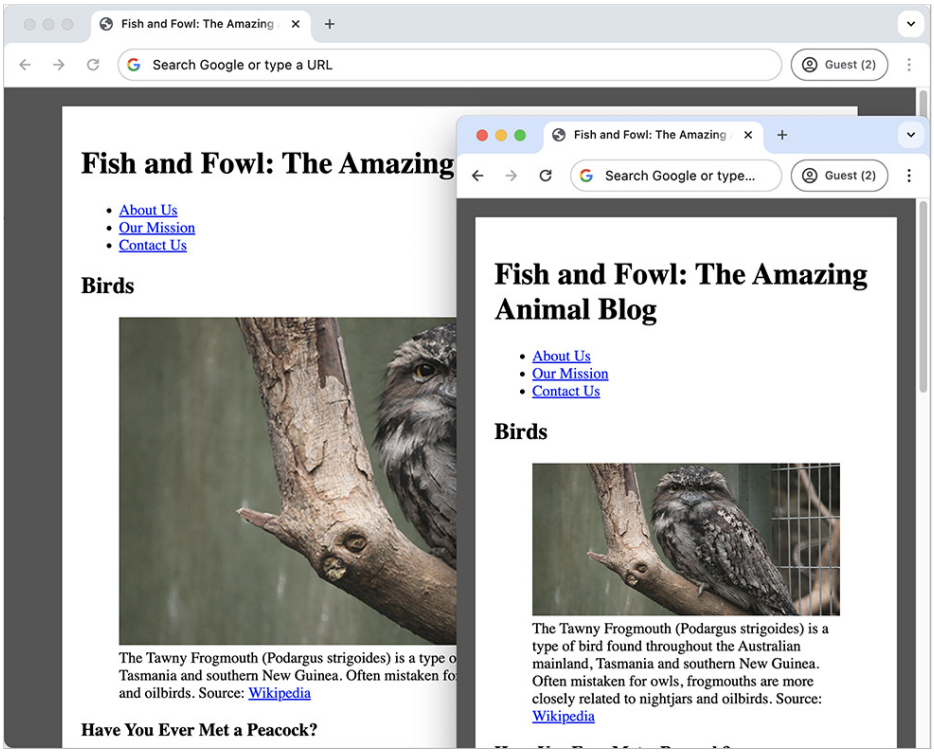
8. The email's subject line can be prefilled by adding **?subject=Some Text Here** after the email address in a **mailto:** link. This is an option in our supplied JavaScript, so let's see how it works. Inside the single quotes, type **Travel Inquiry** as shown below in bold:

```
var eSubject = 'Travel Inquiry';  // optional
```

9. Save the file and preview **contact.html** in a browser.

10. Click the email link. It should work just like the mailto, but now it will be more difficult for spambots to harvest the email address. Notice it even has the custom subject line: **Travel Inquiry**!

Exercise Preview



Exercise Overview

In a previous exercise you added semantic section tags to **Fish and Fowl: The Amazing Animal Blog**. You didn't style the page though, so now is your chance to add your own stylistic flair with CSS.

Getting Started

1. Preview **semantic-elements-partially-styled.html** from the **Structural Semantics** folder in a web browser.

   Notice that we added some very basic styling to the page. The webpage is structurally complete, but it lacks personality. Your challenge is to use what you've learned about CSS to improve the visual appearance.

2. In your code editor, open **semantic-elements-partially-styled.html** from the **Structural Semantics** folder.

   Take a moment to review the code and make note of the different semantic sections.

3. Using CSS, give the webpage some character! Continue reading this exercise for some ideas of what you can do.

Tips & Ideas

Here are a few ideas you can try:

- Modify the page colors.
- Make the section headers stand out with different font sizes, font family, or color.
- Remove the default bullet style for the list.
- Try making the navigation horizontal rather than stacking.
- Give the links interactive style with pseudo-classes.
- Use Google Web Fonts.
- Put your styles in an external style sheet and link it in the webpage.
- Give the wrapper rounded edges using border-radius.
- Test the mobile view using Chrome's Inspector and adjust how the page scales.

Exercise Preview

Exercise Overview

It's important to practice your newly learned coding skills, so in this exercise we challenge you to build a small website without step-by-step instructions.

