Sass Training

with examples and
hands-on exercises

WEBUCATOR

Version: 2.0.3

The Authors

Brian Hoke

Brian Hoke is Principal of Bentley Hoke, a web consultancy in Syracuse, New York. The firm serves the professional services, education, government, nonprofit, and retail sectors with a variety of development, design, and marketing services. Core technologies for the firm include PHP and Wordpress, JavaScript and jQuery, Ruby on Rails, and HTML5/CSS3. Previously, Brian served as Director of Technology, Chair of the Computer and Information Science Department, and Dean of Students at Manlius Pebble Hill School, an independent day school in DeWitt, NY. Before that, Brian taught at Insitut auf dem Rosenberg, an international boarding school in St. Gallen, Switzerland. Brian holds degrees from Hamilton and Dartmouth colleges.

Nat Dunn (Editor)

Nat Dunn is the founder of Webucator (www.webucator.com), a company that has provided training for tens of thousands of students from thousands of organizations. Nat started the company in 2003 to combine his passion for technical training with his business expertise, and to help companies benefit from both. His previous experience was in sales, business and technical training, and management. Nat has an MBA from Harvard Business School and a BA in International Relations from Pomona College.

Follow Nat on Twitter at @natdunn and Webucator at @webucator.

Class Files

Download the class files used in this manual at https://static.webucator.com/media/public/materials/classfiles/SAS101-2.0.3.zip.

Errata

Corrections to errors in the manual can be found at https://www.webucator.com/books/errata/.

Table of Contents

LESSON 1. Introduction/Why Sass?.....1
- The Problem Sass Solves.....1
- Benefits of Sass.....8
- Sass Syntax.....9
- How to Use Sass.....10
- Coming Attractions.....15

LESSON 2. Organizing and Including Sass Code.....17
- Sass Partials.....17
- Partials/@import Example.....18
- Sass Watch.....21
- **Exercise 1: Using Partials.....23**
- Code Organization.....25
- The Sass Blog.....27
- The Sass Blog: With Sass Partials.....27
- **Exercise 2: Extending the Sass Blog.....39**
- More Styling for the Blog: Syntax Highlighting.....46
- **Exercise 3: Adding Syntax Coloring.....47**

LESSON 3. Nesting.....53
- Nesting.....53
- Referencing the Parent Selector with & 54
- Nested Properties.....55
- Nesting Example.....56
- **Exercise 4: Using Nesting.....63**
- Nesting and Media Queries.....66
- How Much to Nest.....72
- **Exercise 5: Using Nesting with the Sass Blog.....74**
- Nested @import 77

LESSON 4. Sass Variables.....79
- Variables.....79
- Variable Naming Guidelines.....80
- Variable Scope.....81
- **Exercise 6: Using Variables in Sass.....87**
- When to Use Variables.....91
- **Exercise 7: More Variables.....93**
- **Exercise 8: Adding Variables to the Blog.....99**
- Advanced Sass.....102

LESSON 1
Introduction/Why Sass?

Topics Covered

☑ What Sass is and how it works.

☑ The benefits of using Sass to author CSS.

☑ The two styles of Sass syntax.

☑ The major features of the Sass language.

Introduction

Sass, or Syntactically Awesome Style Sheets, is a CSS *preprocessor* that makes authoring, updating, and maintaining Cascading Style Sheets (CSS) easier. Sass is a strict superset of CSS: all CSS code is valid Sass code. But Sass contains additional features that aren't present in CSS. Extra features like variables, nesting, and mixins help us write CSS code better and faster. We'll review these features, and you'll get lots of chances to practice with them, throughout this course.

Sass, which is open-source, was first developed in 2006. Useful documentation on Sass can be found at `https://sass-lang.com`.

1.1. The Problem Sass Solves

CSS is notoriously difficult to organize and maintain, and it gets exponentially more cumbersome as a site grows. Sass addresses this problem by adding common programming features (e.g., variables) and organization features (e.g., code nesting) to CSS. Sass *does not replace* CSS. Rather, its purpose is to *generate* CSS. It is what is known as a *CSS preprocessor*, meaning it extends CSS with extra features and then generates valid CSS code, which is sent to the browser.

And Sass is simple to use:

1. Write Sass code.

2. Compile Sass code into CSS code.
3. Include the generated CSS code with your HTML pages in the same way you normally do.

Let's take a look at a very simple example. Examine the HTML and CSS code below:

Demo 1.1: intro-why-sass/Demos/simple-sample-no-sass.html

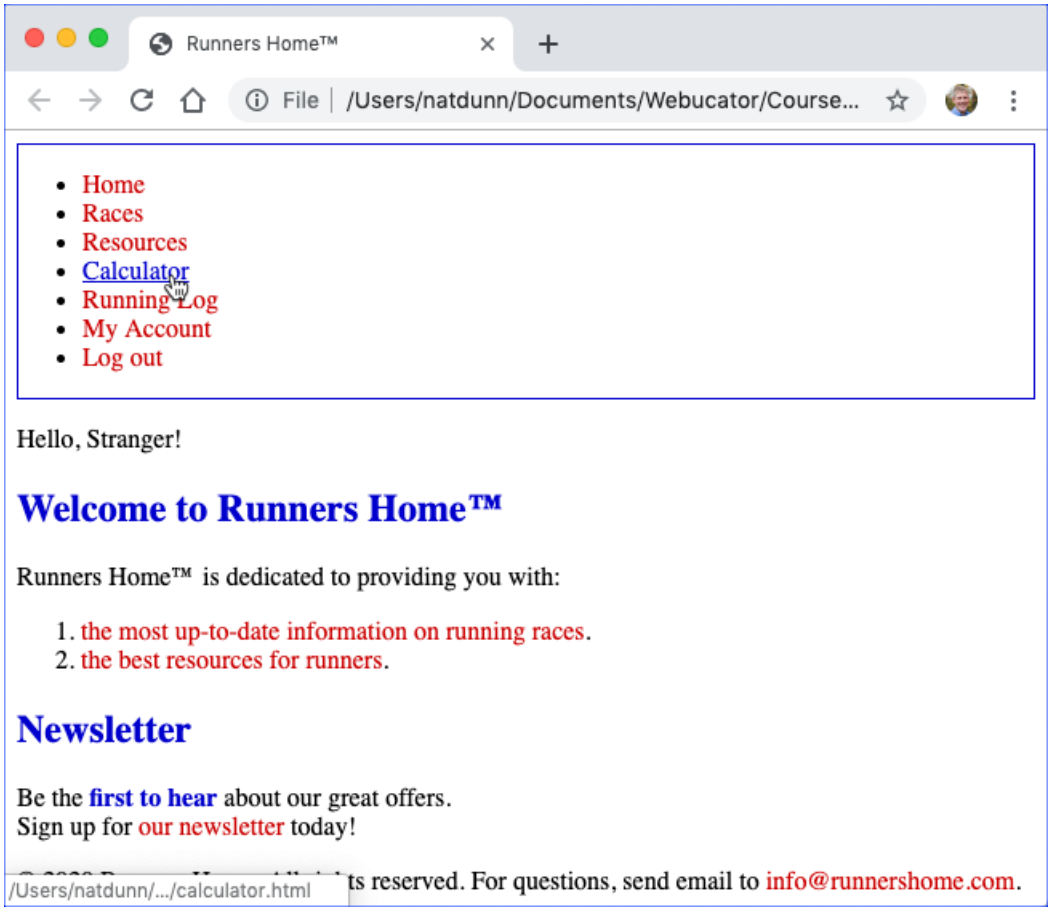
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,initial-scale=1">
  <title>Runners Home&trade;</title>
  <link href="simple-sample-no-sass.css" rel="stylesheet">
</head>
<body>
<header>
  <nav>
    <ul>
      <li><a href="index.html">Home</a></li>
      <li><a href="races.html">Races</a></li>
      <li><a href="resources.html">Resources</a></li>
      <li><a href="calculator.html">Calculator</a></li>
      <li><a href="running-log.html">Running Log</a></li>
      <li><a href="my-account.html">My Account</a></li>
      <li><a href="logout.html">Log out</a></li>
    </ul>
  </nav>
</header>
<main>
  <section id="welcome">
    <p>Hello, Stranger!</p>
    <h1>Welcome to Runners Home&trade;</h1>
  </section>
  <section id="purpose">
    <p>Runners Home&trade; is dedicated to providing you with:</p>
    <ol>
      <li><a href="races.html">the most up-to-date
        information on running races</a>.</li>
      <li><a href="resources.html">the best
        resources for runners</a>.</li>
    </ol>
  </section>
</main>
<aside>
  <h2>Newsletter</h2>
  <p>Be the <em>first to hear</em> about our great offers.<br>
  Sign up for <a href="newsletter.html">our newsletter</a> today!</p>
</aside>
<footer>
  &copy; 2022 Runners Home. All rights reserved.
```

```
45.     For questions, send email to
46.     <a href="mailto:info@runners-home.com">info@runners-home.com</a>.
47.   </footer>
48.   </body>
49.   </html>
```

Demo 1.2: intro-why-sass/Demos/simple-sample-no-sass.css

```
1.    h1, h2 {
2.      color: #1533b5;
3.    }
4.
5.    em {
6.      color: #1533b5;
7.      font-style: normal;
8.      font-weight: bold;
9.    }
10.
11.   header nav {
12.     border: 1px solid #1533b5;
13.   }
14.
15.   a {
16.     color: #c00;
17.     text-decoration: none;
18.   }
19.
20.   a:hover {
21.     color: #1533b5;
22.     text-decoration: underline;
23.   }
```


This is a simple page that will appear as follows in the browser:



Notice that the same “#1533b5” color is used in multiple places in the CSS. Imagine that you decided to change that color. You would have to do so several times over. You could use find and replace, but you might end up changing instances that you don’t want to.

Also, imagine a bigger website that uses this as a standard color on many different CSS pages. Updating that would be difficult.

Finally, imagine that multiple developers are working on the site and you want them all to be consistent with the colors used. They would need to look this up every time as “#1533b5” is not easy to remember.

Sass solves these problems by allowing you to create a variable. The code is shown below:

Demo 1.3: intro-why-sass/Demos/simple-sample.scss

```
1.  $highlight-color: #1533b5;
2.
3.  h1, h2 {
4.    color: $highlight-color;
5.  }
6.
7.  em {
8.    color: $highlight-color;
9.    font-style: normal;
10.   font-weight: bold;
11. }
12.
13. header nav {
14.   border: 1px solid $highlight-color;
15. }
16.
17. a {
18.   color: #c00;
19.   text-decoration: none;
20. }
21.
22. a:hover {
23.   color: $highlight-color;
24.   text-decoration: underline;
25. }
```


Code Explanation

As you can see, the code looks a lot like CSS, but includes a `$highlight-color` variable for the color.

The Sass code above will generate the following CSS document:

Demo 1.4: intro-why-sass/Demos/simple-sample.css

```
1.   h1, h2 {
2.     color: #1533b5; }
3.
4.   em {
5.     color: #1533b5;
6.     font-style: normal;
7.     font-weight: bold; }
8.
9.   header nav {
10.    border: 1px solid #1533b5; }
11.
12.  a {
13.    color: #c00;
14.    text-decoration: none; }
15.
16.  a:hover {
17.    color: #1533b5;
18.    text-decoration: underline; }
```

Code Explanation

As you can see, with the exception of some spacing differences, it is the same as the `simple-sample-no-sass.css` file shown earlier.

But now if we want to change “#1533b5” to something different throughout, we only need to make the change in one place.

Compiling CSS from Sass

Sass can be compiled to CSS manually at the command line or set up to compile automatically each time the Sass file changes. We can (and usually do) split our Sass code into multiple files, and stitch the generated output from those multiple files into a single CSS file.

It is important to understand that Sass code never gets sent to the browser. Rather, we use Sass as a tool to generate CSS. The compiled CSS code that we get from Sass is what we use in our web pages.

1.2. Benefits of Sass

In addition to giving you a complete working knowledge of how to use Sass to author CSS, another goal of this course is to highlight the benefits of using Sass instead of writing "plain old" CSS.

❖ 1.2.1. Code Organization

As websites - and thus CSS - grow in complexity, our CSS code gets harder to organize and harder to maintain. One big stylesheet works, of course, but thousands of lines of code - even if organized logically and well-documented with comments - can be tough to wade through a year from now when you need to make a quick change. Did that `.left` rule apply to the sidebar? Or to a paragraph?

Splitting CSS into multiple files comes with its own challenges (see "Performance" below) when not using a preprocessor. Sass makes it easy to organize our stylesheet code into logical chunks. We might group into files by purpose: a "normalize" or "reset" stylesheet, for example. Or group by section of the website: "about" or "contact". Or group by aspect of the site: "typography" or "graphics". We might even group our files by vendor - specific to, for example, the CSS applicable to a third-party jQuery or WordPress plugin.

Additionally, Sass allows us to nest rules, so that rules for contained elements - the `h1` and `p` tags that live inside a `div` of class `foo`, say - in code like this:

```
div.foo h1 {
  /* rules here */
}

div.foo p {
  /* rules here */
}
```

can be written with less code and more clearly. We'll cover nesting in detail later in the course.

❖ 1.2.2. Performance

Without Sass, we might address the challenge of organizing CSS by splitting our code into multiple files. We could either use CSS's `@import` statement to import CSS files from

other files, or include multiple stylesheet on our pages with multiple `<link>` tags. Both of these strategies incur a performance hit for our sites.

Sass allows us to maintain our stylesheet code across many files, organized however works logically for us or our team, and to generate from those multiple Sass files one single CSS file. Furthermore, we can tell Sass to compress the generated CSS file by removing spaces. This further reduces download time for our (now happier) users.

❖ 1.2.3. Valid CSS

The CSS that results from compiling Sass is always valid, standards-compliant CSS - it can't **not** be valid, as any mistake we make in writing Sass code gets caught by the Sass interpreter and we get an error. This is a great thing - Sass catches our mistakes.

❖ 1.2.4. Libraries

All good developers make use of others' work. Whether you work in the JavaScript library jQuery, PHP/MySQL-based WordPress, the Ruby on Rails developer framework, or some other technology, we all know that including plugins, libraries, and other types of code we get from other creators makes our work faster and better.

Sass is no different. We can include libraries developed by others, often through *mixins* (groups of CSS declarations packaged together), to jumpstart our work. We can create our own mixins, or download libraries like Bourbon[1] to benefit from the efforts of kind developers who share their work.

1.3. Sass Syntax

Originally, Sass used an indented syntax, with indentation to delimit style blocks instead of { and } and used newlines instead of semicolons to separate statements. This syntax looks like this:

