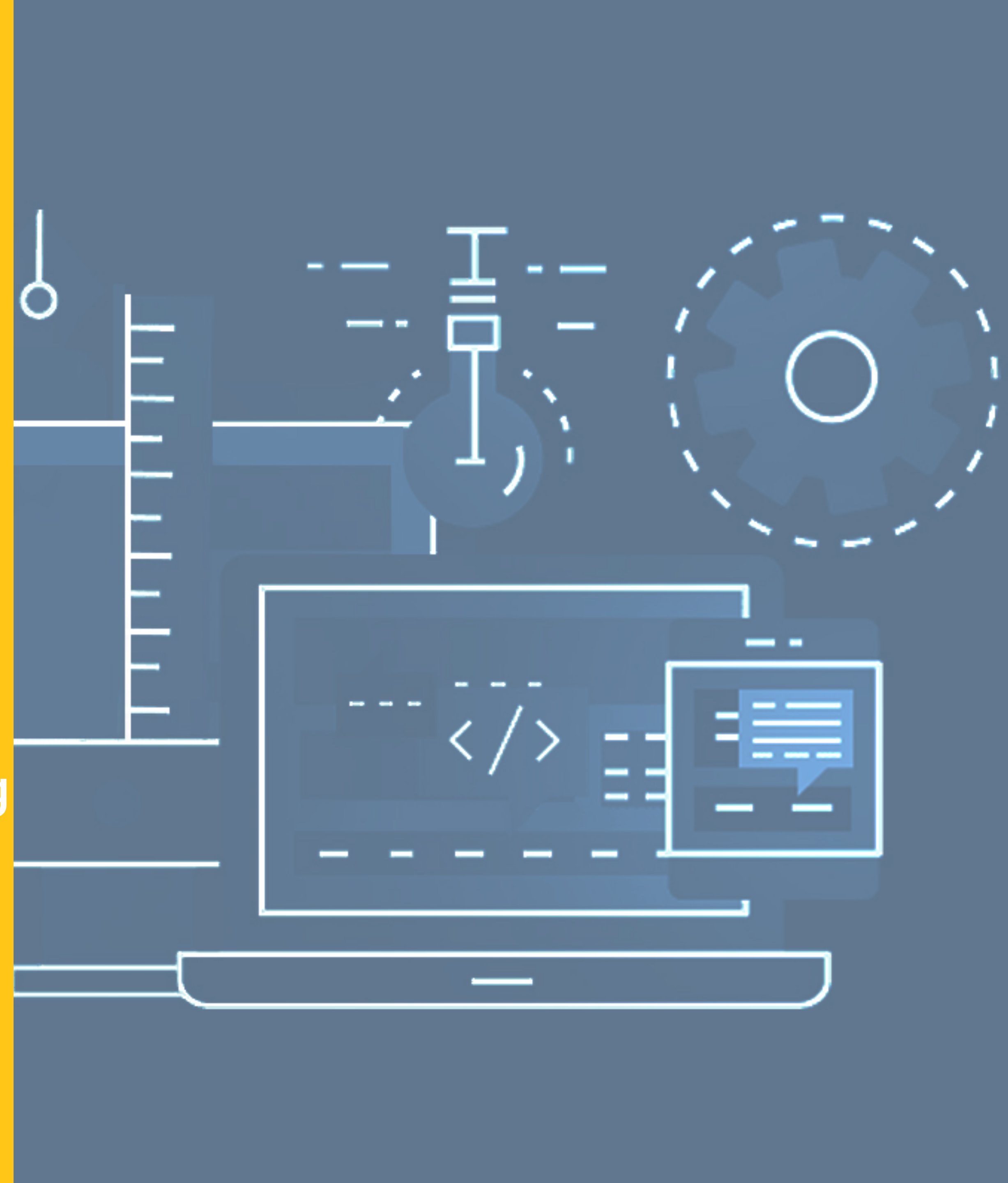


SHELLY GRAHAM, 01/14/2021

WEB DEV 3

WINTER 2021

Week 2: Sass / Language Preprocessing



WEEK 2:



SCSS

```
1  section {  
2    height: 100px;  
3    width: 100px;  
4  
5    .class-one {  
6      height: 50px;  
7      width: 50px;  
8  
9      .button {  
10       color: #074e68;  
11     }  
12   }  
13 }
```

CSS

```
1  section {  
2    height: 100px;  
3    width: 100px;  
4  }  
5  
6  section .class-one {  
7    height: 50px;  
8    width: 50px;  
9  }  
10  
11 section .class-one .button {  
12   color: #074e68;  
13 }
```


WHAT IS SASS?