Getting to Know the Project

The site we challenge you to code is for **NAPS (National Association to Promote Siestas)**. In the **Web Dev Class** folder, you'll find a **NAPS** folder with the following:

- **images**: In this folder you'll find hi-res (Retina) quality photos and the NAPS logo. These have been optimized and are ready for use in the website.
- **page-designs-psd**: In this folder you'll find an editable Photoshop file for each of the four pages of the site.
- **page-designs-jpg**: In case you don't have Photoshop, in this folder you'll find a JPEG file for each of the four pages of the site.
- **text**: In this folder you'll find the text for each page, and the copyright.

Getting Started

We suggest you build the homepage first, and then move on to the other pages.

If you have Photoshop, open the Photoshop page design files (.psd). Inspect the various elements to learn type size, colors, etc.

If you don't have Photoshop, refer to the JPEG designs instead. Because you can't inspect the various elements to learn about them, below we've included some of the specifications we use in the provided designs.

- Google Font: **Lato**
- Blue for the Navigation and Headings: **#00b0dc**
- Darker Blue for the Navigation's Bottom Border: **#009bc2**
- **Headings (h1):** Lato Bold, 34px size, 41px line-height
- **Sub-Headings (h2):** Lato Bold, 28px size, 34px line-height
- **Paragraphs:** Lato Regular, 17px size, 29px line-height
- **Large Paragraphs**: Lato Light, 24px size, 36px line-height

---

Exercise Overview

To get practice and get better at coding, you need to use what you've learned so far to make your own website from scratch.

If you will be taking other web coding classes at Noble Desktop, as you continue learning more techniques you can either add them to this website you build, or better yet create more websites that incorporate them.

Steps to Building Your First Website

To build your first website, here's what you need to do:

Step #1: Figure Out What Site You'll Build

If you you're in Noble Desktop's Full-Stack Certificate Program, we have a specific type of website for you to build. So skip over these ideas and refer to the section specifically for you.

For these ideas, you don't have to build a large website, it's good to start small.

- Restaurant
  - Be sure to include a separate page for their food menu(s)
  - Make sure the location and phone number is prominent (and clickable)
- Any type of service business: dog walker, lawyer, investment management company, etc.
  - Make a page for each service they offer (you don't have make a lot of services), but each service would need it's own page for SEO (Search Engine Optimization) purposes
  - Have an about page
- A website promoting an iOS/Android mobile app
- A blog for a specific niche
  - Eventually this would be turned into a WordPress theme, but the first step is to design/build the various types of pages with HTML and CSS.
  - Minimum pages to build: homepage (you may list blog posts here, or make a dedicated blog page for those), blog post page
- Website for an event (like a conference, etc.)
  - For accepting registrations you could use a service like Eventbrite, so don't worry about creating that functionality
- Website for person, band, musician, motivational speaker, entertainer, author, etc.

Step #2: Design & Code

If you know a design app (such as **Figma** or **Sketch**) you can mock up your design there first, then start coding.

If you're not a designer, refer to existing websites for their style. Replicate their style as you code your website.

Step #3: Get a Domain Name & Web Hosting

We recommend **scalahosting.com** or **bluehost.com** where you'll get your domain name free for the fist year. After the first year you'll have to pay yearly for the domain name, and the monthly web hosting costs will increase to their normal rate (you get a discount on the first year).

You'll have to continue paying as long as you want to have your domain name and web hosting. Luckily it's not too expensive and is essential for anyone working in tech or design.

Step #4: Upload the Site to Make it Live

Use an FTP app like **Cyberduck** from **cyberduck.io/download** to upload your site to make it live.

If you're be making a portfolio website, put the practice website(s) you make into subfolders. For example, let's say your portfolio website folder structure is this:

- **index.html**
- **css** folder containing **main.css**
- **someproject** folder
  - **index.html**
  - **css** folder containing **main.css**
- **anotherproject** folder
  - **index.html**
  - **css** folder containing **main.css**

You could have a links to see your projects like:
yourwebsite.com/someproject
yourwebsite.com/anotherproject

If You're in Noble Desktop's Full-Stack Program

Instead of building one of the sites above, we have a specific type of site for you to start working on. When you get to the back-end portion of the program, you'll add the back-end functionality. For now you should start on the front-end HTML and CSS, so it will be ready for when you need it.

