Writing quicker while making better, leaner and more powerful CSS

Presented by wesruv

MY BIAS

- UI/UX Designer and Front End Dev
- Most interested in lean markup and styles
- Degree in Illustration (not Computer Science)
- Been writing CSS for 12 years
- My focus in SASS has been the language itself, I have no plans to mess with command line Ruby and Compiler nuances

OUTLINE

- What is SASS / CSS Compiler?
- Major Features
- Pitfalls & Traps of SASS / CSS Compilers Cool Examples
- How to setup a SASS Project
- SASS Compilers for Designers

WHAT IS IT?

- CSS Compiler
- Use logic, math, variables: make decisions in code
- More re-usable, shareable
- Use Extensions and Libraries without increasing end-user load
- In other words: smarter, better, faster (probably stronger too)

NESTING

```
#primary-nav {
    ul {
        margin: 0;
        padding: 0;
        list-style: none;
    }
    li {
        float: left;
        a {
          display: block;
          padding: 6px 12px;
          text-decoration: none;
        }
    }
}
```

```
#primary-nav ul {
  margin: 0;
  padding: 0;
  list-style: none;
}

#primary-nav li {
  float: left;
}

#primary-nav li a {
  display: block;
  padding: 6px 12px;
  text-decoration: none;
}
```

Don’t be ‘that guy*’

One big gotcha of SASS, overly authoratative selectors

```
body {
    …
    #page {
        …
        #primary-nav {
            …
            ul {
                …
                li {
                    …
                    a {
                      …
                    }
```

Good luck overriding any of this

*or other gender assignment

PRO-TIPS

On getting started

- Start learning SASS, just by rewriting an existing CSS using SASS' nesting
- Use **CodePen.io**, a great sandbox for quick proof of concepts that has SASS and some of it's libraries pre-loaded!
- While you're on CodePen, see what people are doing: **Search for SASS on CodePen**

FILE TYPES

filename**.SCSS**
Looks an awful lot like CSS, but you can write special SASS commands & tricks!

filename**.SASS**
Same as SCSS, except you don't need any { } or ; Instead, this format uses indentation and new lines

filename**.SCSS**

```scss
#primary-nav {
    ul {
        margin: 0;
        padding: 0;
        list-style: none;
    }
    li {
        float: left;
        a {
          display: block;
          padding: 6px 12px;
          text-decoration: none;
        }
    }
}
```

filename**.SASS**

```sass
#primary-nav
    ul
        margin: 0
        padding: 0
        list-style: none
    li
        float: left
        a
          display: block
          padding: 6px 12px
          text-decoration: none
```

filename**.SCSS**

- No conversion required
- More people write CSS / SCSS
- SCSS seems more explicit
- No one likes finding the extra “}” symbol

