SAAS

(**S**yntactically **A**wesome **S**tyle**s**heet)

**Introduction**

**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

**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.

**SASS Code Snippet as below** ,

$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.

## 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.

SASS file extension as ‘**.scss’.**

## Sass Comments

## Multiline Comment

## /\* define primary colors \*/ $primary\_1: #a2b9bc; $primary\_2: #b2ad7f; /\* use the variables \*/

## Inline Comment

## .main-header {   background-color: $primary\_1; // here you can put an inline comment }

**SAAS Variables**

## Sass Variables : ‘$’ symbol, followed by a name, to declare variables.

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

SAAS variable’s Syntax : $*variablename*: value;

Example :

SCSS Syntax

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

CSS Output:

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

## Sass Variable Scope :

## Example :

## SCSS Syntax

## $myColor: red; h1 {   $myColor: green;   color: $myColor; } p {   color: $myColor; }

CSS Output:

h1 {  
  color: green;  
}  
  
p {  
  color: red;  
}

## Note :

## <p> tag be red or green? It will be red .

## $myColor: green; is inside the <h1> rule, and will only be available there.

## Using Sass !global

The default behavior for variable scope can be overridden by using the !global switch.

!global indicates that a variable is global, which means that it is accessible on all levels.

Example :

SCSS Syntax:

$myColor: red;  
  
h1 {  
  $myColor: green !global;  
  color: $myColor;  
}  
  
p {  
  color: $myColor;  
}

CSS Output:

h1 {  
  color: green;  
}  
  
p {  
  color: green;  
}

**SASS Nesting**

Nesting has been done same way as HTML .

**SCSS Syntax:**

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

Notice that in Sass, the ul, li, and a selectors are nested inside the nav selector.

CSS Syntax:

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

## Sass Nested Properties

## Many CSS properties have the same prefix, like font-family, font-size and font-weight or text-align, text-transform and text-overflow.

## Example

**SCSS Syntax:**

font: {  
  family: Helvetica, sans-serif;  
  size: 18px;  
  weight: bold;  
}  
  
text: {  
  align: center;  
  transform: lowercase;  
  overflow: hidden;  
}

**CSS Output :**

font-family: Helvetica, sans-serif;  
font-size: 18px;  
font-weight: bold;  
  
text-align: center;  
text-transform: lowercase;  
text-overflow: hidden;

# **SASS @import and Partials**

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

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

## Sass @import directive includes the file in the CSS; so no extra HTTP call is required at runtime

**Sass Import Syntax:** @import *filename*;

**Note :**

* 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.

Example

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

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;  
}

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

**SCSS Syntax** (**standard.scss**):

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

**CSS output:**

html, body, ul, ol {  
  margin: 0;  
  padding: 0;  
}  
  
body {  
  font-family: Helvetica, sans-serif;  
  font-size: 18px;  
  color: red;  
}

## SASS Partials

By default, Sass transpiles all the .scss files directly. However, when you want to import a file, you do not need the file to be transpiled directly.

Sass has a mechanism for this: If you start the filename with an underscore, Sass will not transpile it. Files named this way are called partials in Sass.

So, a partial Sass file is named with a leading underscore

**Sass Partial Syntax:** \_*filename*;

### **Example**

**"\_colors.scss**":

$myPink: #EE82EE;  
$myBlue: #4169E1;  
$myGreen: #8FBC8F;

### **Note :** Import the partial file, omit the underscore. Sass understands that it should import the file "\_colors.scss"

### **Example**

@import "colors";  
  
body {  
  font-family: Helvetica, sans-serif;  
  font-size: 18px;  
  color: $myBlue;  
}

# **SASS @mixin and @include**

## 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*;  
  ...  
}

## Example creates a mixin named "important-text" as below,

**SCSS Syntax:**

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

Note : 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*;}

**SCSS Syntax:**

.danger {  
  @include important-text; // included the important-text mixin   
  background-color: green;  
}

**CSS output:**

.danger {  
  color: red;  
  font-size: 25px;  
  font-weight: bold;  
  border: 1px solid blue;  
  background-color: green;  
}

**A mixin can also include other mixins**:

SCSS Syntax:

@mixin special-text {  
  @include important-text; // Another mixin included   
  @include link;  
  @include special-border;  
}

## Passing Variables to a Mixin:

**SCSS Syntax:**

/\* Define mixin with two arguments \*/  
@mixin bordered($color, $width) {  
  border: $width solid $color;  
}  
  
.myArticle {  
  @include bordered(blue, 1px);  // Call mixin with two values  
}  
  
.myNotes {  
  @include bordered(red, 2px); // Call mixin with two values  
}

Notice that the arguments are set as variables and then used as the values (color and width) of the border property.

**CSS Output:**

.myArticle {  
  border: 1px solid blue;  
}  
  
.myNotes {  
  border: 2px solid red;  
}

## Default Values for a Mixin

## It is also possible to define default values for mixin variables

**SCSS Syntax:**

@mixin bordered($color: blue, $width: 1px) {  
  border: $width solid $color;  
}

Then, you only need to specify the values that change when you include the mixin:

**SCSS Syntax:**

.myTips {  
  @include bordered($color: orange);  
}

## Using a Mixin For Vendor Prefixes

**SCSS Syntax:**

@mixin transform($property) {  
  -webkit-transform: $property;  
  -ms-transform: $property;  
  transform: $property;  
}  
  
.myBox {  
  @include transform(rotate(20deg));  
}

**CSS Output:**

.myBox {  
  -webkit-transform: rotate(20deg);  
  -ms-transform: rotate(20deg);  
  transform: rotate(20deg);  
}

# **SASS @extend and Inheritance**

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; // Inherited **.button-basic** Property   
  background-color: red;  
}  
  
.button-submit  {  
  @extend .button-basic; // Inherited **.button-basic** Property   
  background-color: green;  
  color: white;  
}

**CSS Output:**

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

**Note :** By using the @extend directive, you do not need to specify several classes for an element in your HTML code, like this: <button class="button-basic button-report">Report this</button>. You just need to specify **.button-report** to get both sets of styles.

The @extend directive helps keep your Sass code very DRY.