- Syntactically **A**wesome **S**tyle **S**heet
- It's a preprocessor language!
- It's valid CSS!
- CSS extension language:
 - CSS compatible
 - Feature-rich
 - Mature
 - Industry approved
 - Large community
 - Frameworks



SASS IS A PREPROCESSOR LANGUAGE

- Taking one file and spitting out another
- It takes your code written in Sass and automatically reverts it to regular CSS for you
- Turning .scss files to .css files —> This is why we use src and dist folders!
- Get [Syntax Highlight](#) for your text editor

SCSS	CSS
1 section {	1 section {
2 height: 100px;	2 height: 100px;
3 width: 100px;	3 width: 100px;
4 }	4 }
5 .class-one {	5
6 height: 50px;	6 section .class-one {
7 width: 50px;	7 height: 50px;
8 }	8 width: 50px;
9 .button {	9 }
10 color: #074e68;	10
11 }	11 section .class-one .button {
12 }	12 color: #074e68;
13 }	13 }

WHY SASS?

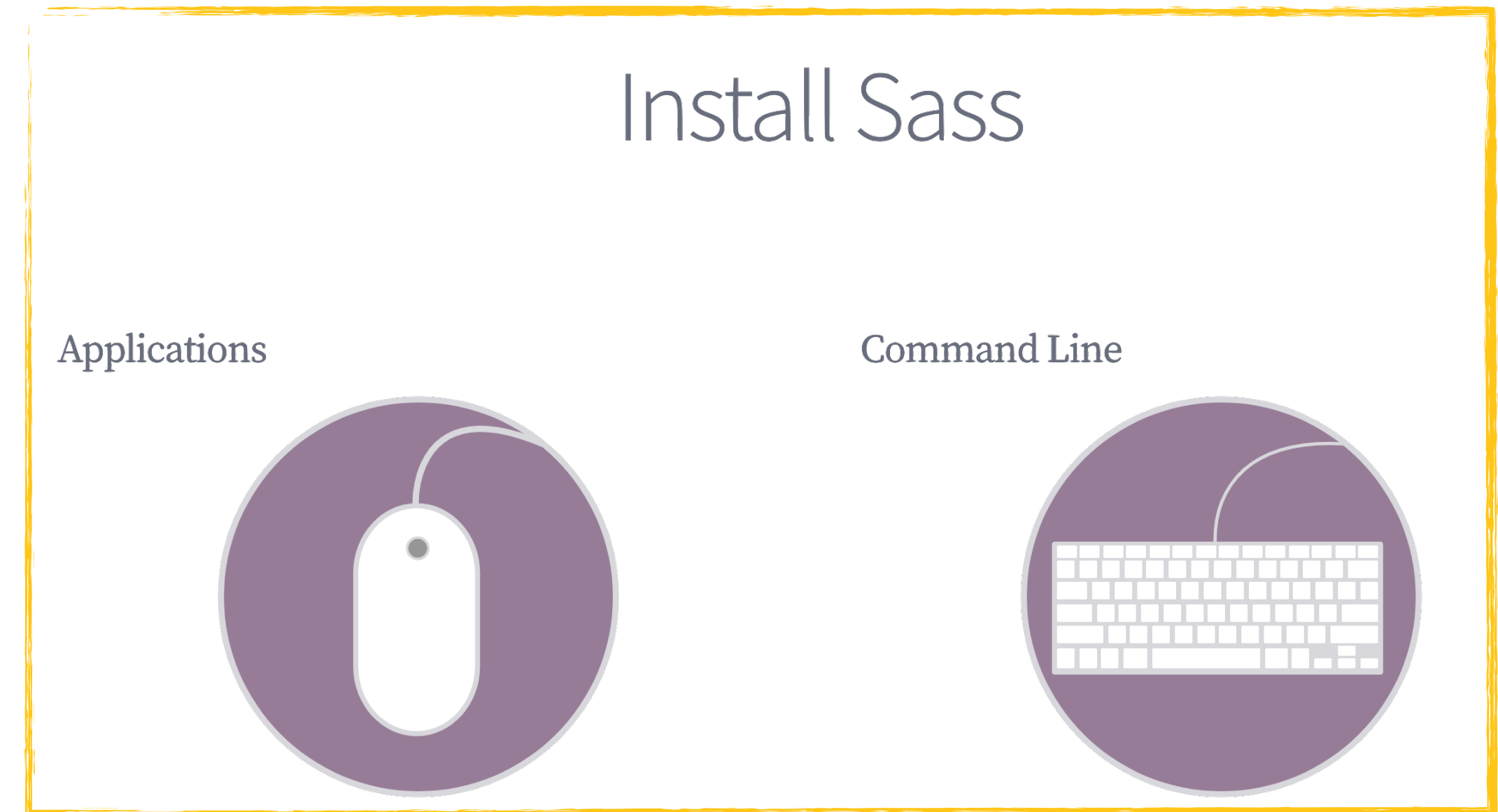
- Helps organize styles in large projects
- Adds new features that vanilla CSS doesn't (yet) have



