What is Sass?

* **Sass** stands for **S**yntactically **A**wesome **S**tyle**s**heet
* Sass is an extension to CSS
* Sass is a CSS pre-processor
* 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
* Sass is free to download and use

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.

## /\* 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; }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.

This process is called transpiling. So, you need to give a transpiler (some kind of program) some Sass code and then get some CSS code back.

**Tip:** Transpiling is a term for taking a source code written in one language and transform/translate it into another language.

Sass File Type

Sass files has the ".scss" file extension.

Sass Comments

Sass supports standard CSS comments /\* comment \*/, and in addition it supports inline comments // comment:

Variables are a way to store information that you can re-use later.

With Sass, you can store information in variables, like:

* strings
* numbers
* colors
* booleans
* lists
* nulls

Sass uses the $ symbol, followed by a name, to declare variables:

Sass Variable Syntax:

$*variablename*: *value*;

$myFont: Helvetica, sans-serif;  
$myColor: red;  
$myFontSize: 18px;  
$myWidth: 680px;  
  
body {  
  font-family: $myFont;  
  font-size: $myFontSize;  
  color: $myColor;  
}  
  
#container {  
  width: $myWidth;  
}

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

Sass keeps the CSS code DRY (Don't Repeat Yourself). One way to write DRY code is to keep related code in separate files.

You can create small files with CSS snippets to include in other Sass files. Examples of such files can be: reset file, variables, colors, fonts, font-sizes, etc.

## Sass Importing Files

Just like CSS, Sass also supports the @import directive.

The @import directive allows you to include the content of one file in another.

The CSS @import directive has a major drawback due to performance issues; it creates an extra HTTP request each time you call it. However, the Sass @import directive includes the file in the CSS; so no extra HTTP call is required at runtime!

Sass Import Syntax:

@import *filename*;

**Tip:** You do not need to specify a file extension, Sass automatically assumes that you mean a .sass or .scss file. You can also import CSS files. The @import directive imports the file and any variables or mixins defined in the imported file can then be used in the main file.

You can import as many files as you need in the main file:

### **Example**

@import "variables";  
@import "colors";  
@import "reset";

Let's look at an example: Let's assume we have a reset file called "reset.scss", that looks like this:

### **Example**

SCSS Syntax (reset.scss):

html,  
body,  
ul,  
ol {  
  margin: 0;  
  padding: 0;  
}

and now we want to import the "reset.scss" file into another file called "standard.scss".

Here is how we do it: It is normal to add the @import directive at the top of a file; this way its content will have a global scope:

SCSS Syntax (standard.scss):

@import "reset";  
  
body {  
  font-family: Helvetica, sans-serif;  
  font-size: 18px;  
  color: red;  
}

Sass Mixins

The @mixin directive lets you create CSS code that is to be reused throughout the website.

The @include directive is created to let you use (include) the mixin.

Defining a Mixin

A mixin is defined with the @mixin directive.

Sass @mixin Syntax:

@mixin *name*{  
  *property*: *value*;  
  *property*: *value*;  
  ...  
}

The following example creates a mixin named "important-text":

SCSS Syntax:

@mixin important-text {  
  color: red;  
  font-size: 25px;  
  font-weight: bold;  
  border: 1px solid blue;  
}

**Tip:** A tip on hyphens and underscore in Sass: Hyphens and underscores are considered to be the same. This means that @mixin important-text { } and @mixin important\_text { } are considered as the same mixin!

Using a Mixin

The @include directive is used to include a mixin.

Sass @include mixin Syntax:

*selector*{  
  @include *mixin-name*;}

So, to include the important-text mixin created above:

SCSS Syntax:

.danger {  
  @include important-text;  
  background-color: green;  
}

Sass @extend Directive

The @extend directive lets you share a set of CSS properties from one selector to another.

The @extend directive is useful if you have almost identically styled elements that only differ in some small details.

The following Sass example first creates a basic style for buttons (this style will be used for most buttons). Then, we create one style for a "Report" button and one style for a "Submit" button. Both "Report" and "Submit" button inherit all the CSS properties from the .button-basic class, through the @extend directive. In addition, they have their own colors defined:

SCSS Syntax:

.button-basic  {  
  border: none;  
  padding: 15px 30px;  
  text-align: center;  
  font-size: 16px;  
  cursor: pointer;  
}  
  
.button-report  {  
  @extend .button-basic;  
  background-color: red;  
}  
  
.button-submit  {  
  @extend .button-basic;  
  background-color: green;  
  color: white;  
}