Sass is completely compatible with all versions of css. Sass is a preprocessor scripting language that is interpreted or compiled into cascading style sheet. SassScript is the scripting language itself. there are two main syntax and the newer scss syntax.

Sass

Sass is stands for syntactical awesome style sheet. Sass is different from scss only in syntax. It uses indentation to separate code blocks and doesnt use semicolons or branches.

Scss

Scss is stands for sassy css. Scss is different from sass only in syntax. It uses a more css like with curly braces and semicolons.

Preprocessing

Variables

Nesting

Partials

Modules

Mixins

Inheritance

Operators

Preprocessing

Sass is a preprocessor scripting language that is interpreted or compiled into css. A preprocessor is essentially a tool that extends the capabilities of css, providing features that are not natively available in standard css.

Css on its own can fun, but stylesheet are getting larger, more complex, and harder tot maintain. This is where a preprocessor can help. Sass has features that doesn’t exist in css yet like nesting, mixins, inheritance, and other nifty goodies that help you write robust, maintainable css.

Once we start tinkering with sass, it will take your preprocessed sass file and save it as a normal css file that we can use in your website.

If we want to watch our input.scss file input.sccs file, we had just add the watch flag to our command

sass –watch input.scss output.css

we can watch output to directories by using folder paths as our input and output and separating them with a colon

sass –watch app/sass:public/stylesheets

variables

variables are used to store information that we want to reuse throughout our stylesheet. We can store things like colors, font stack or any css value we think we will want to reuse.

Sass use the $ symbol to make a variable and use that variable with the respective properties of the css

$font-stack: Helvetica, sans-serif;

$primary-color: #333;

body{

font: 100% $font-stack;

color: $primary-color;

}

Sass

Sass stands for syntactical awesome style sheet. Scss / Sassy css and sass are css preprocessor that allow us to use variables, nested rules, mixins, functions, and more, all with a more powerful and expressive syntax than regular css. Both css and sass are part of the same project but have sligthtly different syntax styles.

Sass is the most mature, stables, and powerful professional grade css extension language in the word.

Sass

The original sass syntax is indentation based and doesn’t use curly braces / semicolons

file.sass

$primary-color: #333

body

color: $primary-color

scss

scss is more similar to regular css, using curly braces and semicolons

file.scss

$primary-color: #333;

body {

color: $primary-color;

}

Css compatible

Sass is completely compatible with all versions of css. We take this compatibility seriously, so that we can seamlessly use any available css libraries.

Feature rich

Sass boats more feature and abilities that any other css extension language out there.

Mature

Sass has been actively supported for over 17 year by its loving core team.

Industry approved

Over and over again, the industry is choosing sass as the premier css extension language

Feature of scss / sass

Variables

We can define variables for colors, fonts / any css value, making our css more maintainable.

$primary-color: #3497db;

body {

color: $primary-color;

}

Nesting

Sass allows us to nest css selectors in a way that follows the same visual hierarchy of our html

nav {

ul {

margin: 0;

padding: 0;

list-style: none;

}

li { display: inline-block; }

a {

text-decoration: none;

&:hover { text-decoration: underline; }

}

}

Partials and import

We can split our css into smaller, reusable pieces and import them into a main stylesheet.

// \_reset.scss

\* {

margin: 0;

padding: 0;

}

// styles.scss

@import 'reset';

body {

font-family: Arial, sans-serif;

}

Mixins

Mixins allow us to create reusable styles. We can even pass arguments to them

@mixin border-radius($radius) {

-webkit-border-radius: $radius;

-moz-border-radius: $radius;

-ms-border-radius: $radius;

border-radius: $radius;

}

.box { @include border-radius(10px); }

Inheritance / extends

We can share a set of css properties from one selector to another.

.message {

border: 1px solid #ccc;

padding: 10px;

color: #333;

}

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

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

Operators

We can perform calculation with operators like +, -, \*, /

.container {

width: 100%;

margin-left: 10px / 2;

}

Functions

Sass provides many built in functions for manipulating colors, numbers, lists. We can also define our own functions

Usecases

Large project, the ability to reuse and maintain code with variables , mixins, and functions makes sass/scss invaluable.

Consistent styling, if we need consistent styling across various elements / pages variables and mixins help maintain this consistency.

Dry, don’t repeat yourself sass/scss helps reduce redundancy in our styles making our code cleaner and easier to manage.