Awesome Sass

HOW TO USE SASS

- Already set up in Circus Starter!
- You can install manually via Terminal
- Use apps like Prepose or CodeKit to track your changes
- Or use Terminal to track changes via Gulp
- For all methods:
 - Gulp or Gui watches for changes in `.src` folder
 - It pre-processes the changed files
 - Updates files written in src folder in dist folder
 - CSS files will always get overwritten!



THE SASS BASICS

1. VARIABLES

- Assign a value to a particular word
- Use \$ symbol to create variables
- Helps store key values in one place
- You can also do basic math with Sass and use some functions!
- Guidelines:
 - Make variables reusable
 - Keep them together (partial!)

How to write a variable:

\$variable-name: variable;

How to use a variable:

```
$primary-color: #c95fc9;

$regular-fontsize: 2rem * 2;

body {
  background: $primary-color;

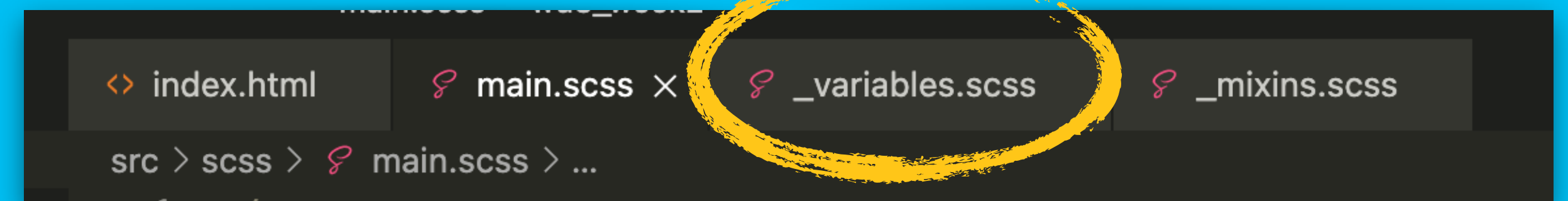
  h2 {
    font-size: $regular-fontsize;
  }
}
```


2. PARTIALS - @USE

- You can create separate Sass files that contain separate code
- Created by saving file with leading underscore _
- Underscore tells Sass that it's only a partial file and therefore won't generate extra CSS file
- Use with @use in main.scss file

How to create partials:

_variables.scss



2. PARTIALS - @USE

- Guidelines:
 - You don't need to use .scss when linking partial in main.scss file
 - No need to add underscore when linking partial either
 - @use order matters!
 - Use for organization, variables and semantic sections of website