You must do this project first. If you finish this and have extra time, then you can go back and do any of the other front-end projects mentioned above.

In the back-end portion of the Full-Stack Certificate Program you'll work on your own project and a movie search website (similar to Netflix). So for your project you need to choose a type of website other than movies. Here are some ideas:

Topic Ideas

- Recipes
- Drinks
- Animals
- Sports
- Apartments for rent
- Jobs
- Classifieds (things for sale, etc.)
- Something else of our choosing that would work similarly, where you can hit a list of content and load in the info.

Pages & Features You'll Need

Here are the types of pages and functionality you'll need to have in your website. For now, focus on designing and coding the HTML/CSS. Don't worry about making the features actually work, because that's what you'll learn in the back-end portion of the full-stack program.

- Homepage
- Join (create an account)
- Login (to an existing account)
- Search results page (use static content for now, you'll make it dynamic and functional later)
- Detail page displaying all the info about an item (you'd see this page after clicking an item in the search results)
- Page to create new items (with form fields for the item details (example: if it were products that would be product name, price, etc)
- Page to update an existing item (with form fields for the item details)
- Have a search field in the navbar
- Have a way to save items as a favorite
- Favorites page:
  - Displays items you've saved
  - Have a way (like a button) to remove an item from the favorites list

Noble Desktop Training

Learn live online or in person in NYC

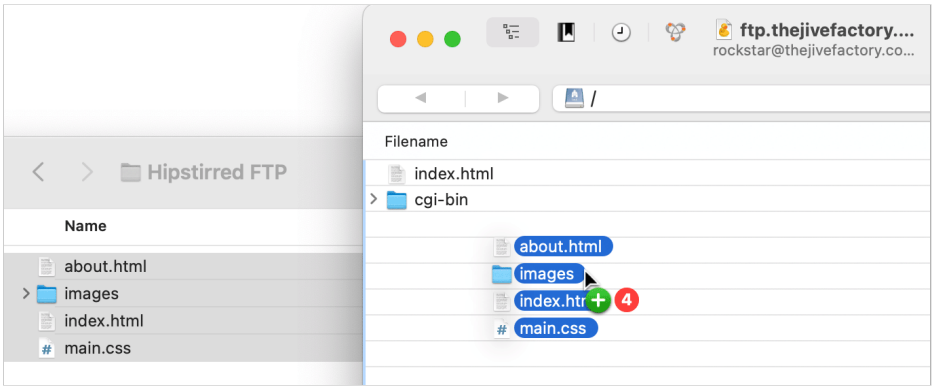
Front-End Web Development
Full-Stack Web Development
JavaScript
Python
Software Engineering
Data Science & Data Analytics
SQL
WordPress

Motion Graphics & Video Editing
Adobe Premiere Pro
Adobe After Effects
InDesign, Illustrator, & Photoshop
Web, UX, & UI Design
Figma
Digital & Social Media Marketing
and much more...

---

nobledesktop.com

Exercise Preview



Exercise Overview

To make your website visible to the world, the files you've coded on your computer must be copied to a web server. In this exercise, you'll learn how to use FTP (File Transfer Protocol) to upload your local files to a web host's remote server.

Web Hosts & Domain Names

Live websites require two things: a domain name and a web host.

1. A **domain name** is the website's address (such as amazon.com, google.com, codepen.io, or bit.ly).
2. A **web host** holds the files (HTML, CSS, images, etc.) that make the website work.

   When someone goes to a domain name, it directs their browser to the web host, which in turn sends the files to the browser for viewing.

   A single company can be both the domain name host and web host, or you can use two different companies. When getting started, it's easier to use one company. There are many companies that offer affordable web hosting and domain names, so you'll have to choose a web host to use.

Preparation: Things You'll Need to Upload a Website

1. **Files to Upload**: The HTML, CSS, images, etc. of your website.