```
.sidebar
    color: red
    font-size: 1.2rem
```

1. https://www.bourbon.io

The newer - and more popular - syntax, officially referred to as “SCSS”, uses curly braces like CSS. The code above in SCSS syntax would be:

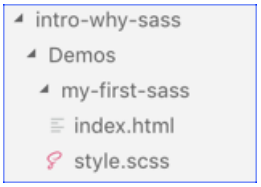
```
.sidebar {
    color: red;
    font-size: 1.2rem;
}
```

We will use the “SCSS” syntax throughout this course, and will refer to it as “Sass code” from here on.

1.4. How to Use Sass

Let’s take a look at how we can use Sass to compile CSS:

1. In your code editor, navigate to the `intro-why-sass/Demos/my-first-sass/` folder from your class files.
2. Open the file `index.html` in a code editor. This file is (almost) completed for you.
3. Create a file named `style.scss` in the same folder.
4. Your directory should now look like this:

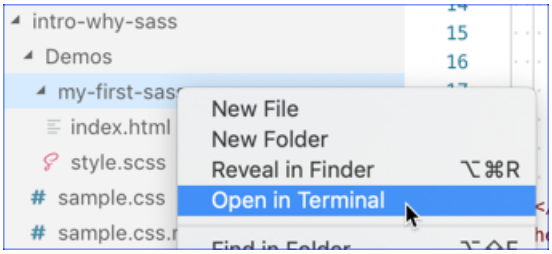


5. Note that the HTML file references `style.css` (not `style.scss`) to get its styles - but this CSS file does not yet exist. Not to worry - we’ll be creating it soon!

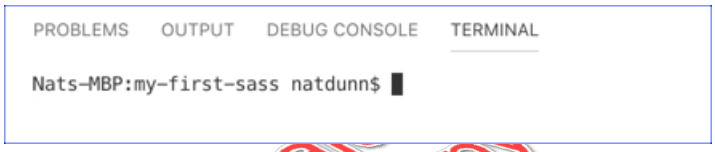
6. Enter the code below in the `style.scss` you created:

```
.page {
    margin:0 auto;
    width:80%;
}

header {
    background:#ff7800;
    padding:0.5rem 2rem;

    h1 {
        color:#fff;
        margin:0;
        padding:0;
    }
}

article {
    padding:0.5rem 2rem;
}

footer {
    background:#3a5aa0;

    p {
        color:#fff;
        font-size: 1.2rem;
        font-style: italic;
        padding: 0.5rem 2rem;
    }
}
```

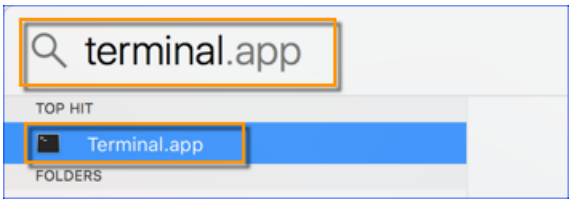

7. Make sure to save `style.scss`.
8. Next, open **Windows Command Prompt** or the **Mac Terminal**. From here on, we will refer to both as the *command prompt* or just the *prompt*.
    - In Visual Studio Code, you can open the command prompt by right-clicking the directory in **Explorer** and selecting **Open in Terminal**:



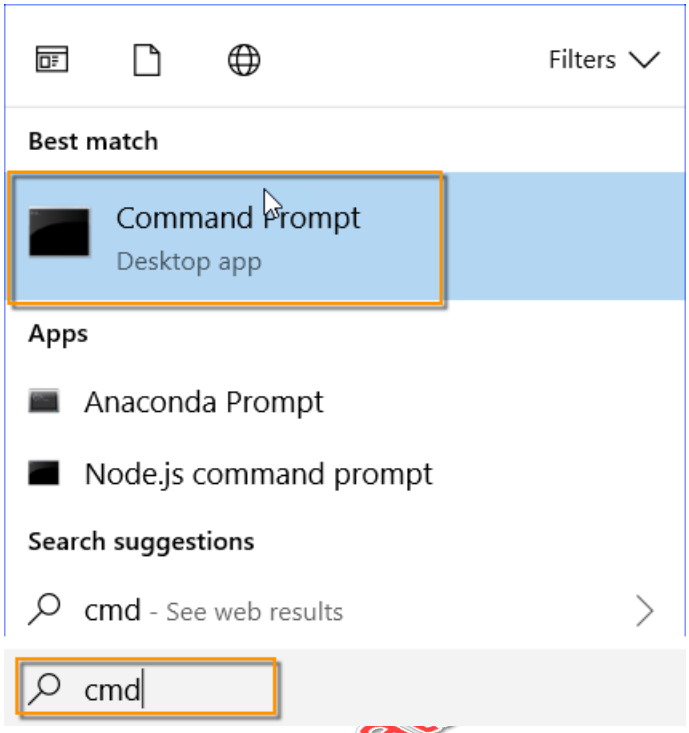
This will open the prompt at the directory:



- Or you can access the prompt directly from the operating system:
  - On a Mac, press **Cmd+Space**, type “Terminal”, and press **Enter**:



  - On Windows, type “cmd” in the **Windows Search Bar** and select **Command Prompt**:



9. Install Sass with npm:

```
npm install -g sass
```

Don't worry if you get some warnings. When it is done, you should see something like:

```
+ sass@1.25.0
added 15 packages from 18 contributors in 2.134s
```

10. Navigate to the `intro-why-sass/Demos/my-first-sass` directory using the `cd` (for **c**hange **d**irectory) command. The exact path will depend on where you installed the class files. It will look something like this:

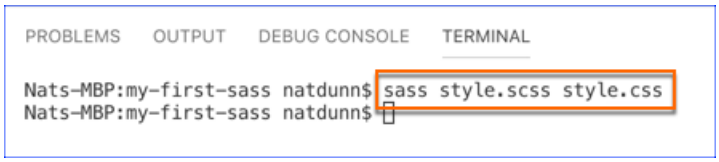
    - **Mac**:

```
cd /Users/myusername/Documents/Webucator/ClassFiles/intro-why-sass/De ↵↵
mos/my-first-sass
```

- **Windows**:

```
cd \Webucator\ClassFiles\intro-why-sass\Demos\my-first-sass
```

11. Once at the proper directory, type `sass style.scss style.css` and press **Enter**:



With this command, you have compiled the Sass code in `style.scss` into CSS code in the newly-created file `style.css` (along with some other files). You may need to refresh the files in Visual Studio Code to see the new files:

12. Lastly, add a `link` in `index.html` to use the newly-created CSS file to style this page: in the `head`, add `<link rel="stylesheet" href="style.css">`. Open the page in a browser to admire your work.

What is the .css.map file?

In addition to the new CSS file, you'll see a new `style.css.map` file created whenever you compile Sass code to CSS code. The map file is a "source map" - a file, in `.json` format, that contains information about where each line of compiled CSS comes from in Sass. The map file allows us to use the Chrome inspector to find where (that is, in which Sass file) a given CSS rule comes from. We'll show an example of this later.

1.5. Coming Attractions

As we work through the course, we'll cover the following features and aspects of Sass:

❖ 1.5.1. Organizing Sass

We will review strategies for effectively organizing our Sass code into meaningful units with partials: individual `.scss` files that we stitch together to produce a single `.css` file.

❖ 1.5.2. Nesting

Adopting the "Don't Repeat Yourself" ("DRY") philosophy, we will review in detail how Sass nesting works and why it is useful. We will cover name-spaced properties and Sass's `&` operator.

❖ 1.5.3. Variables

We will discuss how to use variables to hold values for colors, sizes, and other CSS quantities, how to organize variables effectively, and the usefulness of Sass variables for writing and maintaining CSS code.

Conclusion

In this lesson, you have learned:

- What preprocessors do.
- Some benefits of using Sass.
- How Sass generates CSS.
- Some useful Sass features.

Note that Sass, while the most popular, is not the only CSS preprocessor. Another commonly used one is Less[2].

2. http://lesscss.org

LESSON 2

Organizing and Including Sass Code

Topics Covered

☑ What partials are in Sass.

☑ How to import partials with the `@import` directive.

☑ Strategies for organizing Sass code.

Introduction

Sass allows us to organize our code into separate files, making our projects easier to build and maintain.

2.1. Sass Partials

Sass *partials* are simply files named with a leading underscore (e.g., "_partial.scss"). Using the Sass `@import` directive to import these files, we can split a large codebase across many files, organizing our work into groups that make sense to us and our team, and thus making our projects easier to maintain.

Traditional CSS - that is, non-Sass CSS - also has the `@import` directive. But in traditional CSS, unlike in Sass, grouping files together with `@import` has the result of forcing the client to make multiple HTTP requests, resulting in a slower page load. The Sass `@import`, on the other hand, stiches together multiple files into one compiled CSS file, thus resulting in a single HTTP request and making pages load faster for our users.

2.2. Partials/`@import` Example

Let's look at a simple example to learn more about partials and `@import`. Open the files in the directory `organizing-including-sass-code/Demos/partials-import/` from your class files to review the code. We'll look first at the main Sass file: `style.scss`.

Demo 2.1: organizing-including-sass-code/Demos/partials-import/style.scss

```
1.    @import 'normalize';
2.    @import 'typography';
```

Code Explanation

This file, `style.scss`, is our master Sass file. We use it to `@include` all partials. We import two partials:

1. _normalize.scss
2. _typography.scss

Note that when we use `@import`, we don't include the underscore nor the `.scss` extension: `@import 'normalize'` imports the partial "_normalize.scss".

Normalize Stylesheet

Our normalize stylesheet we take from the open-source Git repository at `http://necolas.github.io/normalize.css/`.

As the author of the code states, this CSS code "makes browsers render all elements more consistently and in line with modern standards."

In this example, we have only two partials. In a "real" production project, we might have dozens or even hundreds of partials, potentially grouped into various directories to further help with keeping our code organized for easier maintenance.

Let's look next at one of the partials files that is imported by `style.scss`: "_typography.scss."

Demo 2.2: organizing-including-sass-code/Demos/partials-import/_typography.scss

```
1.  body {
2.    font-family: 'Helvetica Neue',Helvetica,Arial,sans-serif;
3.    font-size: 13px;
4.  }
5.
6.  h1, h2, h3 {
7.    font-family: Georgia, serif;
8.  }
9.
10. h1 {
11.   font-size: 2rem;
12.   line-height: 2.3rem;
13. }
14.
15. h2 {
16.   font-size: 1.6rem;
17.   line-height: 1.8rem;
18. }
19.
20. h3 {
21.   font-size: 1.1rem;
22.   line-height: 1.3rem;
23.   text-transform: uppercase;
24. }
25.
26. p, ul, ol {
27.   font-size: 0.9rem;
28.   line-height: 1.1rem;
29. }
```


Code Explanation

This file is one of our imported partials. It holds CSS code for typography - font sizes, line heights, and font faces - for our simple project. The other partial, "_normalize.scss" (not shown here), is CSS that removes browser inconsistencies for HTML elements. We take this code from `https://github.com/guerrero/normalize.scss`.

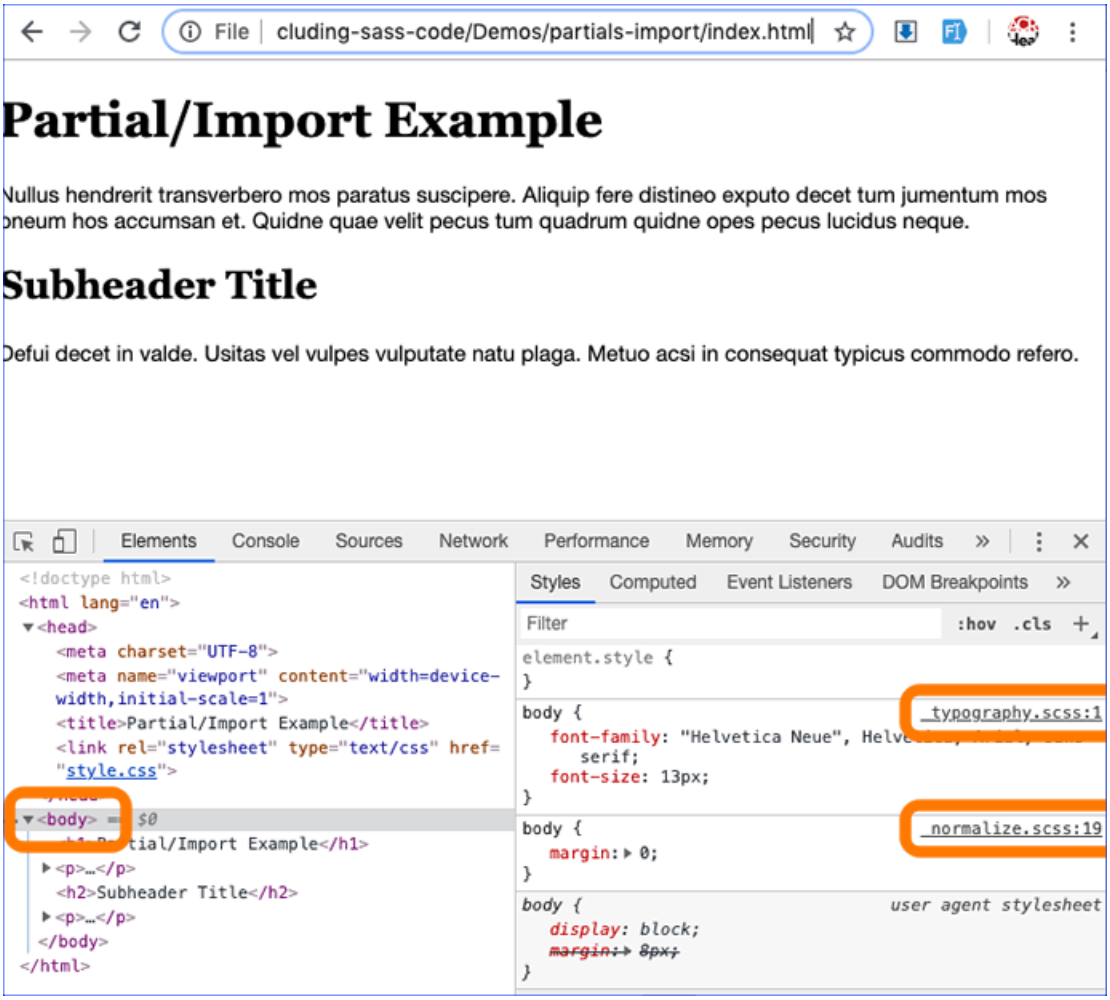
To create the CSS file (located at `organizing-including-sass-code/Demos/partials-import/style.css`) from this collection of Sass files, we:

1. Opened the command prompt.

2. Navigated to the `organizing-including-sass-code/Demos/partials-import/` directory.
3. Ran the command `sass style.scss style.css`.

Note that the compiled CSS code includes the code from both of our partials, imported in the same order ("_normalize.scss" first, then "_typography.scss") as we specified in `style.scss`.

Using Chrome's DevTools to inspect the elements in `index.html`, the HTML file in our simple project which references the compiled CSS file `style.css`, we can see the real value of the CSS map file that Sass generates when compiling Sass into CSS:



Note that Chrome's DevTools usefully gives us information about which partial holds the various rules for our elements. In the screenshot above, we see that the rules for

`font-family` and `font-size` for the `<body>` tag come from the partial "_typography.scss", and another rule (`margin: 0`) comes from "_normalize.scss". This information is extemely helpful when developing CSS with Sass, especially in a large project with many partials, as we can quickly find out where we need to make changes in our code.

Compiled CSS in Demos and Solutions

In the demo above, we included for you the compiled CSS code, `style.css`. For the rest of the course, we generally won't include the compiled CSS code - you can always run the `sass` command to produce the CSS code from the Sass code in our demos and solutions.

2.3. Sass Watch

So far, we have used the command line to compile Sass: a command like:

```
sass sample.scss sample.css
```


tells Sass to compile the Sass code in `sample.scss` and save the results as `sample.css`. This works fine, but it's a bit of a hassle to have to go back to the command line each time we want to recompile the CSS after we make a change to the Sass code.

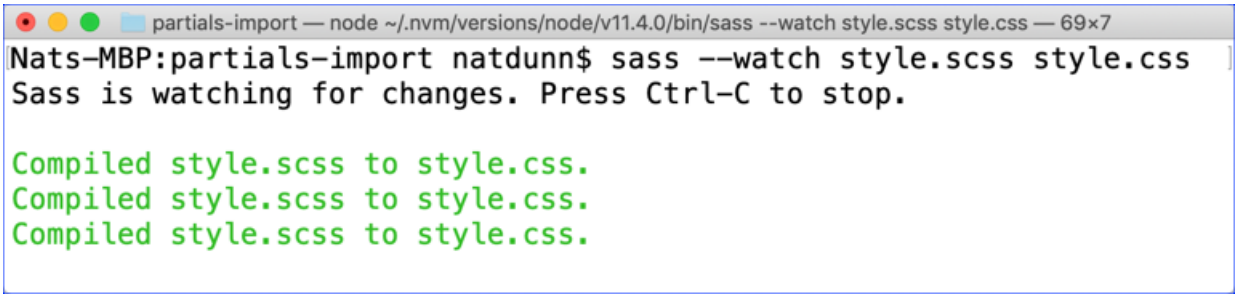
Luckily, Sass offers a "watch": we can tell Sass to watch a Sass file for any changes, and regenerate the CSS file each time there's an update. If we watch a Sass file that includes partials, then any time we edit any of the included partials, new CSS will be generated.

The command looks like this:

```
sass --watch sample.scss sample.css
```

The "--watch" flag tells Sass to keep an eye on `sample.scss` for any changes, and to generate a new copy of `sample.css` after each change.

The command line will display a new line each time the file changes:



Press **CTRL+C** to stop the watch.

Sass Output Style

When compiling Sass to CSS, either with a watch or as a single command, we have two choices for the output style:


```
sass --watch sample.scss sample.css --style=expanded
```

The “expanded” style is the default - if you don’t specify a style, you get “expanded”. This style writes each selector and declaration on its own line.

```
sass --watch sample.scss sample.css --style=compressed
```

The “compressed” style removes extra characters (like spaces) and writes the CSS stylesheet all on a single line. This output style is better for production, since the user downloads a smaller CSS file.

We will use Sass watches throughout the rest of this course.

Exercise 1: Using Partials

10 to 20 minutes

Review the files in `organizing-including-sass-code/Exercises/partials` in your editor. Notice that there is no `style.css` file.

1. Open `index.html` in a browser. Note that the content isn't styled - you'll be generating a CSS file for the styles.
2. Navigate to this directory (`organizing-including-sass-code/Exercises/partials`) from the command line and add a Sass watch:

   ```
   sass --watch style.scss style.css
   ```

   This will create the `style.css` file and update it each time one of the Sass files is changed.

3. From the same directory, open `style.scss` in a code editor. This file is done for you - note that this Sass file imports two partials, "_typography.scss" and "_color.scss".
4. Open "_typography.scss" and "_color.scss" for editing.
   A. In "_typography.scss", add some CSS rules for font for the HTML content in `index.html` - set the size of the `h1` tag, for instance, and the font family of the paragraph content.
   B. In "_color.scss", add some CSS rules for background color, text color, etc.
5. Refresh `index.html` to check the result of the styles you created.

Solution: organizing-including-sass-code/Solutions/partials/_typography.scss

```
1.  html {
2.   font-size:14px;
3.  }
4.
5.  h1 {
6.   font-family: Georgia, serif;
7.   font-size: 3rem;
8.  }
9.
10. p {
11.  font-family: Arial, sans-serif;
12.  font-size: 1.2rem;
13.  line-height: 1.6;
14. }
```

Code Explanation

We added some simple typography styles. Yours can be different. The important thing is that when you change this file, you notice that `style.css` gets updated.

Solution: organizing-including-sass-code/Solutions/partials/_color.scss

```
1.  body {
2.   background-color:#ddd;
3.  }
4.
5.  h1 {
6.   color:#ff7800;
7.  }
8.
9.  p {
10.  color:#3a5aa0;
11. }
```

Code Explanation

We added some simple color styles. Yours can be different. The important thing is that when you change this file, you notice that `style.css` gets updated.

2.4. Code Organization

There are many ways to structure partials in Sass:

1. Organize files in directories by:
    A. Vendors (CSS from external code libraries, such as jQuery plugins or CSS frameworks like Bootstrap).
    B. Sections of the site (home, products, etc.).
    C. Components of the site (buttons, navigation, etc.).
    D. Some other grouping strategy.
2. Organize files in a single directory using intelligently named partials.

Web developer and author Hugo Giraudel espouses the "7-1 pattern"[3] for architecting Sass code: seven different folders and one single file at the root level (`main.scss`) which imports all of the partials from the seven directories into one CSS file. He advocates separating code into the following folders:

1. `base/`: normalize file, typographic rules.
2. `layout/`: rules for laying out the site - header, footer, navigation.
3. `components/`: rules for smaller components and widgets - buttons, carousels, media.
4. `pages/`: page-specific rules for home page, contact page, etc.
5. `themes/`: theme-specific CSS rules.
6. `abstracts/`: Sass tools and helpers: variables, mixins, functions.
7. `vendors/`: CSS files from external libraries and frameworks - Bootstrap, jQuery sliders, etc.

3. https://sass-guidelin.es/#the-7-1-pattern

There isn't one perfect organizing principle for setting up a Sass project: the unique needs of each project and each team (or solo developer) differ greatly, and how you/your team choose to separate your code will depend on many factors. However, please do keep in mind the following general guidelines:

1. **Separate code into partials, optionally into directories:** However you choose to organize your code, the simple fact that you do separate Sass code into separate files means you are taking advantage of the power of Sass. Even separating into two files - a partial for a normalize stylesheet and a partial to hold the rest of the CSS code for your site - is better than using only a single file for all your Sass code.
2. **Pay attention to order:** A normalize or reset stylesheet should be `@import`ed early, since we use this to eliminate the inconsistencies between various browsers and versions of browers. Sass-specific stuff like variables and mixins that you write yourself (all of which we will get to later in the course) should be `@import`ed earlier rather than later, since later Sass code will make use of these constructs.
3. **Adopt conventions and stick to them:** Does your top-level, master Sass file that imports all of the partials import them directly, or does it make more sense for the top-level file to import other directory-index files which themselves import groups of partials? Is there some rule for how to structure import statements in one file - a full whitespace line, perhaps, to separate `@import` statements between separate directories? However you chose to organize your code, sticking to the same guidelines for all of your projects will make it easier to develop as a team and easier to quickly find the code you need to update a year from now.
4. **Use comments judiciously, but use comments:** Just like in CSS (or any language), comments in Sass give other developers - or ourselves, coming back to edit the code a year later - clues as to what this variable/section/partial/etc. is supposed to do. Use comments.

For our course, we'll use the following directory structure:

1. `modules/`: Sass code that doesn't output CSS - things like variables, functions, and mixins that we write ourselves. We haven't, of course, covered these Sass features yet in this course, but will soon in later lessons.
2. `partials/`: The heart of our CSS code - all of the rules for how our site is styled.
3. `vendor/`: CSS code from third-party libraries, plugins, and tools.

Within the three directories, we will organize into partials in different ways, depending on the specifics of the project we have at hand.

2.5. The Sass Blog

Let's look at how we might take an existing HTML/CSS project - built without Sass - and organize the CSS into Sass partials. The pages here are for a simple blog about Sass.

1. Open the files in the directory `organizing-including-sass-code/Demos/sass-blog/` in a code editor.
2. Edit the `<link>` tags in `index.html` and `register.html` so that they point to `styles-old.css`:

```
<link rel="stylesheet" href="styles-old.css">
```

3. This page is now styled by the CSS file `styles-old.css` - a CSS file that wasn't created with Sass.
4. Open `organizing-including-sass-code/Demos/sass-blog/index.html` in a browser. This is the blog, with pages for "home" and "register".

Next, we'll walk through how to use Sass to organize our CSS for these pages.

2.6. The Sass Blog: With Sass Partials

Let's pick apart `styles-old.css` and organize it into Sass partials, then compile the Sass code into the file `style.css` in the `css` directory. Our two pages, `index.html` and `register.html`, reference this CSS file for styling the pages.

First, change the link tags in each of the two pages (`index.html` and `register.html`) back to:

```
<link rel="stylesheet" href="css/style.css">
```

Note that we've created a directory structure inside the `scss` directory:

1. `modules/` will hold Sass code for variables, mixins, and the like. As we've not got any of these as yet, this directory will be empty.
2. `partials/` will hold Sass partials. We will organize our CSS code into files here.
3. `vendor/` will hold external Sass code: stuff we get from third-party plugins and libraries. As with `modules`, this directory will be empty for now.

Let's take a look at the original CSS code from `styles-old.css`, which contains the CSS we'll be moving into individual Sass partial files:

Demo 2.3: organizing-including-sass-code/Demos/sass-blog/styles-old.css

```
1.  body {
2.    background: #aaa;
3.    font-family: Helvetica,sans-serif;
4.    font-size: 14px;
5.  }
6.
7.  a {
8.    color: #a00;
9.    text-decoration: none;
10. }
11.
12. h1 {
13.   font-size: 2rem;
14.   margin: 0.5rem 0 2rem;
15. }
16.
17. h2 {
18.   font-size: 1.5rem;
19.   margin: 0.5rem 0;
20. }
21.
22. header,nav,section,footer {
23.   margin: 0 auto;
24. }
25.
26. header {
27.   background: #fff;
28.   padding: 30px 5%;
29.   width: 75%;
30. }
31.
32. header h2.logo {
33.   display: inline;
34.   font-size: 2.2rem;
35.   letter-spacing: 0.15rem;
36. }
37.
38. header form {
39.   float: right;
40.   margin-top: 5px;
41.   width: 20%;
42. }
43.
```

```
44.  header form input {
45.    font-size: 1.1rem;
46.    padding: 0.2rem;
47.  }
48.
49.  nav {
50.    background-color: #333;
51.    padding: 0;
52.    width: 85%;
53.  }
54.
55.  nav ul {
56.    list-style-type: none;
57.    margin: 0;
58.    overflow: hidden;
59.    padding: 0;
60.    padding-left: 5%;
61.  }
62.
63.  nav ul li {
64.    float: left;
65.  }
66.
67.  nav ul li a {
68.    color: white;
69.    display: block;
70.    padding: 14px 16px;
71.    text-align: center;
72.    text-decoration: none;
73.  }
74.
75.  nav ul li.active a, nav ul li a:hover {
76.    background-color: #000;
77.  }
78.
79.  section {
80.    background: #fff;
81.    padding: 30px 5%;
82.    width: 75%;
83.  }
84.
85.  section:after {
86.    clear: both;
87.    content: "";
88.    display: table;
```

```
  }

  section div {
    border-right: 1px solid #333;
    float: left;
    padding-right: 10%;
    width: 60%;
  }

  section article {
    border-bottom: 1px solid #333;
    margin-bottom: 2rem;
    padding-bottom: 1rem;
  }

  section aside {
    color: #444;
    float: right;
    font-size: 1.1rem;
    font-style: italic;
    line-height: 1.4rem;
    width: 25%;
  }

  footer {
    background: #333;
    color: #fff;
    padding: 10px 5%;
    width: 75%;
  }

  /* forms */

  fieldset {
    border: none;
    clear: both;
    margin: 0;
    padding: 0.5rem 0 0.5rem 0;
  }

  fieldset label {
    display: block;
    float: left;
    width: 20%;
  }
```

Code Explanation

We have styles for general elements (the `body`; `h1` and `h2` elements; link tags, etc.), for the `header`, for the main navigation, for the main body content, for the footer, and for forms. We'll set up a partial for each of these sections.

❖ 2.6.1. Directory Structure & Master Sass File

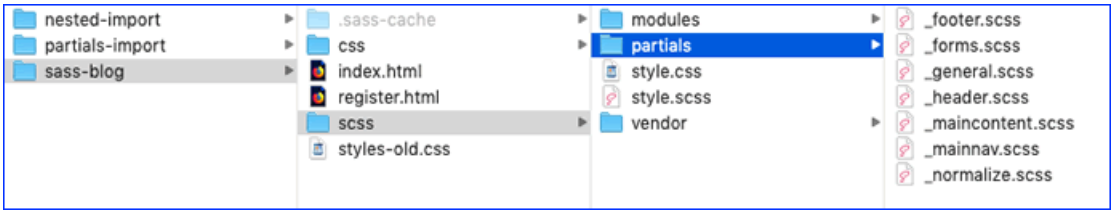
Demo 2.4: organizing-including-sass-code/Demos/sass-blog/scss/style.scss