How to import partials:

@use "partial-name";

How to use content from partials:

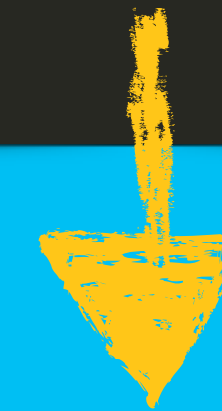
```
@use "reset";  
@use "variables";  
@use "mixins";  
  
body {  
  background: variables.$primary-color;  
}
```

3. NESTING

BASIC NESTING

- Allows you to write nested CSS, just like you write HTML
- Put selectors style rules within parent selectors
- Helps organize code into sections or modules

```
<nav>
  <ul>
    <li><a href="#">Home</a></li>
    <li><a href="#">About Me</a></li>
    <li><a href="#">Contact</a></li>
  </ul>
</nav>
```



```
nav {
  background: lightgreen;

  ul {
    display: flex;

    li {
      background: green;

      a {
        color: white;
      }
    }
  }
}
```

3. NESTING

COMPLEX NESTING

- & stands for “and also”
- > stands for “direct child”
- Guidelines:
 - Don't over specify or nest too deep
 - Apply general rules first
 - Check your output!

```
.test {  
  color: variables.$primary-color;  
  
  //Will be executed if parent ALSO has class .highlight1  
  &.highlight1 {  
    border-bottom: 2px solid ■ white;  
  }  
  
  //Will be applied to first direct child  
  >.highlight2 {  
    border-bottom: 2px solid ■ green;  
  }  
  
  //Will be applied to ALL direct children  
  >* {  
    border-bottom: 2px solid ■ blue;  
  }  
}
```


4. @-RULES AKA DIRECTIVES

- Directives tell Sass to do something special when processing
- Very similar to @directives in vanilla CSS
- Basic @-Rules:
 - @use
 - @media
 - @mixin
 - @include
 - @extend



4.1. @MEDIA

- Acts like standard media query
- Can be nested inline!
- Guidelines:
 - Some people love this, some don't
 - **Pro:** Define all styles for selector in one place
 - **Con:** Media queries are no longer in one central place
 - Try to use with variables and other centralized components!
 - Use sparingly

```
.container {  
  //smaller than 575px  
  width: 90%;  
  
  @media screen and (min-width: variables.$breakpoint-phone) {  
    background: gray;  
    width: 50%;  
  }  
}
```

4.2. @MIXIN

- Code you define once and reuse whenever you want
- This duplicates style rules!
- Can get complex but also very useful
- Great use for breakpoints
- Mixins are everywhere and widely used
- Many pre-written mixing and even mixing libraries like Bourbon

```
@use "variables";

//Small Devices
@mixin bp-small {
  @media screen and (min-width: variables.$breakpoint-phone) {
    @content;
  }
}

//Medium Devices
@mixin bp-medium {
  @media screen and (min-width: variables.$breakpoint-tablet) {
    @content;
  }
}

//Large Devices
@mixin bp-large {
  @media screen and (min-width: variables.$breakpoint-desktop) {
    @content;
  }
}

//XL Devices
@mixin bp-xl {
  @media screen and (min-width: variables.$breakpoint-xl) {
    @content;
  }
}
```

4.3. @INCLUDE

- @include lets you actually use your created mixins
- Allows you to use a mixin anywhere in your code

```
.container {  
  //smaller than 575px  
  width: 90%;  
  
  @include mixins.bp-small {  
    background: grey;  
    width: 50%;  
  }  
  
  @include mixins.bp-medium {  
    background: lightpink;  
    width: 60%;  
  }  
  
  @include mixins.bp-large {  
    background: lightskyblue;  
    width: 70%;  
  }  
  
  @include mixins.bp-xl {  
    background: lightgreen;  
    width: 80%;  
  }  
}
```