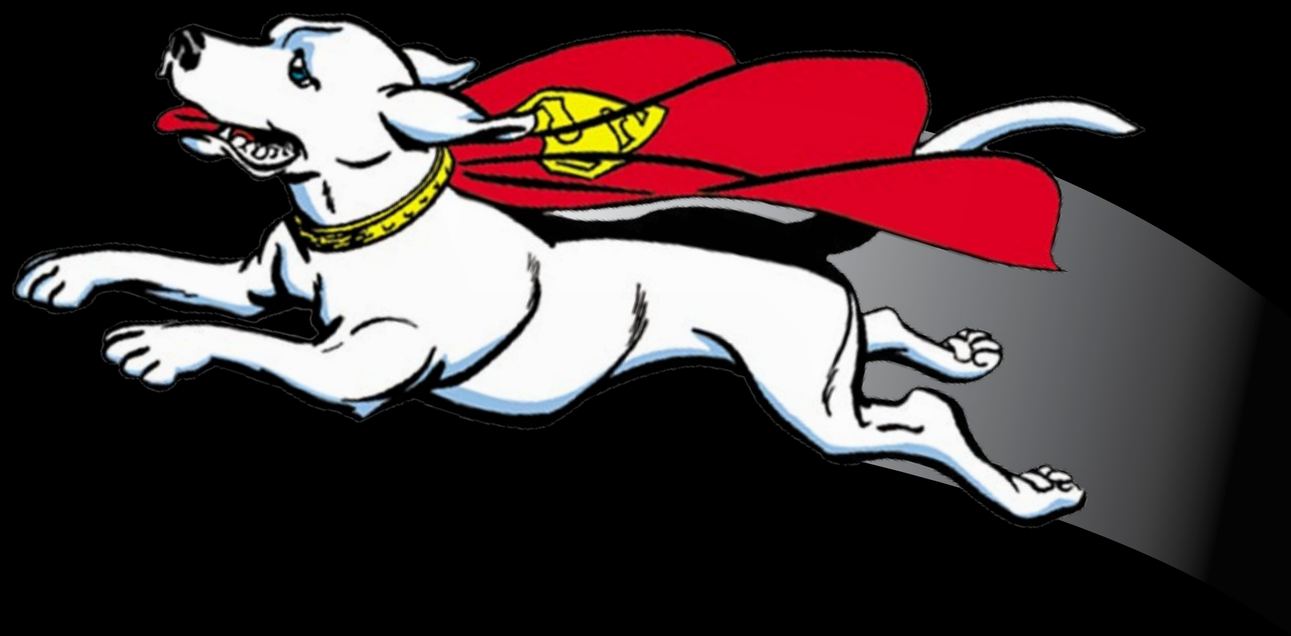
filename**.SASS**

- Seems more efficient
- But forces a certain style of code
- No one likes trying to track down where indentation went wrong

There’s no right or wrong.

.SASS vs .SCSS

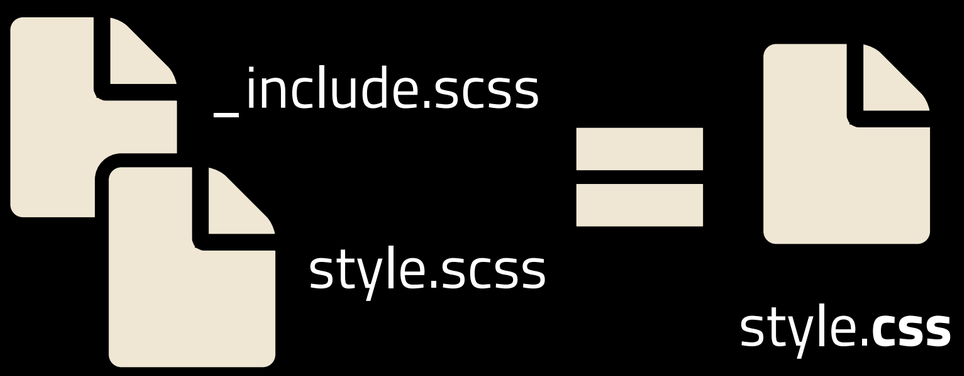
for me it’s SCSS, because



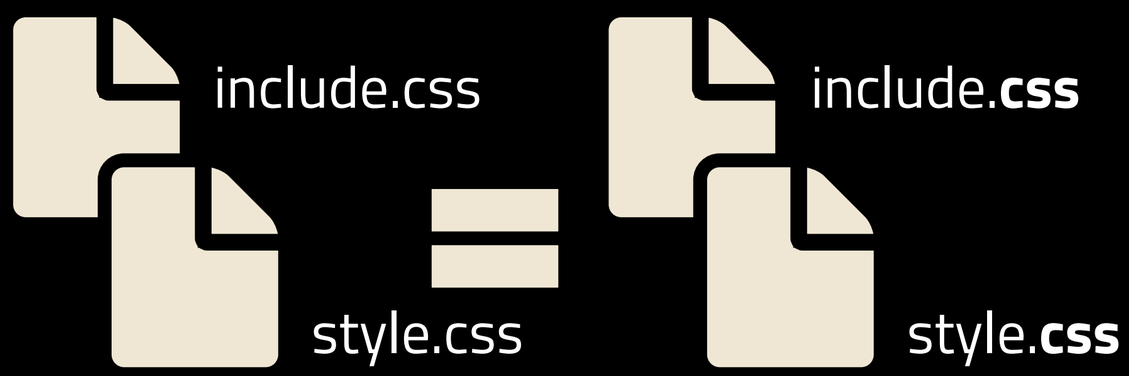
PARTIALS

- It’s like a file type... instead of ending with a different extension a partials begin with ‘_’
- Otherwise they are the same as a .SCSS or .SASS file

Imported partials are put inside of the file(s) they’re called from



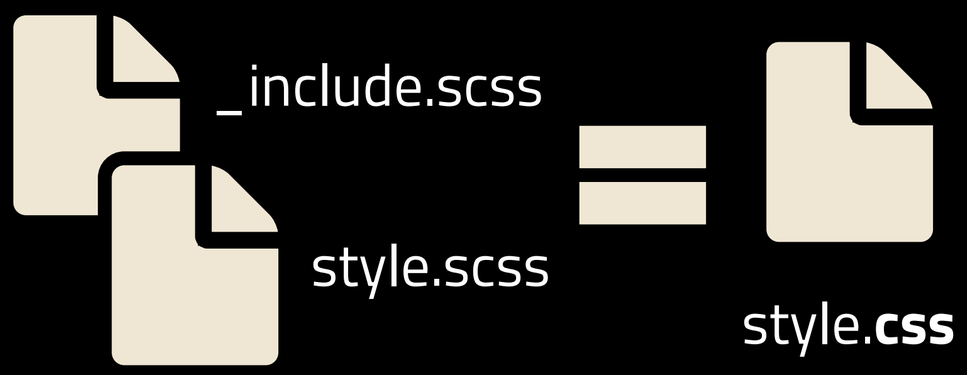
Regular imported files are compiled as separate files