```
1.   @import 'partials/normalize';
2.   @import 'partials/general';
3.   @import 'partials/header';
4.   @import 'partials/maincontent';
5.   @import 'partials/mainnav';
6.   @import 'partials/footer';
7.   @import 'partials/forms';
```


Code Explanation

The file `scss/style.scss` is our "master" Sass file: it contains `@import` statements for each partial we import. Note that we've also added a normalize partial: "_normalize.scss".

Our directory structure looks like this:



The `modules` and `vendor` folders are empty for now.

❖ 2.6.2. The `general` Partial

Demo 2.5: organizing-including-sass-code/Demos/sass-blog/scss/partials/_general.scss

```
1.  /*
2.    General Styles
3.  */
4.
5.  html {
6.    font-size: 14px;
7.  }
8.
9.  body {
10.   background: #aaa;
11.   font-family: Helvetica, sans-serif;
12. }
13.
14. a {
15.   color: #a00;
16.   text-decoration: none;
17. }
18.
19. p {
20.   margin-bottom: 1.3rem;
21. }
22.
23. h1 {
24.   font-size: 2rem;
25.   margin: 0.5rem 0 2rem;
26. }
27.
28. h2 {
29.   font-size: 1.5rem;
30.   margin: 0.5rem 0;
31. }
32.
33. header,nav,section,footer {
34.   margin: 0 auto;
35. }
```


Code Explanation

The partial file “scss/_general.scss” holds styles for the `body`, `h1`/`h2`, and `a` elements, and the `margin` rule for the structural (`header`, `nav`, `section`, and `footer`) elements.

❖ 2.6.3. The `header` Partial

Demo 2.6: organizing-including-sass-code/Demos/sass-blog/scss/partials/_header.scss

```
1.   /*
2.     Styles for the header on all pages
3.   */
4.
5.   header {
6.     background: #fff;
7.     padding: 30px 5%;
8.     width: 75%;
9.   }
10.
11.  header h2.logo {
12.    display: inline;
13.    font-size: 2.2rem;
14.    letter-spacing: 0.15rem;
15.  }
16.
17.  header form {
18.    float: right;
19.    margin-top: 5px;
20.    width: 20%;
21.  }
22.
23.  header form input {
24.    font-size: 1.1rem;
25.    padding: 0.2rem;
26.  }
```


Code Explanation

The partial file “scss/_header.scss” holds styles for the `header` of our blog pages, including our text-based logo and upper-right search form.

❖ 2.6.4. The `maincontent` Partial

Demo 2.7: organizing-including-sass-code/Demos/sass-blog/scss/partials/_maincontent.scss

```
1.  /*
2.    Styles for the body of the pages: main content, sidebar
3.  */
4.
5.  section {
6.    background: #fff;
7.    padding: 30px 5%;
8.    width: 75%;
9.  }
10.
11. section:after {
12.   clear: both;
13.   content: "";
14.   display: table;
15. }
16.
17. section div {
18.   border-right: 1px solid #333;
19.   float: left;
20.   padding-right: 10%;
21.   width: 60%;
22. }
23.
24. section article {
25.   border-bottom: 1px solid #333;
26.   margin-bottom: 2rem;
27.   padding-bottom: 1rem;
28. }
29.
30. section aside {
31.   color: #444;
32.   float: right;
33.   font-size: 1.1rem;
34.   font-style: italic;
35.   line-height: 1.4rem;
36.   width: 25%;
37. }
```

Code Explanation

The partial file “scss/_maincontent.scss” holds style rules for the wider main-content area of our pages and for the narrower right sidebar `aside` element.

❖ 2.6.5. The `mainnav` Partial

Demo 2.8: organizing-including-sass-code/Demos/sass-blog/scss/partials/_mainnav.scss

```
1.  /*
2.    Styles for the main nav element
3.  */
4.
5.  nav {
6.    background-color: #333;
7.    padding: 0;
8.    width: 85%;
9.  }
10.
11. nav ul {
12.   list-style-type: none;
13.   margin: 0;
14.   overflow: hidden;
15.   padding: 0;
16.   padding-left: 5%;
17. }
18.
19. nav ul li {
20.   float: left;
21. }
22.
23. nav ul li a {
24.   color: white;
25.   display: block;
26.   padding: 14px 16px;
27.   text-align: center;
28.   text-decoration: none;
29. }
30.
31. nav ul li.active a, nav ul li a:hover {
32.   background-color: #000;
33. }
```

Code Explanation

The partial file “scss/_mainnav.scss” lists style rules for the `nav` element that runs horizontally across the top of all pages.

❖ 2.6.6. The `footer` Partial

Demo 2.9: organizing-including-sass-code/Demos/sass-blog/scss/partials/_footer.scss

```
1.   /*
2.     Styles for the footer on all pages
3.   */
4.
5.   footer {
6.     background: #333;
7.     color: #fff;
8.     padding: 10px 5%;
9.     width: 75%;
10.  }
```


Code Explanation

The partial file “scss/_footer.scss” holds styles for the `footer` element at the bottom of all pages.

❖ 2.6.7. The `forms` Partial

Demo 2.10: organizing-including-sass-code/Demos/sass-blog/scss/partials/_forms.scss

```
1.   /*
2.     Styles for forms and form fields
3.   */
4.
5.   fieldset {
6.     border: none;
7.     clear: both;
8.     margin: 0;
9.     padding: 0.5rem 0 0.5rem 0;
10.  }
11.
12.  fieldset label {
13.    display: block;
14.    float: left;
15.    width: 20%;
16.  }
```


Code Explanation

The partial file “scss/_forms.scss” details style rules for general forms, like the one found on our registration page.

❖ 2.6.8. Up Next

Up next, you will make a few changes to the Sass blog, adding styles for buttons and incorporating a jQuery plugin for a visually-interesting datepicker.

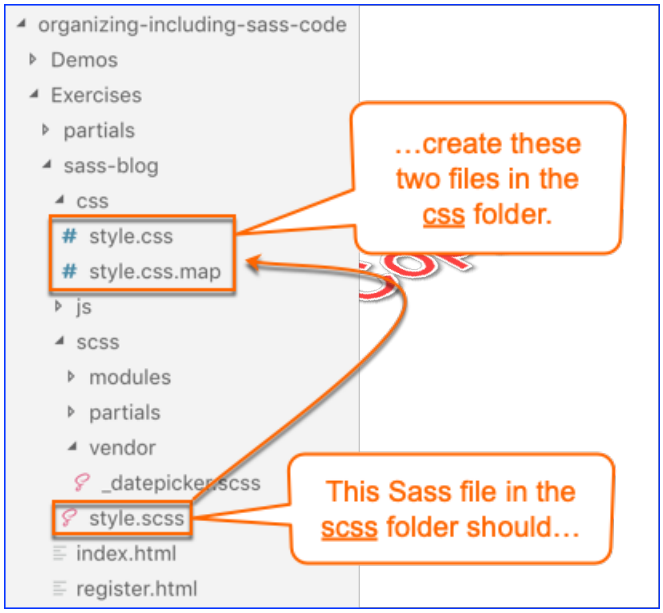
Exercise 2: Extending the Sass Blog

20 to 30 minutes

Open `organizing-including-sass-code/Exercises/sass-blog/index.html` in a browser.

1. We've copied the files from the previous demonstration for you. The Sass code is, as before, split into three directories (two of which are empty) with partials. The file `style.scss` imports all of the partials.

2. The `index.html` and `register.html` pages use `style.css` for styling. Create a Sass watch to compile `scss/style.scss` to `css/style.css`:



   Use an output style of "compressed":

   A. At the prompt, navigate to `organizing-including-sass-code/Exercises/sass-blog`.

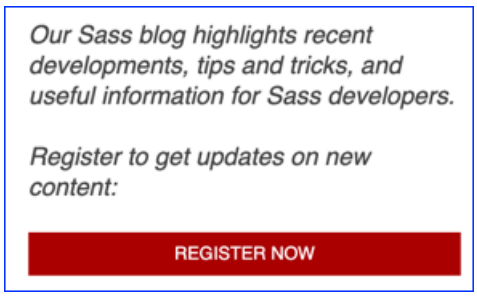
   B. Create the watch:

```
sass --watch scss/style.scss css/style.css --style=compressed
```

3. Add a link button reading “Register Now” to the `aside` element in both HTML pages. The button should link to the `register.html` page and have the ‘btn’ class:

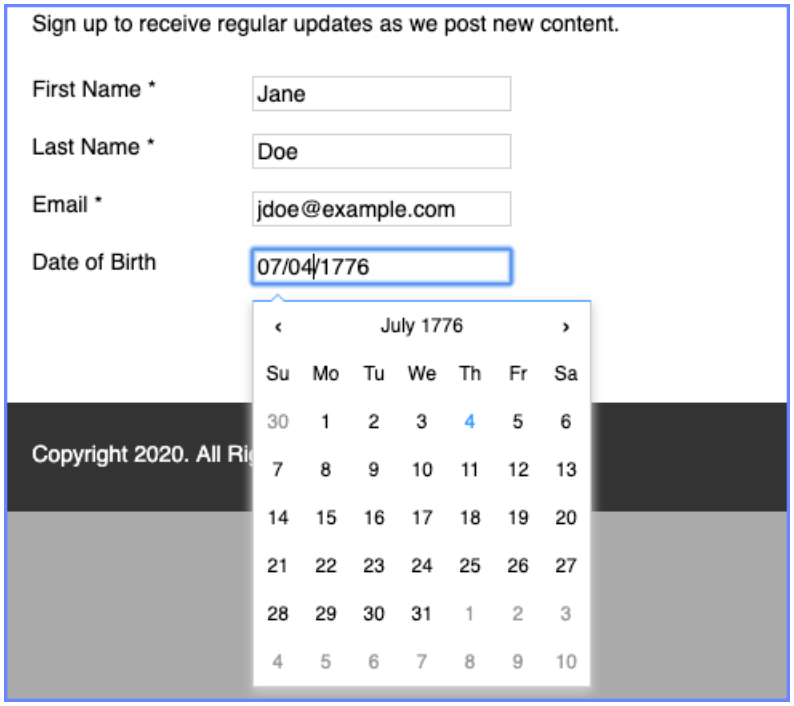
```
<a href="register.html" class="btn">Register Now</a>
```

4. Add styles to create button links for `a` tags with the `btn` class, so that the links that you just added in the last step look like buttons instead of text links. We styled our buttons like this:



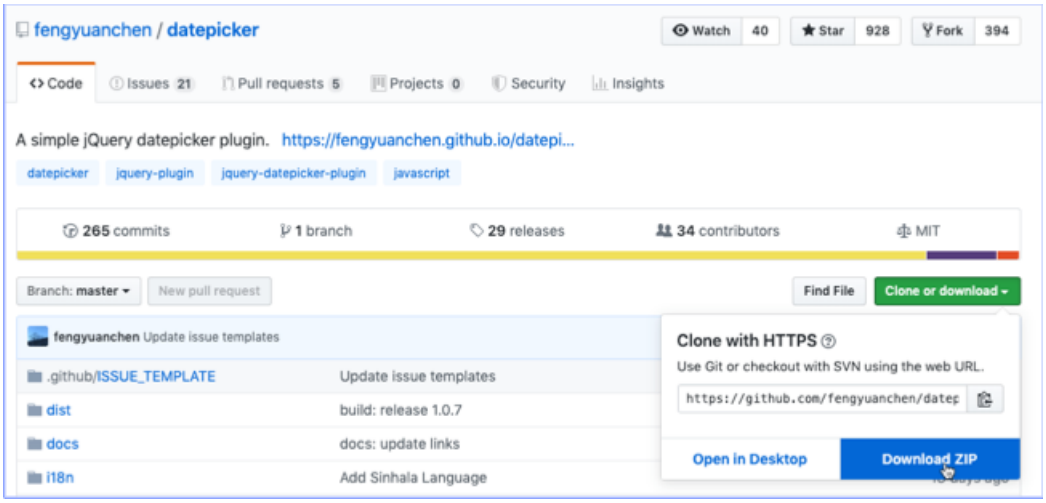
but feel free to design them however you like. Add your CSS in a new partial named “_buttons.scss” and be sure to import the partial in `style.scss`.

5. We had neglected to add a “submit” button at the bottom of our registration form. Add a submit button (an `input` or `button` element of `type submit`) at the bottom of the form. You may wish to next the button in a `fieldset` element. Add styles for this submit button to match the button-link styles you just created. It’s up to you in which partial (the new partial you created for “buttons”, or in the existing “forms” partial) this CSS code goes.
6. Next, you’ll add **datepicker**, a lightweight jQuery plugin which makes it easy to add an attractive date widget for picking dates. The widget will look like this:



We have already downloaded the necessary datepicker files. We have explained how we dowloaded and set up datepicker below. As we have done it already, you don't have to redo it, but you should review the steps and open the related files so you understand what is going on:

A. We downloaded files from **datepicker** (available at `https://github.com/fengyuanchen/datepicker`) and unpacked the zip archive:



B. We copied the minified CSS from `datepicker.min.css` (one of the files we just unzipped) and added it to our Sass blog as a new partial ("_datepicker.scss") in the `vendor` directory.

C. The plugin requires the use of jQuery, the JavaScript library.

JavaScript and jQuery

These next steps are not specific to Sass. They involve JavaScript and jQuery (a popular JavaScript library). If you have no experience with JavaScript, this may be confusing. Don't worry about it. We've already set everything up, and we're reviewing how we did it for the benefit of those students who have worked with JavaScript.

We included jQuery by adding the following code to each of our pages:

```
<script
  src="https://code.jquery.com/jquery-3.4.1.min.js"
  integrity="sha256-CSXorXvZcTkaix6Yvo6HppcZGetbYMGWSFlBw8HfCJo="
  crossorigin="anonymous"></script>
```


We added this code just before the closing `</body>` tag on all pages.

D. We created a new directory `js` (for "JavaScript") and copied the datepicker file `datepicker.min.js` into it.

E. Right below the `script` element we added for jQuery, we included the datepicker script on all pages:

```
<script src="js/datepicker.min.js"></script>
```

Note that datepicker gives us minified and non-minified versions of the JavaScript file and of the CSS file. We use the minified files, which are smaller.

F. Next, we need to call datepicker on desired elements: we want the user to experience the datepicker whenever an element of class `datefield` receives focus. We added that class to the "Date of Birth" field in `regis ter.html`. (Note that we've changed the `input type` for the "Date of Birth" field from `date` to `text`.) We added a file `app.js` to the new `js` directory

and added a `script` element, right below the `script` element for datepicker, on all pages:

```
<script src="js/app.js"></script>
```

G. We added the following code to `app.js`:

```
$(function() {
  $( ".datefield" ).datepicker();
});
```


With this jQuery code we create a listener for the "ready" state (when the DOM has loaded) and apply datepicker to any element of class `datefield`.

7. Now with datepicker set up, **all you have to do** is add an `@include` statement in `style.scss` to include the new "_datepicker.scss" partial. Go ahead and do that.
8. Be sure that your Sass watch is compiling your CSS changes. Test your work in a browser.

Solution: organizing-including-sass-code/Solutions/sass-blog/scss/style.scss

```
/*
 our CSS styles:
*/
@import 'partials/normalize';
@import 'partials/buttons';
@import 'partials/general';
@import 'partials/header';
@import 'partials/maincontent';
@import 'partials/mainnav';
@import 'partials/footer';
@import 'partials/forms';

/*
 third-party styles:
*/

@import 'vendor/datepicker';
```

Code Explanation

We've updated our master Sass file, `style.scss`, to import our new "_buttons.scss" partial (from the `partials` directory) and the new "_datepicker.scss" partial (from the `vendor` directory). The "_datepicker.scss" partial holds the CSS we got from the downloaded datepicker plugin - we made no changes to it.

Solution: organizing-including-sass-code/Solutions/sass-blog/scss/partials/_buttons.scss

```
a.btn {
  background-color: #a00;
  color: #fff;
  display: block;
  font-size: 0.80rem;
  font-style: normal;
  font-weight: normal;
  padding: 0.3rem 0.8rem;
  text-align: center;
  text-transform: uppercase;
}
```

Code Explanation

In the "_buttons.scss" partial, we added a rule for links (`a` elements) of class `btn`, giving the links a red background and white text color.

Solution: organizing-including-sass-code/Solutions/sass-blog/scss/partials/_forms.scss

```
     -------Lines 1 through 16 Omitted-------
17.
18.  input[type=submit] {
19.    background-color: #a00;
20.    border: none;
21.    color: #fff;
22.    display: block;
23.    font-size: 0.80rem;
24.    font-style: normal;
25.    font-weight: normal;
26.    margin-left: 20%;
27.    padding: 0.3rem 0.8rem;
28.    text-transform: uppercase;
29.  }
```


Code Explanation

We chose to add the styling for submit buttons to the "_forms.scss" partial (instead of the "_buttons.scss" partial), because it feels more like a "forms" thing than a "button" thing. The rule for the submit button is almost the same as for the link buttons. The only differences are:

1. There is a new declaration to remove the border.
2. There is a new declararation adding left margin.
3. There is no `text-align` declaration as button text is center aligned by default.

In a later lesson we'll look at ways to avoid having to maintain duplicative code like this.

2.7. More Styling for the Blog: Syntax Highlighting

Since our Sass blog will likely show some code snippets, it would be helpful to be able to highlight code like HTML, Sass, and CSS. In the next exercise, you will use Prism, a JavaScript/CSS library, to format code syntax.

Why Prism?

You may be thinking “I’m not going to create a blog about code. Why do I need to learn Prism?” The answer is that you don’t. This isn’t about learning Prism, it’s about learning to include pre-written third-party code in your code base.

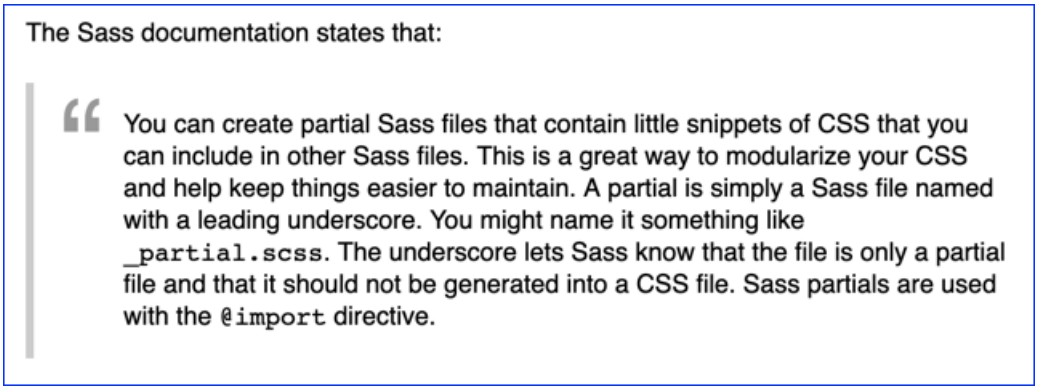
Exercise 3: Adding Syntax Coloring

15 to 25 minutes

Prism[4] "is a lightweight, extensible syntax highlighter." To get Prism to work on our Sass blog, we have included the CSS and JavaScript files from the Prism website. You will incorporate the Prism CSS as a Sass partial.

Open `organizing-including-sass-code/Exercises/sass-blog-syntax-coloring/index.html` in a browser.

1. Note that we've included the HTML, Sass, CSS, and JavaScript files from the previous Sass blog exercise solution so you don't have to redo any of this code.
2. We've added a new HTML file, `post-sass-import.html` - a detail (single) page for a blog post. We've updated `index.html` to include a post excerpt linking to this new page.
3. Create a Sass watch to compile Sass changes into the file `css/style.css`.
4. Add a new partial, "_single.scss", to hold styles for single-post pages. Note that the `body` element on this page has the class `single`, which we can use to target pages of this type. Add any styles you think might help improve this page. For example, you could increase the font size and line height for the main content.
5. Note that the quote on this page ("You can create partial Sass files that contain little snippets...") is marked up as a `blockquote`. Add a rule for styling `blockquote`s. You can design it however you like or try to make it look like this:



(We used `content: "\201C"` for `blockquote:before`) to add the quotation symbol.)

4. https://prismjs.com/

6. We have already downloaded Prism from `https://prismjs.com/download.html`, choosing the "Development version", the default theme, the "Sass" language, and the "Line Numbers" Plugin. We downloaded the JavaScript file to the `js` directory and added a `script` element for it at the bottom of all pages. We downloaded the CSS and added it as a new partial in the `scss/vendor` directory.
7. Add an `@import` statement in `style.scss` to import the new "_prism.scss" partial we added.
8. Prism works by finding `code` elements inside of `pre` elements: an element like this:

```
<pre><code class="language-css line-numbers"></code></pre>
```

   would format its contents with keywords colored according to the language specified (class `language-css`) with line numbers (class `line-numbers`) listed on the left. Other than adding the Prism CSS to our Sass, there are no other CSS changes to make to get it to work.
9. Your finished page should look something like this:

Solution:
organizing-including-sass-code/Solutions/sass-blog-syntax-coloring/scss/style.scss

```
/*
 our CSS styles:
*/
@import 'partials/normalize';
@import 'partials/buttons';
@import 'partials/general';
@import 'partials/header';
@import 'partials/maincontent';
@import 'partials/mainnav';
@import 'partials/footer';
@import 'partials/forms';
@import 'partials/single';

/*
 third-party styles:
*/

@import 'vendor/datepicker';
@import 'vendor/prism';
```


Code Explanation

Our master Sass file, `style.scss`, now looks as above: we've added partials `partials/_single.scss` (styles for single blog posts) and `vendor/_prism.scss` (CSS we got from Prism).

Solution:
organizing-including-sass-code/Solutions/sass-blog-syntax-coloring/scss/partials/_single.scss

```
.single section.main {
  font-size:1.0rem;
  line-height: 1.3rem;
}
```

Code Explanation

Our `partials/_single.scss` partial simply increases the font size and line height on single pages, targeting the `single` class on the `body` element.

Solution: organizing-including-sass-code/Solutions/sass-blog-syntax-coloring/scss/partials/_general.scss

```
     -------Lines 1 through 36 Omitted-------
37.  blockquote {
38.    background: #fff;
39.    border-left: 5px solid #ccc;
40.    display: block;
41.    margin: 0 0 2rem;
42.    padding: 1rem 2rem 1rem 3.5rem;
43.    position: relative;
44.  }
45.
46.  blockquote:before {
47.    color: #999;
48.    content: "\201C";
49.    font-size: 5rem;
50.    left: 1.0rem;
51.    line-height: 5rem;
52.    position: absolute;
53.    top: 0;
54.  }
```


Code Explanation

We added the rules for `blockquote` to `partials/_general.scss`, because we might use these styles elsewhere on the site - not just on single blog posts.

Conclusion

In this lesson, you have learned:

- What partials are in Sass.
- How to import partials with the `@import` directive.
- Strategies for organizing Sass code.

LESSON 3
Nesting

Topics Covered

☑ How to nest related rules in Sass.

☑ How to access the parent selector with `&`.

☑ How to nest properties.

Introduction

Nesting Sass code, grouping related rules together, can make code easier to write and easier to maintain.

3.1. Nesting

In CSS, we often make our rules more specific by stating that the styles for a given element only apply if that element is contained within another element. Consider the following CSS code:

```
.sidebar div {
  border-radius: 2rem;
}

.sidebar ul {
  list-style: none;
}
```

The rules above say that:

1. Any `div` contained within an element of class `sidebar` should have corners rounded at a radius of `2rem`.
2. Any unordered list contained within an element of class `sidebar` should get a `list-style` of `none`.

We don't, of course, want all `div`s to have rounded corners or all `ul`s to have `list-style: none` - just those found inside `.sidebar`.

While this specificity in CSS offers us great control over our style rules, it also can lead to code that is difficult to maintain. Sass, on the other hand, provides a mechanism - nesting - by which we can more easily write and update our code. Consider the following Sass code:

```
.sidebar {
    div {
        border-radius: 2rem
    }

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

The Sass code above will produce exactly the same CSS as we saw before, so users browsing our pages won't see any difference. But we've used Sass's nesting syntax to make our lives easier - the code is easier to read, easier to write, and easier to maintain.

3.2. Referencing the Parent Selector with `&`

Sometimes we want to reference the parent selector of a nested rule. In Sass we use the `&` character to explicitly reference the parent selector. Consider the following CSS rules for a link (`a`) element, in which we specify a different style when the link is hovered over:

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

a:hover {
  text-decoration: underline;
}
```

Users who mouseover our links - styled in the default without an underline - will see the underline appear. In Sass, we can rewrite this style using nesting:

```
a {
  text-decoration: none;

  &:hover {
    text-decoration: underline;
  }
}
```

The Sass code above produces the same CSS code as we saw before. But nesting helps us organize our code better, tying it more clearly to the link element to which it applies.

3.3. Nested Properties

We can also use Sass nesting to make our code more concise. CSS properties that are in namespaces (like `font-size`, `font-family`, and `font-weight`, or `text-align`, `text-decoration`, and `text-transform`) can be written more simply using nesting. Here's a CSS example for setting some font properties for an element:

```
div.foo {
  border: thin solid red;
  border-radius: 1em;
  font-family: Arial;
  font-size: 1.2em;
  font-weight: bold;
}
```

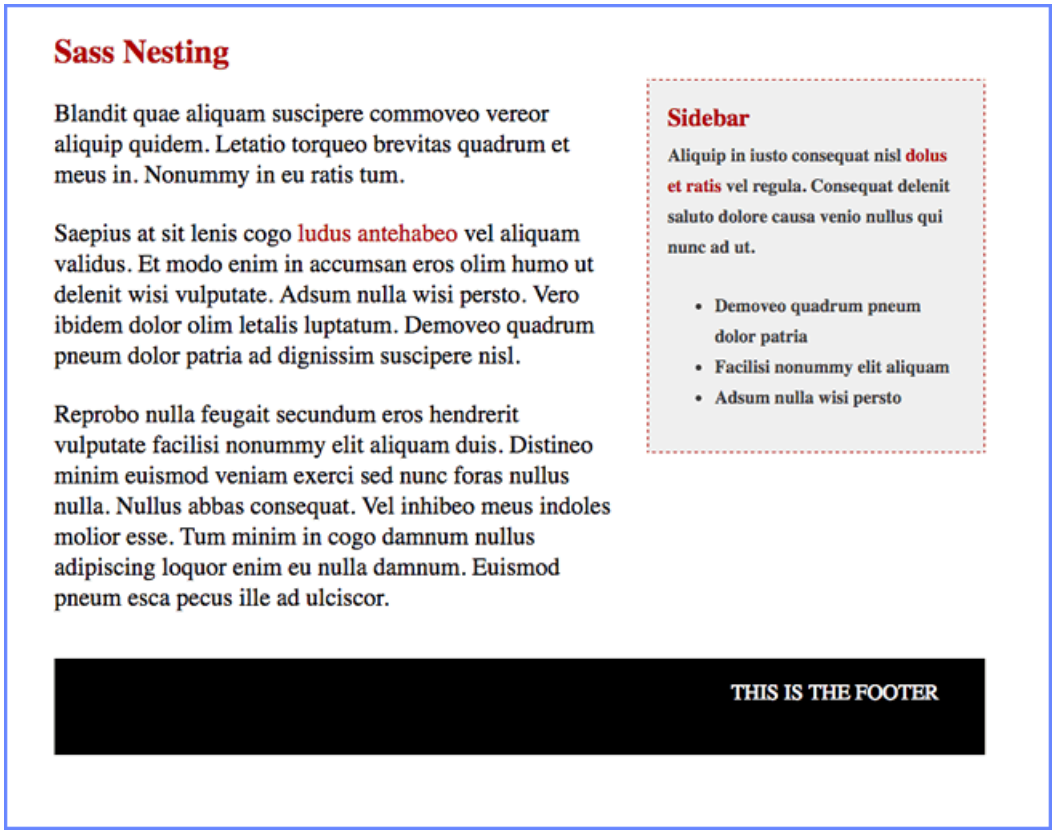
Any `div` of class `foo` is styled with a border with rounded corners and font of family of `Arial`, size `1.2em`, and weight of `bold`. We can rewrite this code in Sass using nesting. Note that we use a colon (`:`) after `font`:

```
div.foo {
  border: thin solid red;
  border-radius: 1em;

  font: {
    family: Arial;
    size: 1.2em;
    weight: bold;
  }
}
```

We find the nested font rules more succinct and clearer; whether you use nesting for something like `font-size`, `font-weight`, and `font-family` is, of course, up to you and your team.

3.4. Nesting Example

Let's look at an example in which we employ Sass nesting to produce CSS. Open the file `nesting/Demos/index.html` in a browser, and open `nesting/Demos/index.html` and `nesting/Demos/style.scss` in a code editor. Note that `index.html` references `style.css` for its style rules, and that `style.css` is generated by compiling `style.scss`. The page looks like this:


Sass Nesting
Blandit quae aliquam suscipere commoveo vereor aliquip quidem. Letatio torqueo brevitas quadrum et meus in. Nonummy in eu ratis tum.
Saepius at sit lenis cogo ludus antehabeo vel aliquam validus. Et modo enim in accumsan eros olim humo ut delenit wisi vulputate. Adsum nulla wisi persto. Vero ibidem dolor olim letalis luptatum. Demoveo quadrum pneum dolor patria ad dignissim suscipere nisl.
Reprobo nulla feugait secundum eros hendrerit vulputate facilisi nonummy elit aliquam duis. Distineo minim euismod veniam exerci sed nunc foras nullus nulla. Nullus abbas consequat. Vel inhibeo meus indoles molior esse. Tum minim in cogo damnum nullus adipiscing loquor enim eu nulla damnum. Euismod pneum esca pecus ille ad ulciscor.
Sidebar
Aliquip in iusto consequat nisl dolus et ratis vel regula. Consequat delenit saluto dolore causa venio nullus qui nunc ad ut.
Demoveo quadrum pneum dolor patria
Facilisi nonummy elit aliquam
Adsum nulla wisi persto
THIS IS THE FOOTER

Demo 3.1: nesting/Demos/nesting/index.html

```
1.  <!DOCTYPE HTML>
2.  <html lang="en">
3.  <head>
4.    <meta charset="UTF-8">
5.    <meta name="viewport" content="width=device-width,initial-scale=1">
6.    <title>Sass Nesting</title>
7.    <link rel="stylesheet" href="style.css">
8.  </head>
9.  <body class="page">
10.   <header>
11.     <h1>Sass Nesting</h1>
12.   </header>
13.   <article>
14.     <p>Blandit quae aliquam suscipere commoveo vereor aliquip quidem.
15.       Letatio torqueo brevitas quadrum et meus in.
16.       Nonummy in eu ratis tum.</p>
      -------Lines 17 through 27 Omitted-------
28.   </article>
29.   <aside class="sidebar">
30.     <h2>Sidebar</h2>
31.     <p>Aliquip in iusto consequat nisl <a href="#">dolus et ratis</a>
32.       vel regula. Consequat delenit saluto dolore causa venio nullus
33.       qui nunc ad ut.</p>
34.     <ul>
35.       <li>Demoveo quadrum pneum dolor patria</li>
36.       <li>Facilisi nonummy elit aliquam</li>
37.       <li>Adsum nulla wisi persto</li>
38.     </ul>
39.   </aside>
40.   <footer>
41.     <p>This is the footer</p>
42.   </footer>
43. </body>
44. </html>
```


Code Explanation

Our markup is fairly simple:

1. The `body` tag gets a class `page`.
2. The `article` element floats `left`.
3. The `aside.sidebar` element floats `right`.

4. The `footer` element has a black background with white text.

Demo 3.2: nesting/Demos/nesting/style.scss

```
1.  html {
2.    font-size: 14px;
3.    margin: 0;
4.    padding: 0;
5.  }
6.
7.  body {
8.    margin: 0;
9.    padding: 0;
10. }
11.
12. .page {
13.   margin: 0 auto;
14.   padding: 2% 5%;
15.   width: 60%;
16.
17.   a {
18.     color: #a00;
19.     text-decoration: none;
20.     &:hover {
21.       border-bottom: 1px solid #a00;
22.     }
23.   }
24.
25.   h1, h2 {
26.     color: #a00;
27.     margin: 0 0 0.5rem 0;
28.     padding: 0;
29.   }
30.
31.   p {
32.     font-size: 1.5rem;
33.     line-height: 1.9rem;
34.     margin: 0 0 1.7rem 0;
35.   }
36.
37.   ul li {
38.     font-size: 1.5rem;
39.     line-height: 1.9rem;
40.   }
41.
42.   article {
43.     float: left;
44.     padding: 2% 3% 2% 0;
```

```
45.       width: 60%;
46.     }
47.
48.     aside.sidebar {
49.       background-color: #efefef;
50.       float: right;
51.       padding: 1.4rem 2%;
52.       width: 32%;
53.
54.       p, ul li {
55.         color: #333;
56.         font: {
57.           size: 1.1rem;
58.           weight: bold;
59.         }
60.       }
61.
62.       border: {
63.         color: #a00;
64.         style: dashed;
65.         width: 1px;
66.       }
67.     }
68.
69.     footer {
70.       background-color: #000;
71.       clear: both;
72.       margin-top: 3rem;
73.       padding: 2% 5%;
74.
75.       p {
76.         color: #fff;
77.         font-size: 1.3rem;
78.         text: {
79.           align: right;
80.           transform: uppercase;
81.         }
82.       }
83.     }
84.   }
```


Code Explanation

Our Sass code makes use of the following nesting structure:

- `html`
- `body`
    - `a`
        - `&:hover`
    - `h1, h2`
    - `p`
    - `ul li`
    - `article`
    - `aside.sidebar`
        - `border:`
        - `p, ul li`
    - `footer`
        - `p`
            - `text:`

You will use Sass nesting in the next exercise.

Exercise 4: Using Nesting

15 to 20 minutes

1. Open the following two files in your editor:

   A. `nesting/Exercises/nesting-original.css`. This file contains a set of empty CSS rules, which you will rewrite as Sass. The code is shown below the instructions. We use comments in the rules as placeholders.

   B. `nesting/Exercises/nesting.scss`. This is where you will rewrite the CSS rules as Sass. Use comments as placeholders as we did in the CSS file or just make up some declarations. If you do not include any content in the rule, then the rule will not be included in the compiled CSS file.

2. Create a Sass watch to compile `nesting.scss` to `nesting.css`.

3. Use Sass nesting to write the CSS found in `nesting-original.css` with as few references to `.sidebar`, `p`, and `table` as possible.

4. Compare `nesting.css` (the generated CSS from your compiled Sass code) to `nesting-original.css` to ensure that they are the same. Don't worry if the spacing is a little different.

Exercise Code 4.1: nesting/Exercises/nesting-original.css

```
1.    .sidebar { /* rule */ }
2.
3.    .sidebar p { /* rule */ }
4.
5.    .sidebar p a { /* rule */ }
6.
7.    .sidebar p a:hover { /* rule */ }
8.
9.    .sidebar p strong { /* rule */ }
10.
11.   .sidebar table { /* rule */ }
12.
13.   .sidebar table tr { /* rule */ }
14.
15.   .sidebar table tr:nth-child(3n) { /* rule */ }
16.
17.   .sidebar table td,
18.   .sidebar table th { /* rule */ }
19.
20.   .sidebar table td { /* rule */ }
21.
22.   .sidebar table td p { /* rule */ }
23.
24.   .sidebar table th { /* rule */ }
```

Solution: nesting/Solutions/nesting.scss

```
1.    .sidebar {
2.      p {
3.        a { /* rule */
4.          &:hover { /* rule */ }
5.        }
6.        strong { /* rule */ }
7.      }
8.      table { /* rule */
9.        tr { /* rule */
10.         &:nth-child(3n) { /* rule */ }
11.       }
12.       td, th { /* rule */ }
13.       td { /* rule */
14.         p { /* rule */ }
15.       }
16.       th { /* rule */ }
17.     }
18.   }
```


Code Explanation

We nest all of the elements inside of `.sidebar`, and nest the various contained elements for `p` and `table`. Note that we use the `&` character to reference the parent selector for `a:hover` and for `tr:nth-child(3n)`.

3.5. Nesting and Media Queries

We use CSS media queries to adapt our pages to conditions such as screen width or resolution. Often, we will use media queries to style pages differently for different devices, providing visitors to our sites layouts tailored to mobile, tablet, or desktop devices' screens.

Without Sass, we might write a media query like this:

```
article {
  float: left;
  width: 60%;
}

@media only screen and (max-width: 600px) {
  article {
    float: none;
    width: 100%;
  }
}
```

With Sass, we can write the media query as a nested item under the element to which it applies, like this:


```
article {
  float: left;
  width: 60%;

  @media only screen and (max-width: 600px) {
    float: none;
    width: 100%;
  }
}
```

Sass nesting allows us to tie the media query more closely to the element itself, which many developers find more readable and easier to maintain. Of course, we can still write media queries the “CSS way”, even if using Sass.

The next example shows how we might add media queries to our earlier demo:

Demo 3.3: nesting/Demos/media-queries/index.html

```
1.   <!DOCTYPE HTML>
2.   <html lang="en">
3.   <head>
4.     <meta charset="UTF-8">
5.     <meta name="viewport" content="width=device-width,initial-scale=1">
6.     <title>Sass Nesting</title>
7.     <link rel="stylesheet" href="style.css">
8.   </head>
9.   <body class="page">
10.    <header>
11.      <h1>Sass Nesting</h1>
12.    </header>
13.    <article>
14.      <p>Blandit quae aliquam suscipere commoveo vereor aliquip quidem.
15.        Letatio torqueo brevitas quadrum et meus in.
16.        Nonummy in eu ratis tum.</p>
     -------Lines 17 through 27 Omitted-------
28.    </article>
29.    <aside class="sidebar">
30.      <h2>Sidebar</h2>
31.      <p>Aliquip in iusto consequat nisl <a href="#">dolus et ratis</a>
32.        vel regula. Consequat delenit saluto dolore causa venio nullus
33.        qui nunc ad ut.</p>
34.      <ul>
35.        <li>Demoveo quadrum pneum dolor patria</li>
36.        <li>Facilisi nonummy elit aliquam</li>
37.        <li>Adsum nulla wisi persto</li>
38.      </ul>
39.    </aside>
40.    <footer>
41.      <p>This is the footer</p>
42.    </footer>
43.  </body>
44.  </html>
```


Code Explanation

The HTML markup here is exactly the same as our earlier example.

Demo 3.4: nesting/Demos/media-queries/style.scss

```
     -------Lines 1 through 11 Omitted-------
12.  .page {
13.    margin: 0 auto;
14.    padding: 2% 5%;
15.    width: 60%;
16.
17.    @media only screen and (max-width: 992px) {
18.      width: 90%;
19.    }
20.
21.    a {
     -------Lines 22 through 44 Omitted-------
45.
46.
47.    article {
48.      float: left;
49.      padding: 2% 3% 2% 0;
50.      width: 60%;
51.
52.      @media only screen and (max-width: 992px) {
53.        float: none;
54.        padding: 0;
55.        width: auto;
56.      }
57.    }
58.
59.    aside.sidebar {
60.      background-color: #efefef;
61.      float: right;
62.      padding: 1.4rem 2%;
63.      width: 32%;
64.
65.      p, ul li {
66.        color: #333;
67.        font: {
68.          size: 1.1rem;
69.          weight: bold;
70.        }
71.      }
72.
73.      border: {
74.        color: #a00;
75.        style: dashed;
```

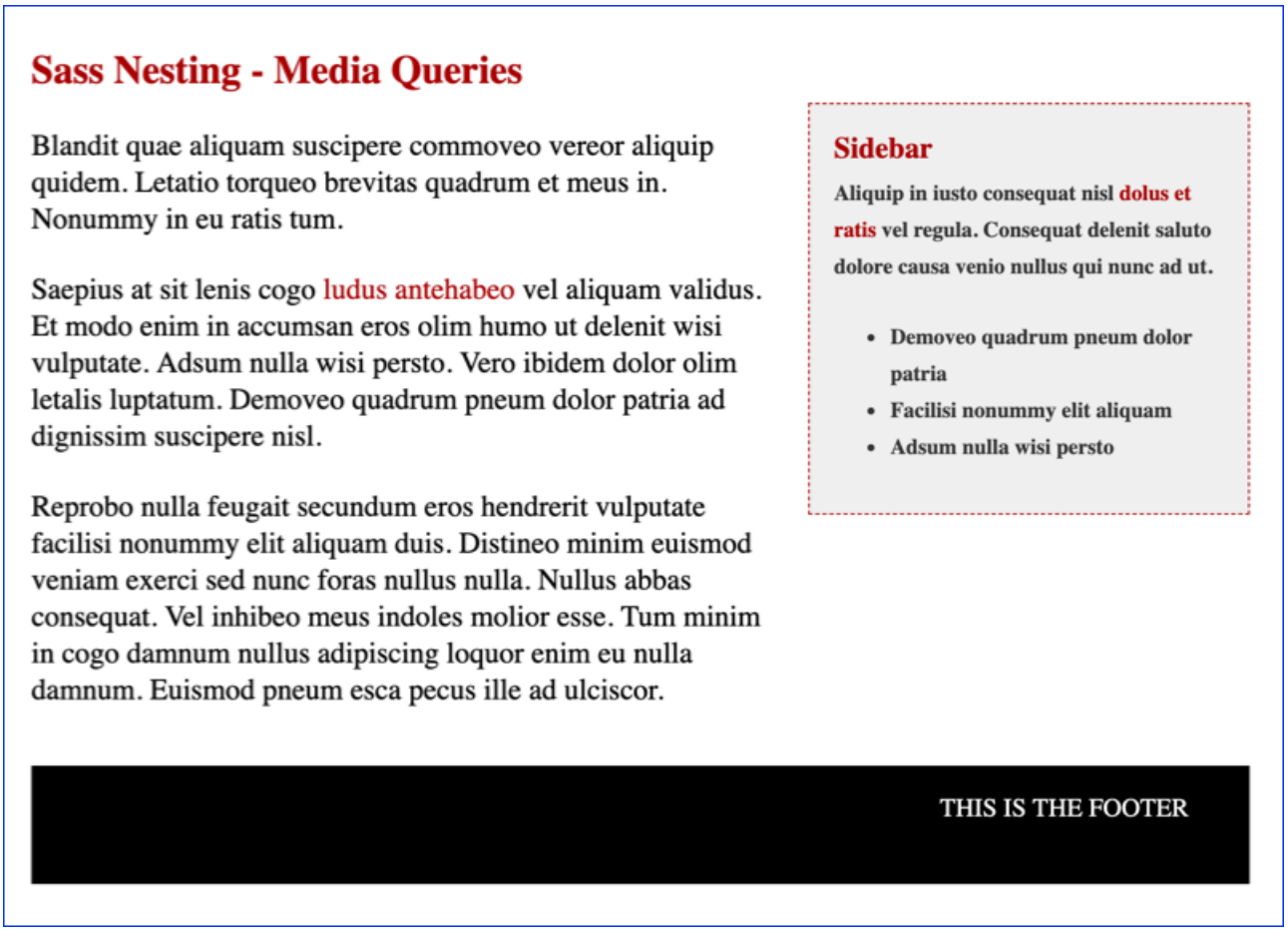
```
76.         width: 1px;
77.       }
78.
79.       @media only screen and (max-width: 992px) {
80.         float: none;
81.         padding: 1.4rem 5%;
82.         width: auto;
83.       }
84.     }
85.
86.     footer {
      -------Lines 87 through 100 Omitted----------
101.  }
```


Code Explanation

We use the same Sass code to style the page as before, but we've added media queries. We add a media query for the `.page` wrapper element to make it wider when the browser is narrow. And we add a media query for the `article` and `aside` elements, to un-float them when the browser is narrow.

The screenshots below show the difference:

Wide View (>992px):



Narrow View (<=992px):

3.6. How Much to Nest

Nesting in Sass doesn't allow us to do anything we couldn't do without nesting. The context of your particular project and your personal preferences will dictate how much you use nesting in your Sass code. Consider, for example, the `border-` rules we specified for `aside.sidebar` in the example above:

```
border: {
  color: #a00;
  style: dashed;
  width: thin;
}
```

Is this code clearer and easier to write/maintain than the CSS shorthand we could have used without Sass?