2. **FTP Client**: This is an app for your Mac or PC that copies files between two computers (your computer and the web server). There are many FTP clients, but we recommend **Cyberduck** because it's free and cross-platform. (All FTP clients work on the same basic principle, uploading local files to a remote webserver.)

   Download and install **Cyberduck** from **cyberduck.io/download** using the download links for Windows or macOS. Do NOT click any download links in the ad at the top!

3. **FTP Upload Information**: Your web host will provide you with an address, username, and password to FTP your files. You should be able to find this information when you log into your hosting account. If you can't find it, ask your web host and they should be able to help you. Make sure you know the following:

   - **FTP Server Address**: Something like **ftp.yourwebsite.com**
   - **FTP Username**
   - **FTP Password**

Getting Started

1. Before you upload anything, let's see how the live website currently looks. Open a browser and type in the address bar type in your domain name, something like **`http://yourwebsite.com`**
2. Hit **Return** (Mac) or **Enter** (Windows) to go to the page. What you'll see depends on your host. You will soon be replacing whatever is currently there.

Going Live

1. Launch **Cyberduck** (the FTP app we said to install earlier in this exercise).
2. Click the **Open Connection** button at the top of the main window.
   - Next to **Server**, enter the FTP server address provided by your web host, something like **`ftp.yourwebsite.com`**
   - Next to **Username**, enter the username provided by your web host.
   - Next to **Password**, enter the password provided by your web host.
3. Verify that you entered the correct info for **server**, **username**, and **password**.
4. To connect to the server, click **Connect**. If you get a warning that you have an **Unsecured FTP connection**, hit **Continue**.

   NOTE: If Cyberduck cannot connect, double-check the info and make sure there are no extra spaces.

5. Once connected, you will see the window that lists the files for your site. Depending on your web host, you may need to go into a folder (such as **public_html**).

   TIP: Once connected, you can save this information for future use by choosing **Bookmark > New Bookmark**.

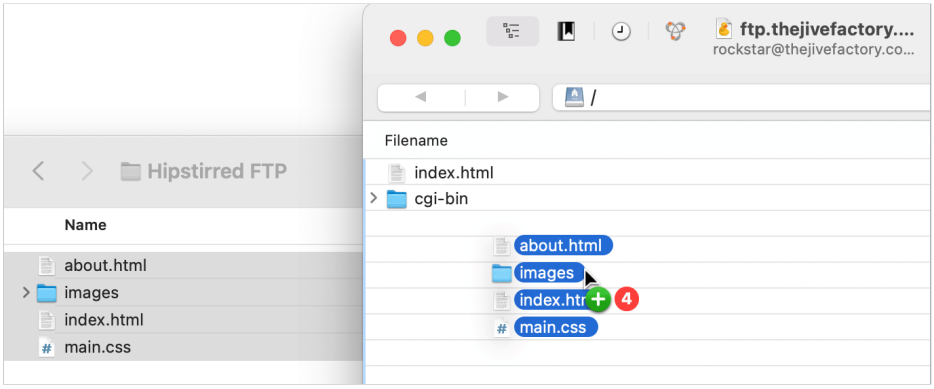
6. You're now ready to upload your files. In order to do this, move the Cyberduck window to one side so you'll be able to see your local file folder next to it.

7. Keep the Cyberduck window open, but switch to the Desktop and navigate to your website's files.

8. Arrange the Desktop window and the Cyberduck window so you can see them both.

9. To upload your files (also known as **going live**):

   - Select the files in your Desktop window.
   - Drag the selected files and folders from your computer's window onto the Cyberduck window. TIP: Be careful to drop your files into the empty white space, NOT one of the existing folders.



10. If you get another **Unsecured FTP connection** warning, hit **Continue** once more.

11. A **Transfers** window will appear, showing the progress of the upload. (It may be running in the background.) If you get an **Overwrite** warning about overwriting the pre-existing index page, hit **Continue** to overwrite those files with your own.

12. After the upload has finished, you can close the Transfers window.

13. You now have a live site! To see it, switch to a browser and go to your domain name, something like **`http://yourwebsite.com`**

    NOTE: You may have to reload the browser to see the newly uploaded site.