4.4. @EXTEND

- Repeats styles you've already written, hence keeps code very dry
- Duplicates class names
- Guidelines:
 - Centralizes generic behavior
 - Base modules are good places to define media queries

```
.general__message {  
  color: ■ pink;  
  background: ■ white;  
  font-size: 1.2rem;  
}  
  
.general__message--warning {  
  @extend .general__message;  
  
  font-size: 1.5rem;  
  border: 2px solid ■ blue;  
}  
  
.general__message--error {  
  @extend .general__message;  
  
  font-size: 2rem;  
  border: 2px solid ■ black;  
  padding: 1.2rem 1.5rem;  
}
```

4.4. @EXTEND

- Can be used with Sass placeholder class
- A placeholder class only shows/executes when written with @extend
- Create placeholder class with the % symbol

```
%error {  
  width: 100px;  
  height: 50px;  
  color:  red  
}  
  
.submit {  
  @extend %error;  
  
  background:  blue;  
}
```

4.5. MORE @-RULES

- **@forward**
- **@function**
- **@error...**

At-Rules ▼

Overview

@use

@forward

@import

@mixin and @include

@function

@extend

@error

@warn

@debug

@at-root

SASS VS. SCSS

- It's confusing but you can encounter both file extensions
- .sass is older than .scss and outdated
- We only use .scss!

THAT'S IT!

- Sass has so many features and methods it can be used
 - Only use what you find useful at first
 - Gradually start using more features as you get comfortable
-
- From now on, I expect you to use Sass for all projects
 - There are no requirements as to how much, but I do want you to get comfortable with the language - it's essential!

QUESTIONS?

SASS DOCUMENTATION

Sass Documentation



SASS Cheat Sheet

Variables & Interpolation:

Sass allows to declare variables that can be used throughout the stylesheet. Variables begin with symbol \$, and are declared just like properties. They can have any value that is allowed for a CSS property, such as colors, numbers (with units), or text. Variables can be used for more than just property values. Indeed is possible to use #{\$var} to insert them into property names or selectors.

SASS

```
$defaultLinkColor: #46EAC2;

a {
  color: $defaultLinkColor;
}

$wk: -webkit-;

.rounded-box {
  #{$wk}border-radius: 4px;
}
```

CSS

```
a {
  color: #46EAC2;
}

.rounded-box {
  -webkit-border-radius: 4px;
}
```

Nesting:

Sass allows to avoid repetition of CSS selectors by nesting the child selectors within the parent selector. It is possible to nest also pseudoclass selectors. The special character & references the parent selector.

SASS

```
ul {
  list-style-type: none;
}
li {
  border: {
    style: solid;
    left: {
      width: 1px;
      color: #999999;
    }
  }
  display: inline-block;
  margin: 0;
  padding: 0 5px;
}
a {
  text-decoration: none;
  &:hover { text-decoration: underline; }
}
}
```

CSS

```
ul {
  list-style-type: none;
}
ul li {
  border-style: solid;
  border-left-width: 1px;
  border-left-color: #999999;
  display: inline-block;
  margin: 0;
  padding: 0 5px;
}
ul li a {
  text-decoration: none;
}
ul li a:hover {
  text-decoration: underline;
}
}
```

Operations and Functions:

In a SASS stylesheet is possible to use the classic arithmetic operations; moreover the SASS engine makes available a large set of new useful functions. The entire list of these functions could be found @ <http://sass-lang.com/docs/yardoc/Sass/Script/Functions.html>

SASS

```
$defaultWindowSize: 960px;

#nav_side {
  float: right;
  width: $defaultWindowSize / 3;
}
a {
  color: lighten($defaultLinkColor, 10%);
}
}
```

CSS

```
#nav_side {
  float: right;
  width: 320px;
}
#nav_side a {
  color: #74EFD1;
}
}
```

Mixins:

Mixins allow re-use of styles without having to copy and paste them or move them into a non-semantic class. To define a mixin is used the @mixin directive, which takes a block of styles that can then be included in another selector using the @include directive.

