DIGNITAS DIGITAL

Sass

- Syntactically Awesome Stylesheet
- Sass is a CSS pre-processor
- Look like CSS
- Sass is completely compatible with all versions of CSS
- Sass reduces repetition of CSS and therefore saves time
- Sass was designed by Hampton Catlin and developed by Natalie Weizenbaum in 2006

Why Use Sass?

- Stylesheets are getting larger, more complex, and harder to maintain. This is where a CSS pre-processor can help.
- Sass lets you use features that do not exist in CSS, like variables, nested rules, mixins, imports, inheritance, built-in functions, and other stuff.

How to Use/install Sass?

Applications

CodeKit (Paid) Mac

Compass.app (Paid, Open Source) Mac Windows Linux

Ghostlab (Paid) Mac Windows

Hammer (Paid) Mac

Koala (Open Source) Mac Windows Linux

LiveReload (Paid, Open Source) Mac Windows

Prepros (Paid) Mac Windows Linux

Scout-App (Free, Open Source) Windows Linux Mac

Command Line

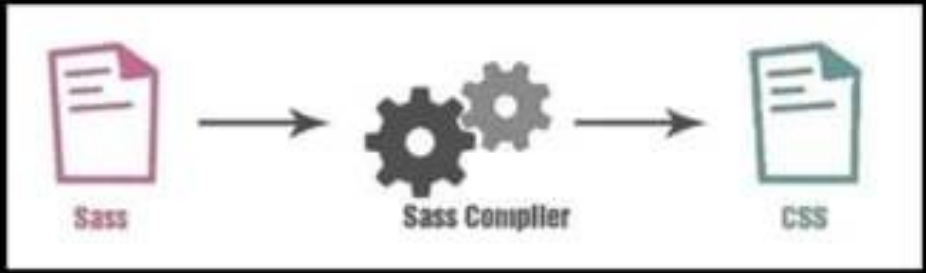
When you install Sass on the command line, you'll be able to run the sass executable to compile .sass and .scss files to .css files. For example:

```
sass source/stylesheets/index.scss build/stylesheets/index.css
```

```
npm install -g sass
```

How Does Sass Work?

A browser does not understand Sass code. Therefore, you will need a Sass pre-processor to convert Sass code into standard CSS.



A Simple Example

Let's say we have a website with three main colors:

#a2b9bc
#b2ad7f
#878f99

So, many times we type those HEX values.

Instead of typing the above values a lot of times, we can use Sass and write this:

Sass Example

```
/* define variables for the primary colors */
$primary_1: #a2b9bc;
$primary_2: #b2ad7f;
$primary_3: #878f99;

/* use the variables */
.main-header {
  background-color: $primary_1;
}

.menu-left {
  background-color: $primary_2;
}

.menu-right {
  background-color: $primary_3;
}
```

So, when using Sass, and the primary color changes, you only need to change it in one place.

Variables

Variables as a way to store information that you want to reuse throughout your stylesheet.

SCSS Sass

```scss
$font-stack:    Helvetica, sans-serif;
$primary-color: #333;

body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

CSS

```css
body {
  font: 100% Helvetica, sans-serif;
  color: #333;
}
```

Nesting

When writing HTML you've probably noticed that it has a clear nested and visual hierarchy.

Sass will let you nest your CSS selectors in a way that follows the same visual hierarchy of your HTML.

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

  li { display: inline-block; }

  a {
    display: block;
    padding: 6px 12px;
    text-decoration: none;
  }
}
```

```
nav ul {
  margin: 0;
  padding: 0;
  list-style: none;
}
nav li {
  display: inline-block;
}
nav a {
  display: block;
  padding: 6px 12px;
  text-decoration: none;
}
```

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
</head>
<body>
    <nav>
        <ul>
            <li>
                <a href="#">link</a>
            </li>
            <li>
                <a href="#">link</a>
            </li>
        </ul>
    </nav>
</body>
</html>
```

Partials

You can create partial Sass files that contain little snippets of CSS that you can include in other Sass files. This is a great way to modularize your CSS and help keep things easier to maintain. A partial is a Sass file named with a leading underscore. You might name it something like _partial.scss. The underscore lets Sass know that the file is only a partial file and that it should not be generated into a CSS file. Sass partials are used with the @use rule.

