SASS

Overview

SASS (Syntactically Awesome Style Sheets) allows for features which don’t exist in CSS such as variables and mixins to be used and then compiled into CSS files. SASS uses two syntaxes, .scss and .sass, scss is the most common and is what will be shown here.

Note, many new functions of sass are not supported by the SassC library yet. For example to use the @use feature, the dartsass complier must be used.

Variables

Objects to store CSS information which can be reused throughout the stylesheet. SASS uses the $ symbol to make a variable such as:

$font-stack: Helvetica, sans-serif;

$primary-color: #333;

body {

font: 100% $font-stack;

color: $primary-color;

}

Variables which start with - or \_ are private variables and will not be exported outside of their normal scope with @use imports.

Nesting

Similar to HTML, SASS allows for clear nesting in CSS. Overly nested CSS can prove hard to maintain so it should be kept to a minimum:

nav {

ul {

margin: 0;

padding: 0;

list-style: none;

}

li { display: inline-block; }

a {

display: block;

padding: 6px 12px;

text-decoration: none;

}

}

Modules - @use

SASS can be written in multiple files (partials) and imported as modules using the @use rule. No matter how many times @use is called for a specific partial, the file will only be imported once. Variables, mixins, and extends can be used between files, however they do get namespaced by default. Therefore, to use variables across files, reassign the variables required at the top of the file:

// \_base.scss

$font-stack: Helvetica, sans-serif;

$primary-color: #333;

body {

font: 100% $font-stack;

color: $primary-color;

}

// styles.scss

@use 'base';

$base-color: base.$primary-color;

.inverse {

background-color: $base-color;

color: white;

}

Namespaces can also be changed by adding the as <name> syntax to the end of the @use import, with \* removing the namespace:

// styles.scss

@use 'base' as \*;

$base-color: $primary-color;

@use 'typograpgy' as t;

$text-color: t.$primary-color;

Mixins

Mixins let you make groups of CSS declarations which can be resused throughout the application. To make a mixin, use the @mixin rule and give the mixin a name, it can then be used as a CSS declaration starting with an @include:

@mixin transform($property) {

-webkit-transform: $property;

-ms-transform: $property;

transform: $property;

}

.box { @include transform(rotate(30deg)); }

Extend/Inheritance

@extend allow sharing of a set of CSS from one selector to another. Extends can be used with both normal selectors and also placeholders which start with a % symbol and are only printed if extended into a class.

/\* This CSS will print because %message-shared is extended. \*/

%message-shared {

border: 1px solid #ccc;

padding: 10px;

color: #333;

}

.message {

@extend %message-shared;

}

.success {

@extend %message-shared;

border-color: green;

}

.error {

@extend %message-shared;

border-color: red;

}

Operators

SASS allows for simple mathematical operators to be used in the file:

.container {

width: 100%;

}

article[role="main"] {

float: left;

width: 600px / 960px \* 100%;

}

aside[role="complementary"] {

float: right;

width: 300px / 960px \* 100%;

}

Maps

SASS maps allow for keys to be assoicated with values, similar to a ruby hash:

$breakpoints: (

'small' : 767px,

'medium' : 992px,

'large' : 1200px

);

Responsive Design

Using maps and a mixin to define screen size in one place in the application, then use strings to define specific

Default Variables

Re-usable libraries and third party software, often come with standard global variables, with set defaults:

// \_buttons.scss

$color: blue !default;

The default variables can be overwritten on import using the with function:

@use 'buttons' with (

$color: red,

$style: 'flat',

);

Forward

To collect files in one location, but not actually use/import them, the forward rule can be used. For example, multiple form related partials can be forwarded in a file for importing as a single namespace later:

// forms/\_index.scss

@forward 'input';

@forward 'textarea';

@forward 'select';

@forward 'buttons';

If the partials have conflicting variables they can be forwarded in with a namespace prefix:

@forward 'input' as input-\*;

@forward 'buttons' as btn-\*;

Will then be imported as:

@use 'forms';

@include forms.input-background();

@include forms.btn-background();

SASS File Structure

Work in progress:

/\*

stylesheets/

|

|– base/

| |– \_normalize.scss # Normalize

| |– \_typography.scss # Typography rules

| |– \_globals.scss # Sass Variables

| |– \_functions.scss # Sass Functions

| |– \_mixins.scss # Sass Mixins

|

|– components/

| |– \_buttons.scss # Buttons

| |– \_messages.scss # Messages

| |– \_forms.scss # Forms

| |– \_navbar.scss # Navbar - top and side

| |– \_comments.scss # Comments

|

|– vendor/

| |– \_hamburger.scss # Hamburger icon for navbar

|

|– ./ # Page specific styles auto imported by rails

|

`– main.scss # Primary Site Sass file

\*/