SASS

```
@mixin default-box {
  $borderColor: #666;
  border: 1px solid $borderColor;
  clear: both;
  display: block;
  margin: 5px 0;
  padding: 5px 10px;
}

footer, header { @include default-box; }
```

CSS

```
footer, header {
  border: 1px solid #666;
  clear: both;
  display: block;
  margin: 5px 0;
  padding: 5px 10px;
}
```

Arguments:

The real power of mixins comes when you pass them arguments. Arguments are declared as a parenthesized, comma-separated list of variables. Each of those variables is assigned a value each time the mixin is used.

SASS

```
@mixin default-box($color, $boxModel, $padding) {
  $borderColor: $color;
  border: 1px solid $borderColor;
  clear: both;
  display: block;
  margin: 5px 0;
  padding: 5px $padding;
}

header { @include default-box(#666, block, 10px); }
footer { @include default-box(#999, inline-block, 5px); }
```

CSS

```
header {
  border: 1px solid #666;
  clear: both;
  display: block;
  margin: 5px 0;
  padding: 5px 10px;
}

footer {
  border: 1px solid #999;
  clear: both;
  display: inline-block;
  margin: 5px 0;
  padding: 5px 5px;
}
```

Selector Inheritance:

The SASS @extend directive makes possible for a selector to inherit all the styles of another selector without duplicating the CSS properties.

SASS

```
.error {
  border: 1px #F00;
  background: #FDD;
}

.badError {
  @extend .error;
  border-width: 3px;
}
```

CSS

```
.error, .badError {
  border: 1px #F00;
  background: #FDD;
}

.badError {
  border-width: 3px;
}
```

Import stylesheet

SASS @import directive allows to break a stylesheet up into multiple files. Any style rules, variables or mixins defined in @imported files are available to the files that import them. Usually, as naming convention, the name of the files meant to be imported, begins with an underscore. In order to support both .scss and .sass files, SASS allows files to be imported without specifying a file extension.

SASS

```
@import 'partials/_vars';

body {
  color: $color;
}

/* content of _vars.scss */
$color: #333;
```

CSS

```
body {
  color: #333;
}
```

[blog](#) | [twitter](#) | [g+](#) | [linkedin](#)

HOMEWORK - PART 1

- Make a new folder called “wd3_week2_sass”, and add a copy of the Circus Starter to it. Create a Git repository. Make a pricing page for a fake software product. Good examples of the format I'm looking for:
 - <https://store.unity.com/>
 - <https://bitbucket.org/product/pricing?tab=host-in-the-cloud>
 - <https://slack.com/pricing>
 - <https://www.invisionapp.com/planss>
- The site itself should be a simple landing page to purchase three tiers of a software service, built using Sass. At the minimum, your Sass file should contain examples of the following:
 - Nested style rules. No specific number here, just try to nest things in a way that is efficient and makes sense to you. This is one of Sass' handiest features.
 - At least two variables, one for a primary text color, one for a secondary accent color for links and buttons.
 - At least one inline media query, to make your page go from a three column layout to one column. Bonus points if you use a Mixin for this.
 - At least one partial Sass file included in your main file.
 - At least one Mixin. This can be as simple or complex as you like. While the breakpoint Mixin we discussed in class is useful, I encourage you to look online for other Sass Mixins as well.

HOMEWORK - PART 2

- The page itself must be responsive and should include 2 main sections:
 - Section 1 should contain: A header image & a Call-to-action text
 - Section 2 should contain three columns, each with a module that lists one level of pricing for your fake service. Each module should include:
 - A title
 - A price
 - A short description
 - A list of features
 - A Buy button
 - Additionally, one of these columns should have some unique styling, to indicate that it is the suggested price tier. Use `@extend` to make styling modifications for this section.
- You will be graded on your Sass file as much as the site itself. Keep in mind that the main goal of this assignment is to get you comfortable using Sass on a regular basis, so focus your efforts on writing effective, reusable styles. I will look at your Sass files in future projects, but this is the best time to ask about best practices and figure out the parts that you like using. There are about a million bad habits you can develop while using sass, so let's try to address them early.
- Submit a link to the Git repository for this assignment.

FIN