Importing a Partial

```
// Don’t need _ or .scss
@import "include";
```

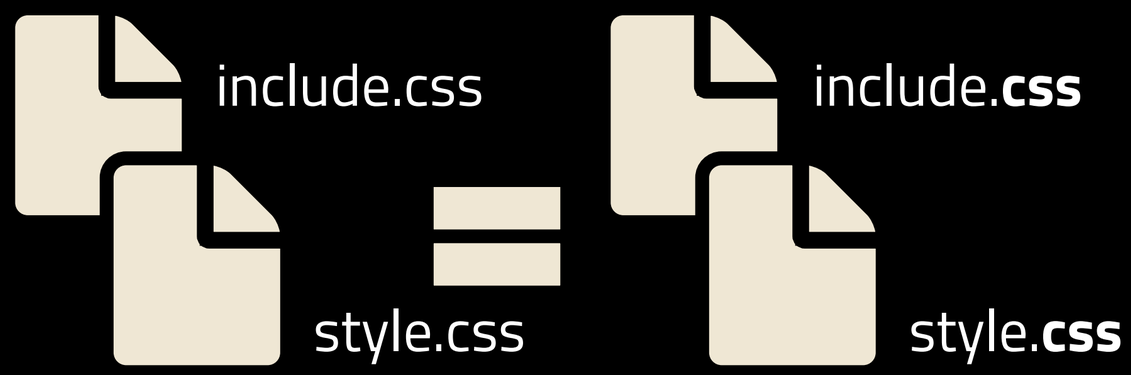
Results in 1 HTTP hequests for end user



Importing CSS

```
// Use @import as usual
@import "include.css";
```

Results in 2 HTTP requests for end user



Partials are great for...

- Re-used variables, functions & mixins
- Keeping Base64 strings out of your working files, which can slow down Code Editors
- Breaking up large code projects into meaningful parts

Here’s where it gets awesome...

MATHS

You can have them!

```
div {
    width: 1in + 8pt;
    width: (240px / 960px) * 100%;
    margin-left: 8px/2px;
    color: #010203 + #040506;
    color: #010203 * 2;
}
```

```
div {
    width: 1.111in;
    width: 25%;
    margin-left: 4px;
    color: #050709;
    color: #020406;
}
```

VARIABLES

- Sick of typing the same color 50 times?

```
$link-blue: rgb(38, 139, 210);
a { color: $link-blue;}
button { background: $link-blue;}
```

- Sick of typing 3 font fall backs for all special elements?

```
$special-font: ‘Titillium Web’, Helvetica, sans-serif;
a { font-family: $special-font;}
button { font-family: $special-font;}
```

Variable Scope

A variable only exists inside the scope it’s created in.

```scss
$link-blue: rgb(38, 139, 210);

a {
    color: $link-blue;
}

button {
    background: $link-blue;
}
```

Works!

```scss
a {
    $link-blue: rgb(38, 139, 2…
    color: $link-blue;
}

button {
    background: $link-blue;
}
```

Compile Error

@EXTEND

DRY CSS!

```scss
.button {
    border: 1px solid $dark-blue;
    border-radius: .8em;
    padding: .25em 1em;
    color: #fff;
    background: $blue;
}

.primary-button {
    @extend .button;
    border-color: $dark-red;
    background: $red;
}
```

```css
.button,
.primary-button {
    border: 1px solid #0B5
    border-radius: .8em;
    padding: .25em 1em;
    color: #fff;
    background: #0D83B5;
}

.primary-button {
    border-color: #7A0D0B;
    background: #C92522;
}
```

An Issue with @extend

```
// Assuming code from
// previous slide
.article a {
    @extend .button;
}

#sidebar .signup .button {
  margin-top: 1.5em;
}
```

```
.button,
.primary-button,
.article a {
    …styles…
}

// then later
#sidebar .signup .button,
#sidebar .signup .primary-button,
#sidebar .signup .article a {
    …styles…
}
```

SWEET!

