Sass basics

Has features like mixing, inheritance.

Variables in

SASS

Variables in sass can store things like colors, font stacks, or any css.

* Value which you think you can reuse.
* Example:
* Sass:
* $font-stack: Helvetica, sans-serif

$primary-color: #333

body

font: 100% $font-stack

color: $primary-color

Nesting

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

Nesting done in css:

.nav ul{

Property: value}

.nav li{

Property: value;

}

.nav a{

Property: value;

}

Nesting done using Sass

nav

ul

property: value;

li

property: value;

a

property: value;

nesting done using Scss

nav{

ul{

property: value;

}

li{ property: value;

}

a{property: value;

}

}

Modules

CSS code can be split up into more files.

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 module, which means you can refer to its variables, mixins, and functions in your Sass file with a namespace based on the filename. Using a file will also include the CSS it generates in your compiled output!

USING MODULES IN SCSS

1. Base.scss file extension
2. In style.scss file include

@use ‘base’;

Using modules in SASS

1. Base.sass
2. In style.sass include

@use ‘base’

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. It helps keep your Sass very DRY. You can even pass in values to make your mixin more flexible. Here’s an example for theme.

@mixin theme($theme: DarkGray)

background: $theme

box-shadow: 0 0 1px rgba($theme, .25)

color: #fff

.info

@include theme

.alert

@include theme($theme: DarkRed)

To create a mixin you use the @mixin directive and give it a name. We’ve named our mixin theme. We’re also using the variable $theme inside the parentheses so we can pass in a theme of whatever we want. After you create your mixin, you can then use it as a CSS declaration starting with @include followed by the name of the mixin.

Extend / inheritance

Using @extend lets you share a set of CSS properties from one selector to another. In our example we’re going to create a simple series of messaging for errors, warnings and successes using another feature which goes hand in hand with extend, placeholder classes.

Example:

%message-shared

border: 1px solid #ccc

padding: 10px

color: #333

.message

@extend %message-shared

Operators

Doing math in your CSS is very helpful. Sass has a handful of standard math operators like +, -, \*, math.div(), and %. In our example we’re going to do some simple math to calculate widths for an article and aside.

SCSSSassCSS

SASS SYNTAX

@use "sass:math"

.container

display: flex

article[role="main"]

width: math.div(600px, 960px) \* 100%

aside[role="complementary"]

width: math.div(300px, 960px) \* 100%

margin-left: auto