```
// Theme files

@import 'theme/variables';
@import 'theme/typography';
@import 'theme/buttons';
@import 'theme/forms';
@import 'theme/images';
@import 'theme/cf7';

// Layout

@import 'layout/siteheader';
@import 'layout/mobilenav';
@import 'layout/bottomchecklist';
@import 'layout/footer';

// Modules: Generic

@import 'modules/pageheader';
@import 'modules/singlecolumn';

// Modules: Homepage
```

Modules

You don't have to write all your Sass in a single file. You can split it up however you want with the @use rule. This rule loads another Sass file as a moduleUsing a file will also include the CSS it generates in your compiled output!

```
// _base.scss
$font-stack:    Helvetica, sans-serif;
$primary-color: #333;

body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

```
// styles.scss
@use 'base';

.inverse {
  background-color: base.$primary-color;
  color: white;
}
```

```
body {
  font: 100% Helvetica, sans-serif;
  color: #333;
}

.inverse {
  background-color: #333;
  color: white;
}
```

This PC > Local Disk (D:) > Shafeeq > Joshua > yellowexpress > Yellow Express > Yellow Express > scss >

Name	Date modified	Type	Size
layout	7/5/2018 11:48 PM	File folder	
modules	7/5/2018 11:48 PM	File folder	
theme	7/5/2018 11:48 PM	File folder	
.DS_Store	7/6/2018 12:21 AM	DS_STORE File	7 KB
style.scss	11/29/2019 11:41 ...	SCSS File	2 KB

Notice we're using @use 'base'; in the styles.scss file. When you use a file you don't need to include the file extension. Sass is smart and will figure it out.

Mixins

Some things in CSS are a bit tedious to write, especially with CSS3 and the many vendor prefixes that exist. A mixin lets you make groups of CSS declarations that you want to reuse throughout your site. You can even pass in values to make your mixin more flexible. A good use of a mixin is for vendor prefixes. Here's an example for transform.

SCSS Sass

```
@mixin transform($property) {
  -webkit-transform: $property;
  -ms-transform: $property;
  transform: $property;
}
.box { @include transform(rotate(30deg)); }
```

CSS

```
.box {
  -webkit-transform: rotate(30deg);
  -ms-transform: rotate(30deg);
  transform: rotate(30deg);
}
```

Extend/Inheritance

This is one of the most useful features of Sass. Using @extend lets you share a set of CSS properties from one selector to another.

SCSS Sass ⇒ CSS

```scss
/* This CSS will print because %message-shared is extended. */
%message-shared {
  border: 1px solid #ccc;
  padding: 10px;
  color: #333;
}

// This CSS won't print because %equal-heights is never extended.
%equal-heights {
  display: flex;
  flex-wrap: wrap;
}

.message {
  @extend %message-shared;
}

.success {
  @extend %message-shared;
  border-color: green;
}

.error {
  @extend %message-shared;
  border-color: red;
}

.warning {
  @extend %message-shared;
  border-color: yellow;
}
```

SCSS Sass ⇒ CSS

```scss
/* This CSS will print because %message-shared is extended. */
.message, .success, .error, .warning {
  border: 1px solid #ccc;
  padding: 10px;
  color: #333;
}

.success {
  border-color: green;
}

.error {
  border-color: red;
}

.warning {
  border-color: yellow;
}
```

Operators

Doing math in your CSS is very helpful. Sass has a handful of standard math operators like +, -, *, /, and %.

SCSS Sass

```
.container {
  width: 100%;
}

article[role="main"] {
  float: left;
  width: 600px / 960px * 100%;
}

aside[role="complementary"] {
  float: right;
  width: 300px / 960px * 100%;
}
```

CSS

```
.container {
  width: 100%;
}

article[role="main"] {
  float: left;
  width: 62.5%;
}

aside[role="complementary"] {
  float: right;
  width: 31.25%;
}
```

SASS AND MEDIA QUERIES

SASS

```
section.main {
    float: left;
    width: 65%;
    font-size: 16px;
    Line-height: 1.4;

    @media screen and (max-width: 800px) {
    float: none;
    width: auto;
    }

    @media screen and (max-width: 500px) {
    font-size: 12px;
    line-height: 1.4;
    }
}
```

CSS

```
section.main {
    float: left;
    width: 65%;
    font-size: 16px;
    line-height: 1.4;
}
@media screen and (max-width: 800px) {
    section.main {
    float: none;
    width: auto;
    }
}
@media screen and (max-width: 500px) {
    section.main {
    font-size: 12px;
    line-height: 1.4;
    }
}
```

THANK YOU!

DIGNITAS DIGITAL