Making Changes

If you need to edit your website, you should first make changes to the local files on your computer and then upload them to the local server using FTP once more. Here is a brief recap of the steps to make your files live:

- Return to **Cyberduck**.
- If you are not still connected to your remote server, press the **Open Connection** button.
- Enter your **Server**, **Username**, and **Password** and click **Connect**.
- On your computer navigate to the local site folder, and arrange this window and the Cyberduck window so they are side-by-side.
- To upload the updated file, drag the file from your computer's window onto the Cyberduck window.
- If you want to upload more than one file, **Cmd–click** (Mac) or **Ctrl–click** (Windows) on each file and then drag them all over to the Cyberduck window. (If you click on one file and **Shift–click** another file, all the files in between them will also be selected.)

  The new files will replace the older files. Remember, whatever you do to the live remote site CANNOT be undone, so upload carefully!

Links to Reference Websites, Online Tools, & More

Because there are so many wonderful online tools, blog writeups, and more, we decided to make a site with all the links instead of printing them in this book. Not only does this save some typing, but it also will allow us to keep it more up-to-date.

Check out our suggested resources at **nobledesktop.com/webdev-links**

HTML vs. XHTML Syntax

HTML can be written using HTML or XHTML syntax. There are only minor stylistic differences between the two. We prefer HTML syntax, which is what we use in this book. It's even a bit less typing.

If you prefer to code using XHTML syntax, here are some things to keep in mind:

- You must specify a document type at the top of the file.
- All tags must be lowercase.
- All attributes must have quotes.
- All tags must have a close tag `<p>this is a paragraph</p>`
- If tags do not have a close tag, such as a `<br>` or `<img>` tag, they must be written as self-ending: `<br />` or `<img src="image.gif" />`

Web graphics are made of pixels or vectors:

- Pixels are square blocks you can see if enlarged too much.
- Vectors are mathematically based so they look great even when enlarged.

All the following formats are pixel-based, except for SVG.

JPEG (Joint Photographic Experts Group)

JPEG is the best format for photographic images, because it supports millions of colors and compresses color gradations without much obvious loss in quality. More compression yields a smaller file, but also creates more visible artifacts, such as loss of detail and the appearance of unsightly square blocks.

PNG (Portable Network Graphics)

PNG is the best format for pixel-based graphics with partial transparency. There are two different types of PNG:

- **PNG-24** supports millions of colors. Their pixel-perfect image quality means the file size is often large.
- **PNG-8** supports up to 256 colors, so they are typically a small file size.

GIF (Graphic Interchange Format)

GIF is the only image format that supports animation (with no coding). We rarely use GIF for static images because PNG-8 gives you the same color palette options at a file size that is 5–25% smaller.

SVG (Scalable Vector Graphics)

SVG is a vector-based format that's ideal for graphics consisting of geometric shapes, such as logos and icons. SVG scale to any size without losing clarity, so they look great on high resolution (HiDPI or Retina) displays. SVG can be used as the source (src) for an `<img>` tag or the SVG code can be embedded into an HTML file. SVG can be animated with JavaScript code.

Once you create your site, you will want to be listed on the major web search engines. There are two types of listings: free and paid. If you are in a competitive field, you may need to pay to get listed well. However, it is possible with the use of strategic keywords that you can show up on Google, Bing, or Yahoo for free.

Before submitting your site to various engines/directories, you need to do a few things to the important pages on your site (those that you would like to be **indexed**, or found by the search engines).

Page Titles

Google will index the title of your pages and use those words as priority indexed words. In this case, the more information your title has, the better. For example, the title of your apple-picking farm website might be **Cornwall Farms**. But that does not mean anything to people who search the internet for **apple picking**.

What works better is this: **Apple Picking in New York State**

Try to make your titles as descriptive as possible, but keep them 60 characters or less.

Meta Tags

<meta> tags go within the <head> tag to provide information on your page. There are a variety of meta tags, but the most important is the **description** tag. A search engine may display the description as the summary of your site. Another meta tag is the **keyword** tag, but it's virtually irrelevant now.