```
border: thin dashed #a00;
```

Some front-end developers find Sass nesting of great use; others use it sparingly. As a Sass developer, you should know it exists; how often you use it is up to you.

In the next exercise, you will apply nesting to our Sass blog.

Exercise 5: Using Nesting with the Sass Blog

15 to 30 minutes

We've copied the latest version of the Sass blog to `nesting/Exercises/sass-blog/`.

1. As usual, create a Sass watch to compile `scss/style.scss` to `css/style.css`
2. Open each partial (except "_normalize.scss") in the `partials` folder and, where appropriate, rewrite rules to use nesting.

Solution: nesting/Solutions/sass-blog/scss/partials/_mainnav.scss

```
1.  /*
2.      Styles for the main nav element
3.  */
4.
5.  nav {
6.    background-color: #333;
7.    padding: 0;
8.    width: 85%;
9.
10.   ul {
11.     list-style-type: none;
12.     margin: 0;
13.     overflow: hidden;
14.     padding: 0;
15.     padding-left:5%;
16.
17.     li {
18.       float: left;
19.
20.       a {
21.         color: white;
22.         display: block;
23.         padding: 14px 16px;
24.         text-align: center;
25.         text-decoration: none;
26.       }
27.     }
28.
29.     li.active a, li a:hover {
30.       background-color: #000;
31.     }
32.   }
33. }
```


Code Explanation

In the "_mainnav.scss" partial, shown above, we've nested several layers deep to streamline our code.

We also used nesting in the following partials:

1. _forms.css
2. _general.css

3. _header.css
4. _mainnav.css

3.7. Nested `@import`

Sass allows for nesting `@import` statements within CSS rules. Like the importing of partials we've already seen, a nested `@import` statement will include the contents of the imported file, but will be nested in the same place as the original import. Let's take a look at a quick demonstration of this to see how it works: open the files in the directory `organizing-including-sass-code/Demos/nested-import/` from your class files to explore the code.

Demo 3.5: nesting/Demos/nested-import/highlighted.scss

```
1.   .highlighted {
2.     color: red;
3.     font-style: italic;
4.     font-weight: bold;
5.   }
```


Code Explanation

The Sass file `highlighted.scss` holds a rule for elements of class `highlighted`, styling them as red, italic, and bold.

Demo 3.6: nesting/Demos/nested-import/style.scss

```
1.   section.maincontent {
2.     @import 'highlighted';
3.   }
```

Code Explanation

The Sass file `style.scss` has a rule for `section` elements of class `maincontent`; it imports `highlighted.scss` inside the rule for the `section.maincontent` selector.

Demo 3.7: nesting/Demos/nested-import/style.css

```
1.    section.maincontent .highlighted {
2.      color: red;
3.      font-style: italic;
4.      font-weight: bold;
5.    }
6.
7.    /*# sourceMappingURL=style.css.map */
```

Code Explanation

The result of running the command `sass style.scss style.css` is to create/update CSS in the file `style.css`. Note that the selector here is `section.maincontent .highlighted`.

Conclusion

In this lesson, you have learned how to use nesting in Sass to group similar rules together.

LESSON 4
Sass Variables

Topics Covered

☑ How to use variables in Sass.

☑ Why variables make our work easier.

☑ The syntax, scope, and data types for variables.

Introduction

Variables make Sass code easier to write and maintain.

4.1. Variables

Variables store values for reuse later by other code, making it easier to keep your CSS properties consistent and to maintain and update your code. This makes it easy to change dozens, even hundreds of font sizes, colors, or other values across thousands of lines of code.

Variables also make our Sass code better by the very fact that they have names. Consider the following CSS code:

```
h2 {
  color: #ff7800;
}
```

To most of us, it's not immediately obvious what color the `h2` elements will be. Now consider the following in Sass:

```
$color-branding-orange: #ff7800;

h2 {
  color: $color-branding-orange;
}
```

This produces the same result: orange `h2` elements, but the code itself conveys *meaning*: that color isn't just some random shade of orange, but rather a hue that is part of the company's visual brand.

4.2. Variable Naming Guidelines

Variable identifiers (names) are preceded by a dollar sign (`$`) character:


```
$font-size-default: 13px;
```

The name of the variable is what comes after that dollar sign. When naming variables, we recommend you follow these guidelines:

1. Don't use a number as the first character. It's not allowed.
2. Only use letters, numbers, underscores, and hyphens in your variable names.
3. Separate words in your variable names with hyphens. For example:

```
// Bad
$colorbluedark: #0000C8;

// Good
$color-blue-dark: #0000C8;
```

4. Make your variable names meaningful. For example:

```
// Bad
$color-1: #0000C8;
$color-2: #C62D42;

// Good
#color-blue-dark: #0000C8;
#color-red-brick: #C62D42;
```

5. Move from *generic* words to *specific* words. For example:

```
// Bad
$dark-blue-color: #0000C8;
$light-blue-color: #5BE5EC;

$primary-branding-color: #0000C8;
$secondary-branding-color: #3a5ba0;

// Good
$color-blue-dark: #0000C8;
$color-blue-light: #5BE5EC;

$color-branding-primary: #ff7800;
$color-branding-secondary: #0000C8;
```


6. Consider naming variables by their function. For example:

```
$color-border: #00f;
$color-border-light: #71D9E2;
$color-border-dark: #770F05;
```

For more tips on variable naming, see `https://webdesign.tutsplus.com/articles/quick-tip-name-your-sass-variables-modularly--webdesign-13364`.

4.3. Variable Scope

The scope of Sass variables depends on where the variables are declared:

1. **Global Variables**. Variables defined outside of blocks (denoted by curly brackets) are global, meaning that they can be accessed from anywhere, even in other stylesheets.
2. **Local Variables**. Variables defined within a block are local to that block.

Consider the following example:

Demo 4.1: sass-variables/Demos/scope.scss

```
1.   $webucator-orange: #ff7800; // global variable
2.
3.   h1 {
4.     color: $webucator-orange;
5.   }
6.
7.   h2 {
8.     $webucator-blue: #3a5ba0; // local variable
9.
10.    color: $webucator-blue;
11.  }
12.
13.  .title {
14.    background-color: $webucator-orange;
15.
16.    // The following is not allowed as $webucator-blue
17.    // is local to the h2 block.
18.    // color: $webucator-blue;
19.  }
```


This will produce the following CSS:

Demo 4.2: sass-variables/Demos/scope.css

```
1.   h1 {
2.     color: #ff7800;
3.   }
4.
5.   h2 {
6.     color: #3a5ba0;
7.   }
8.
9.   .title {
10.    background-color: #ff7800;
11.  }
```

❖ 4.3.1. Shadowing

When you declare a local variable with the same name as a global variable, it is called *shadowing*. Even though they share the same name, the two variables have no relationship to each other. When called, the local variable takes precedence. Consider the following example:

Demo 4.3: sass-variables/Demos/scope-shadowing.scss

```
1.   $color-orange: #f60; // global variable
2.
3.   h1 {
4.     color: $color-orange; // will use global variable
5.   }
6.
7.   h2 {
8.     // The following does not change the value of
9.     // the global variable.
10.    $color-orange: #ff7800; // local variable
11.
12.    color: $color-orange; // will use local variable
13.  }
14.
15.  h3 {
16.    color: $color-orange; // will use global variable
17.  }
```


This will produce the following CSS:

Demo 4.4: sass-variables/Demos/scope-shadowing.css

```
1.   h1 {
2.     color: #f60;
3.   }
4.
5.   h2 {
6.     color: #ff7800;
7.   }
8.
9.   h3 {
10.    color: #f60;
11.  }
```

Code Explanation

Notice that the `color` for the `h2` element is `#ff7800`, which was the value of the local variable.

❖ 4.3.2. Overwriting Global Variables within Blocks

You can use the `!global` flag to overwrite a global variable from within a block. Consider the following:

Demo 4.5: sass-variables/Demos/overwrite-global.scss

```
1.   $color-orange: #f60; // global variable
2.
3.   h1 {
4.     color: $color-orange; // will be #f60;
5.   }
6.
7.   h2 {
8.     $color-orange: #ff7800 !global; // overwrites variable
9.
10.    color: $color-orange; // will be #ff7800;
11.  }
12.
13.  h3 {
14.    color: $color-orange; // will be #ff7800;
15.  }
```


This will produce the following CSS:

Demo 4.6: sass-variables/Demos/overwrite-global.css

```
1.   h1 {
2.     color: #f60;
3.   }
4.
5.   h2 {
6.     color: #ff7800;
7.   }
8.
9.   h3 {
10.    color: #ff7800;
11.  }
```

Code Explanation

Notice that the `color` for the `h2` and `h3` elements are both `#ff7800`, because the value of the global `$color-orange` variable has been overwritten.

You should, however, not declare new global variables from within blocks. This used to be allowed, but has been deprecated. Consider the following:

Demo 4.7: sass-variables/Demos/declare-global-in-block.scss

```
1.   h2 {
2.     $orange: #ff7800 !global;
3.
4.     color: $orange;
5.   }
```

Code Explanation

In Dart Sass 2.0.0, this will still work, but you will get a warning when you compile the `.scss` file:

```
sass write-global-in-block.scss write-global-in-block.css
Deprecation Warning: As of Dart Sass 2.0.0, !global assignments won't be able to
declare new variables. Consider adding `$color-orange: null` at the top level.

  |
2 |   $color-orange: #ff7800 !global;
  |   ^^^^^^^^^^^^^^^^^^^^^^^^
  |
    write-global-in-block.scss 2:3  root stylesheet
```


If you get this warning, you should fix the problem by either using a local variable or declaring the variable outside of a block.

In the next exercise, you will use variables to replace color values in a simple HTML page whose CSS is built with Sass.

Exercise 6: Using Variables in Sass

15 to 25 minutes

Linear Gradient

This exercise makes use of the `linear-gradient` CSS function to create a gradient background. . To learn more about this function, see `https://developer.mozilla.org/en-US/docs/Web/CSS/linear-gradient`.

1. Beginning in `sass-variables/Exercises`, create a Sass watch to compile `style.scss` to `style.css` (both in the same directory).
2. Open `sass-variables/Exercises/index.html` in a browser. The page is fairly simple, consisting of a `header` with an `h1` title, an `article` with some greeking (fake text)[5] and a pull quote (a highlighted excerpt of text)[6] marked up as an `aside`, and a `footer`.
3. Notice the colors used on the page.
4. At the top of `style.scss`, define variables for the colors. Give the variables meaningful names.
5. Replace the color values in `style.scss` with the variables you defined.
6. Test your work in a browser.
7. Make some changes to the colors to test different color combinations. See how easy it is!

5. https://en.wikipedia.org/wiki/Greeking#In_computing
6. https://en.wikipedia.org/wiki/Pull_quote

Solution: sass-variables/Solutions/style.scss

```
1.  $color-off-white: #f8f7fc;
2.  $color-branding-primary: #0a0;
3.  $color-branding-secondary: #004f4f;
4.  $color-background-light: #aaa;
5.  $color-highlight: #ffa;
6.
7.  html {
8.    font-size: 16px;
9.    height: 100%;
10. }
11.
12. body {
13.   background: linear-gradient(to bottom,
14.                               $color-off-white,
15.                               $color-highlight);
16.   background-attachment: fixed;
17.   height: 100%;
18.
19.   &.page {
20.     margin: 1rem auto;
21.     padding: 2% 5%;
22.     width: 60%;
23.   }
24. }
25.
26. header > h1 {
27.   border-bottom: {
28.     color: $color-branding-primary;
29.     style: solid;
30.     width: 4px;
31.   }
32.   color: $color-branding-primary;
33.   font-size: 2em;
34. }
35.
36. a {
37.   border-bottom: {
38.     color: $color-branding-primary;
39.     style: dotted;
40.     width: 1px;
41.   }
42.   color: $color-branding-primary;
43.   text-decoration: none;
44. }
```

```
45.
46.  article {
47.    padding-right: 5%;
48.
49.    h1 {
50.      color: $color-branding-secondary;
51.      font-size: 1.6em;
52.    }
53.
54.    h2 {
55.      color: $color-branding-secondary;
56.      font-size: 1.2em;
57.    }
58.  }
59.
60.  aside.pull-quote {
61.    background-color: $color-background-light;
62.    border: {
63.      color: $color-branding-primary;
64.      style: dotted;
65.      width: 2px;
66.    }
67.    float: right;
68.    margin: 0 0 20px 20px;
69.    padding: 1% 5%;
70.    width: 35%;
71.
72.    h3 {
73.      border-bottom: {
74.        color: $color-highlight;
75.        style: dashed;
76.        width: 1px;
77.      }
78.      color: $color-branding-primary;
79.      margin-bottom: .5em;
80.      padding-bottom: .2em;
81.    }
82.
83.    p {
84.      margin-top: 0;
85.
86.      q {
87.        color: $color-off-white;
88.        font-style: italic;
89.      }
```