AAAH!! NO!

But there's a solution!

PLACEHOLDERS

- Uses % instead of # or .
  For example: `%clearfix { … }`
- Explicitly meant for SASS @extends that shouldn’t be compiled as classes in CSS

```
#sidebar {
    @extend %clearfix;
}
```

@MIXIN

- If CSS had functions (like a proper coding language)

```scss
@mixin block-pseudo-element{
    content: ' ';
    display: block;
    position: absolute;
}

.thing:before {
    @include block-pseudo-element();
}
```

```scss
.thing:before {
    content: ' ';
    display: block;
    position: absolute;
}
```

Simple Mixin Example


```
@mixin block-pseudo-element{
    content: ‘ ‘;
    display: block;
    position: absolute;
}

//          MIXIN NAME                  PARAMETER,  PARAM: DEFAULT
@mixin bg-block-pseudo-element($element-bg, $width: 20px,
$height: 20px){
  @include block-pseudo-element;
  width: $width;
  height: $height;
  background: $element-bg;
}
```

Nesting in mixins

```
@mixin add-quote {
    $char-ldquo: '\201C';
    $char-rdquo: '\201D';
    &:before,
    &:after {
      display: inline-block;
    }
    &:before {
        content: $char-ldquo;
    }
    &:after {
        content: $char-rdquo;
    }
}

.quote {
    @include add-quote();
}
```

```
.quote:before,
.quote:after {
    display: inline-block;
}

.quote:before {
    content: '\201C';
}

.quote:after {
    content: '\201D';
}
```

@IF @ELSE

Variables just got a whole lot more useful...

```
@mixin border-pseudo-element($element-bg: null, $width: 10px,
$height: 10px){
  @include block-pseudo-element;
  width: 0;
  height: 0;
  @if $width == $height {
    border: $width solid $element-bg;
  } @else {
    border: $width solid $element-bg;
    border-top-width: $height;
    border-bottom-width: $height;
  }
}
```

@FOR

```scss
//Write 3 elements with increasing
widths
@for $i from 1 through 3 {
  .item-#{$i} { width: 2em * $i; }
}
```

```css
.item-1 {
    width: 2em;
}
.item-2 {
    width: 4em;
}
.item-3 {
    width: 6em;
}
```

#{$un-var-me-dawg}

I guess it’s called interpolation…

```
$property: 'background';

.awesome-thing {
    $property: #000;
}
```

Sets Variable Value

```
.awesome-thing {
}
```

```
$property: ‘background’;

.awesome-thing {
    #{$property}: #000;
}
```

SWEET!

```
.awesome-thing {
    background: #000;
}
```

But wait, There's more!


```
$property: ‘background’;

.awesome-thing {
    #{$property}-color: #000;
    &:after {
        content: 'The #{$property} is #000';
        display: inline-block;
    }
}
```

```
.awesome-thing {
    background-color: #000;
}
.awesome-thing:after {
    content: 'The background is #000';
    display: inline-block;
}
```

@DEBUG

Can’t figure out why something isn’t spitting out as expcted? Send a message to your compiler!

```
@mixin border-pseudo-element($element-bg: null, $width: 10px,
$height: 10px){
    @include block-pseudo-element;
    width: 0;
    height: 0;
    border: $width solid $element-bg;
    border-top-width: $height;
    border-bottom-width: $height;
    @debug ‘height = #{$height}’;
}
```

DEMO O'CLOCK

- Simple Button Demo:
  http://codepen.io/wesruv/pen/zHGIe
- Custom Characters Demo:
  http://codepen.io/wesruv/pen/GvAoi
- Button Maker with Perceptual Brightness
  http://codepen.io/wesruv/pen/gKBnq
- Intersection & Reflection
  http://codepen.io/wesruv/pen/blika

ARRAYS!*

Well, kinda… they’re actually called ‘lists’ and not as full featured.

Unfortunately I *just* found out about lists, and will have to point you to a great article on them:

http://hugogiraudel.com/2013/07/15/understanding-sass-lists/

And documentation on list functions:
http://sass-lang.com/documentation/Sass/Script/Functions.html#list-functions

SOURCES

SASS-lang.com
- SASS Basics http://sass-lang.com/guide
- SASS Reference http://sass-lang.com/documentation
- List Functions http://sass-lang.com/documentation/Sass/Script/Functions.html#list-functions

Mastering SASS Extends and Placeholders
http://8gramgorilla.com/mastering-sass-extends-and-placeholders/

Understanding SASS Lists by Hugo Giraudel
http://hugogiraudel.com/2013/07/15/understanding-sass-lists/