Here is an example of how the meta tags might look for an apple picking site:

```
<head>
   <title>Apple Picking in New York State</title>
   <meta name="description" content="Cornwall Farms Apple Picking in Upstate
New York">
   <meta name="keywords" content="Cornwall Farms, Apple Picking, Upstate NY">
</head>
```

When someone performs a search, the site will usually show up in this context:
Apple Picking in New York State
Cornwall Farms Apple Picking in Upstate New York

Getting Listed on Google

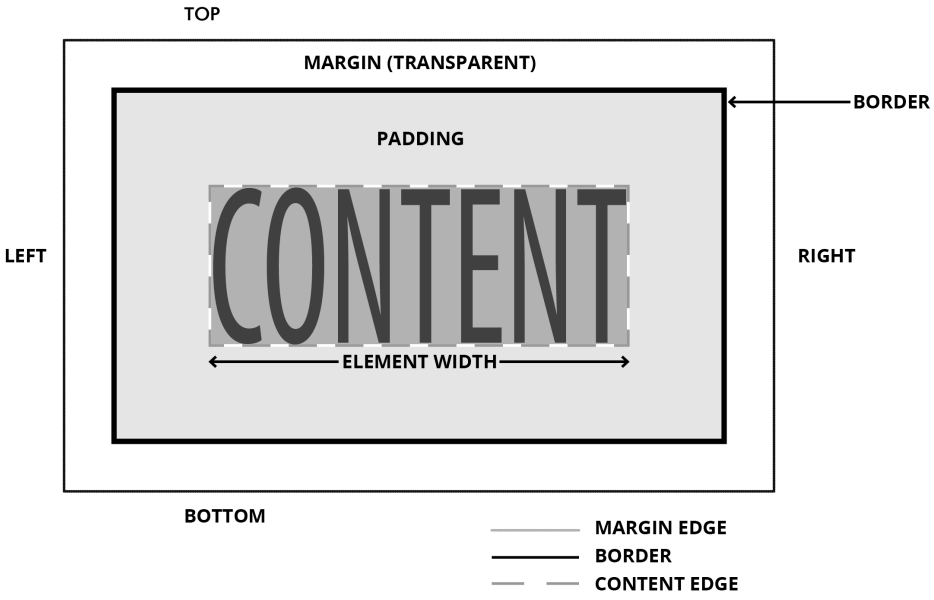
Google should eventually find your website, but to speed it up and control how Google sees your website, you should add it to your **Google Search Console**.

1. Go to **search.google.com/search-console** and get started with Google Search Console for your website.
2. Follow their instructions to verify your website and get it set up.

Learn More with Noble Desktop's SEO Training

Check out our classes at **nobledesktop.com/topics/seo-training-nyc** to learn more about SEO.

The Box Model Illustrated

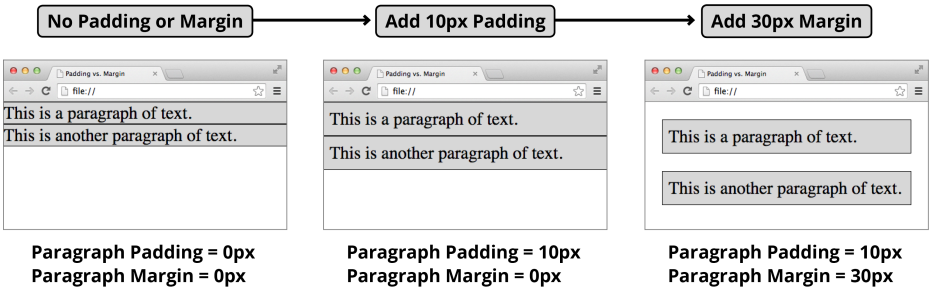


An important thing to note in the diagram above is that the width of an element is based on the content (text, image, etc.). That means when you add padding and borders, you increase the size of the element!

For instance, a 100px wide text block with 2px borders and 10px of padding is:
2px left border + 10px left padding + 100px width + 10px right padding + 2px right border = 124px total width of element

Padding vs. Margin Illustrated

In the example shown below, two paragraphs have the same style that has a gray background, black 1px border, and some margin and padding. The page's default margins have been removed so all the margin space you see is only from the style. We also beefed up the font-size so the paragraphs' text would be more legible.



All examples have Paragraph Border = 1px and Page Margin = 0px