```
90.     }
91.   }
92.
93.   footer {
94.     background-color: $color-branding-secondary;
95.     color: $color-off-white;
96.     margin-top: 3rem;
97.     padding: 2% 5%;
98.
99.     a {
100.      border-bottom: {
101.        color: $color-highlight;
102.        style: dotted;
103.        width: 1px;
104.      }
105.      color: $color-highlight;
106.      text-decoration: none;
107.
108.      &:hover {
109.        border-bottom-color: $color-off-white;
110.        color: $color-off-white;
111.      }
112.    }
113.  }
```


Code Explanation

Notice how we began each color variable name with “$color” and then move to more specific attributes of the color for the rest of the name. One big advantage of doing this is that IDEs, such as Visual Studio Code, that use intellisense, drop down the variable names that begin with “c” as soon as you type “$c”:

You may have used different variable names, but hopefully you followed the variable naming guidelines covered earlier in this lesson.

4.4. When to Use Variables

Thinking through your variables is key to writing good Sass code. Perhaps the biggest benefit of variables is that they provide friendly names for hard-to-remember values. Here are some good use cases:

1. **Color Names** - for friendly names and branding colors.
2. **Font sizes** - if you want more control over this than you get from `xx-large`, `x-large`, `large`, etc, you could create your own `$font-size-xxl`, `$font-size-xl`, `$font-size-l` variables.
3. **Standardizing borders**:


```
$border-heavy: 4px solid $color-branding-primary;
$border-medium: 2px solid $color-branding-secondary;
$border-light: 1px dotted #000;
```

   Notice how you can use one variable within the value of another. Just be sure to have previously declared the variable used in the value.

4. **Screen dimensions for responsive breakpoints**:

```
$screen-sm: 767px;
$screen-md: 992px;
$screen-lg: 1200px;
```

You could then use these variables when creating responsive designs:

```
article {
    float: none;
    width: 100%;

  @media only screen and (min-width: $screen-md) {
    float: left;
    width: 60%;
  }
}
```

Exercise 7: More Variables

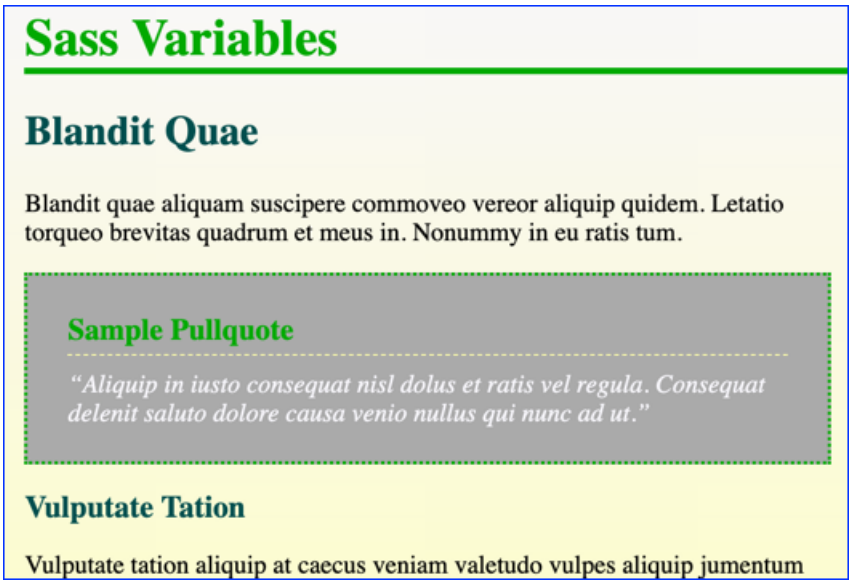
15 to 25 minutes

In this exercise, you will add and use variables for different border styles and for holding screen widths.

1. Beginning in `sass-variables/Exercises`, if you don't still have a Sass watch compiling `style.scss` to `style.css`, create one.
2. Open `style.scss` in your editor.
3. Create and use variables to hold the different types of borders used in the code.
4. Create variables to hold different screen widths.
5. Modify the `aside.pull-quote` rule to use:

```
float: none;
margin: auto;
width: 90%;
```


This will make the pull quote `article` appear on a block by itself, like this:



6. Using one of the screen width variables you created, add an `@media` rule nested within the `aside.pull-quote` rule with these declarations when the screen width is greater than 992px:

```
float: right;
margin: 0 0 20px 20px;
width: 35%;
```

Solution: sass-variables/Solutions/style-2.scss

```
1.  $color-off-white: #f8f7fc;
2.  $color-branding-primary: #0a0;
3.  $color-branding-secondary: #004f4f;
4.  $color-background-light: #aaa;
5.  $color-highlight: #ffa;
6.
7.  $border-heavy: 4px solid $color-branding-primary;
8.  $border-medium: 2px dotted $color-branding-primary;
9.  $border-light: 1px dotted $color-branding-primary;
10. $border-highlight: 1px dashed $color-highlight;
11.
12. $screen-sm: 767px;
13. $screen-md: 992px;
14. $screen-lg: 1200px;
15.
16. html {
17.   font-size: 16px;
18.   height: 100%;
19. }
20.
21. body {
22.   background: linear-gradient(to bottom,
23.                              $color-off-white,
24.                              $color-highlight);
25.   background-attachment: fixed;
26.   height: 100%;
27.
28.   &.page {
29.     margin: 1rem auto;
30.     padding: 2% 5%;
31.     width: 60%;
32.   }
33. }
34.
35. header > h1 {
36.   border-bottom: $border-heavy;
37.   color: $color-branding-primary;
38.   font-size: 2em;
39. }
40.
41. a {
42.   border-bottom: $border-light;
43.   color: $color-branding-primary;
44.   text-decoration: none;
```

```
45.  }
46.
47.  article {
48.    padding-right: 5%;
49.
50.    h1 {
51.      color: $color-branding-secondary;
52.      font-size: 1.6em;
53.    }
54.
55.    h2 {
56.      color: $color-branding-secondary;
57.      font-size: 1.2em;
58.    }
59.  }
60.
61.  aside.pull-quote {
62.    background-color: $color-background-light;
63.    border: $border-medium;
64.    float: none;
65.    margin: auto;
66.    padding: 1% 5%;
67.    width: 90%;
68.
69.    @media only screen and (min-width: $screen-md) {
70.      float: right;
71.      margin: 0 0 20px 20px;
72.      width: 35%;
73.    }
74.
75.    h3 {
76.      border-bottom: $border-highlight;
77.      color: $color-branding-primary;
78.      margin-bottom: .5em;
79.      padding-bottom: .2em;
80.    }
81.
82.    p {
83.      margin-top: 0;
84.
85.      q {
86.        color: $color-off-white;
87.        font-style: italic;
88.      }
89.    }
```

```
90.   }
91.
92.   footer {
93.     background-color: $color-branding-secondary;
94.     color: $color-off-white;
95.     margin-top: 3rem;
96.     padding: 2% 5%;
97.
98.     a {
99.       border-bottom:  $border-highlight;
100.      color: $color-highlight;
101.      text-decoration: none;
102.
103.      &:hover {
104.        border-bottom-color: $color-off-white;
105.        color: $color-off-white;
106.      }
107.    }
108.  }
```


Code Explanation

Your Sass file will still be called `style.scss`.

Note that we adopt a “mobile-first” approach here, first defining the default styles for the pull quote for smaller screens and then changing those styles for screens once they are wider than the specified width.

Exercise 8: Adding Variables to the Blog

25 to 40 minutes

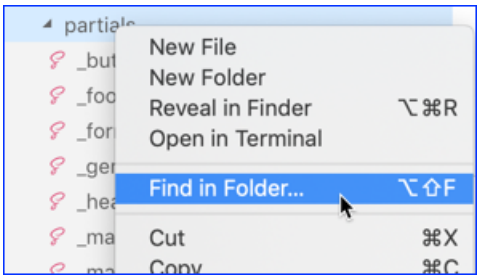
In this exercise, you will add and use variables in the Sass Blog. In the solution, we used color and screen-width variables and added code to make the blog responive. You can choose to do the same or try something different. No matter what variables you choose to create, you should put them in a new "_variables.scss" partial in the `modules` folder.

Solution: sass-variables/Solutions/sass-blog-variables/scss/modules/_variables.scss

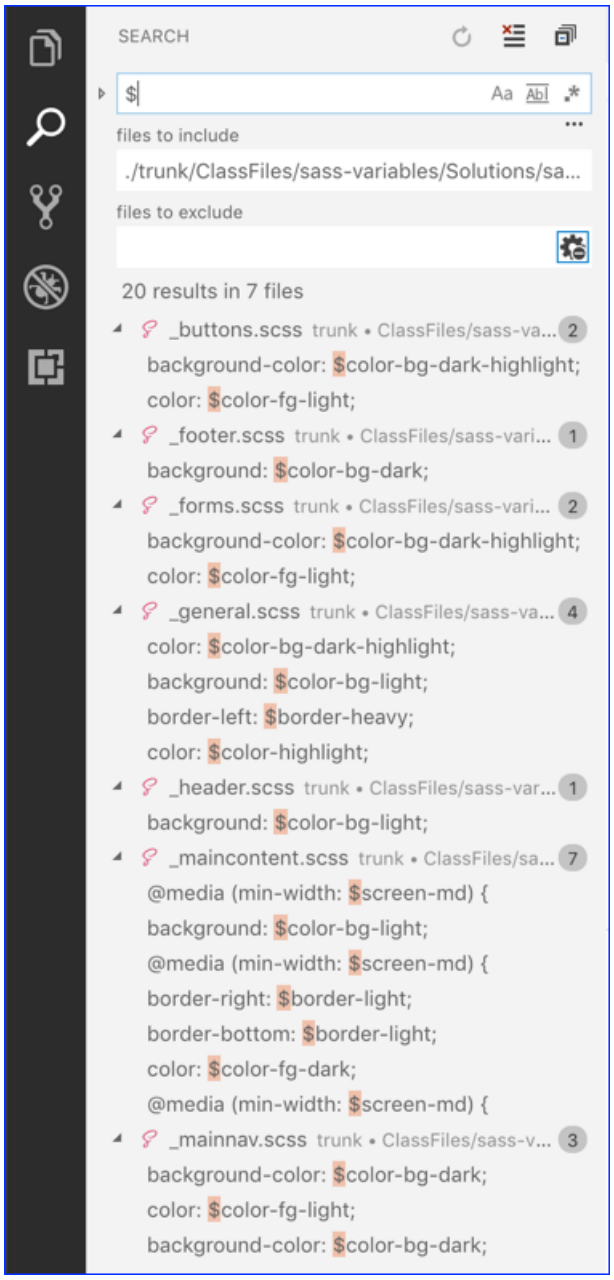
```
1.   /* responsive breakpoints */
2.   $screen-sm: 767px;
3.   $screen-md: 992px;
4.   $screen-lg: 1200px;
5.
6.   /* colors: */
7.   $color-bg-dark: #333;
8.   $color-bg-dark-highlight: #a00;
9.   $color-bg-light: #fff;
10.  $color-fg-dark: #444;
11.  $color-fg-light: #ffa;
12.  $color-highlight: #aaa;
13.
14.  $border-heavy: 5px solid $color-highlight;
15.  $border-light: 1px solid $color-bg-dark;
```


Code Explanation

Browse the `partials` folder to see where we used our variables. **A quick tip for making this type of search easier**: In Visual Studio Code, right-click a folder and select **Find in Folder...** to search all the files in a folder:



Search for the `$` sign to find where we used variables:


SEARCH
$
Aa
files to include
./trunk/ClassFiles/sass-variables/Solutions/sa...
files to exclude
20 results in 7 files
_buttons.scss trunk • ClassFiles/sass-va... 2
background-color: $color-bg-dark-highlight;
color: $color-fg-light;
_footer.scss trunk • ClassFiles/sass-vari... 1
background: $color-bg-dark;
_forms.scss trunk • ClassFiles/sass-vari... 2
background-color: $color-bg-dark-highlight;
color: $color-fg-light;
_general.scss trunk • ClassFiles/sass-va... 4
color: $color-bg-dark-highlight;
background: $color-bg-light;
border-left: $border-heavy;
color: $color-highlight;
_header.scss trunk • ClassFiles/sass-var... 1
background: $color-bg-light;
_maincontent.scss trunk • ClassFiles/sa... 7
@media (min-width: $screen-md) {
background: $color-bg-light;
@media (min-width: $screen-md) {
border-right: $border-light;
border-bottom: $border-light;
color: $color-fg-dark;
@media (min-width: $screen-md) {
_mainnav.scss trunk • ClassFiles/sass-v... 3
background-color: $color-bg-dark;
color: $color-fg-light;
background-color: $color-bg-dark;

4.5. Advanced Sass

This course has covered the features that you would commonly use in Sass. Sass includes many more advanced features, which are powerful, but less commonly used. Some of these are listed below:

❖ 4.5.1. Operations and Control Directives

You can use calculations like multiplication to construct complex CSS. Operations can be used on numbers, colors, strings, and other data types. Sass also includes control directives for writing conditionals and loops.

See `https://sass-lang.com/documentation/operators` for more information on operations.

See `https://sass-lang.com/documentation/at-rules/control` for more information on control directives.

❖ 4.5.2. Functions

You can use built-in Sass functions and write your own functions in Sass.

See `https://sass-lang.com/documentation/functions` for more information on functions.

❖ 4.5.3. Mixins

Sass allows for the creation and reuse of mixins, the Sass mechanism for grouping CSS declarations into reuseable chunks. A popular Sass toolset called `Bourbon (https://www.bourbon.io/)` provides many useful free mixins for Sass developers.

See `https://sass-lang.com/documentation/at-rules/mixin` for more information on mixins.

❖ 4.5.4. Debugging

Sass includes methods for debugging that are particularly useful when using Sass's more complex features.

See `https://sass-lang.com/documentation/at-rules/debug` for more information on debugging.

Conclusion

In this lesson, you have learned how to use Sass variables and how they make our work easier.