SASS

CSS with superpowers

What is Sass?

Sass (Syntactically Awesome Style Sheets) is a CSS preprocessor, that adds functionality that doesn't exist in CSS yet, like variables, nesting, mixins and inheritance.
This makes writing maintainable CSS easier.
You can get more done, in less code, more readable, in less time.

SASS & SCSS

Sass first came out, the main syntax was noticeably different from CSS. It used indentation instead of braces, didn’t require semicolons and had shorthand operators.

SCSS is a superset of CSS, and is basically written the exact same, but with all the Sass features.

In this slide I will be using the .scss syntax.

Preprocessing

The browser can not read scss or sass file natively.
Once you start tinkering with Sass, it will take your preprocessed Sass file and save it as a normal CSS file that you can use in your web site.

SASS

- Syntactically Awesome StyleSheets
- “Compiles” to CSS
- Introduces programming features to CSS

Setup

To use Sass we need Ruby and Sass installed.

For Mac Ruby installation go to slide 4.

For Windows Ruby installation go to slide 5.

Install Ruby (Mac)

The easiest way to install ruby is using the homebrew package manager.
To do so copy and paste `/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"` into terminal.
After installation finishes, use homebrew to install Ruby by entering `brew install ruby` in the terminal.

Install Ruby (Windows)

To install Ruby on windows download (version 2.2.6) and run the RubyInstaller for Windows.
During installation when prompted, check "add ruby to path".

Install Sass

Install Sass through the command line or terminal by writing `gem install sass`
Check if Sass installed correctly by writing `sass -v` in the command line.

If you do not have permission write `sudo gem install sass`

Create project

For our first introduction we will create a project with three files.

- index.html
- style.css
- style.scss

Variables

Sass brings variables to CSS.
Acceptable values for variables include numbers, strings, colors, null and lists.
Variables in Sass are scoped using the $ symbol.

```
$a: Black;                          //Color
$b: 4px;                            // Units
$c: Helvetica, sans-serif;          //Lists (divided by comma)
$d: 1px #000 Solid 0 0;             //Complex list
body {
    background: $a;                 //Sets body background to black
}
```

Compiling SCSS to CSS

Once you finish tinkering with Sass, we need to take your preprocessed Sass/Scss file and save it as a normal CSS file that you can use in your web site. The most direct way to make this happen is in your terminal. Once Sass is installed, you can run `sass input.scss:output.css` from your terminal.

You can watch either individual files or entire directories with the `--watch flag`.

```
sass --watch input.scss:output.css
```

In addition, you can watch folders or directories.

```
sass --watch input-dir:output-dir
```

SCSS to CSS Example

SCSS

```scss
$font-stack: Helvetica, sans-serif;
$primary-color: #333;

body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

CSS

```css
body {
  font: 100% Helvetica, sans-serif;
  color: #333;
}
```

Operators

Doing math in your CSS is very helpful. Sass has a handful of standard math operators like +, -, *, /, and %.

```
font-size: 4px + 4; //8px
font-size: 20px * .8; //16px

//Parentheses use to define order of operations
width: (100% /2) +25%; //75%
```

Operator Quirks

First, because the / symbol is used in shorthand CSS font properties like `font: 14px/16px`, if you want to use the division operator on non-variable values, you need to wrap them in parentheses like `font: (14px/16px)`

Second, you can't mix value units:

```
$container-width: 100% - 20px;
```

The above example won't work. Instead, for this particular example you could use the css `calc` function, as it needs to be interpreted at render time.

Functions

Sass has many built-in functions and the full list can be seen here.
Some of the best are the color functions

Examples

```
$color : lighten($color, 10%);      //Makes a color lighter
$color : darken($color, 10%);       //Makes a color darker
$color : saturate($color, 10%);     //Makes a color more saturated.
$color : fade_in($color, .1);       //Makes a color more opaque.
$color : fade_out($color, .1);      //Makes a color more transparent.
$color : invert($color) ;           //Returns the inverse of a color.
```

String Interpolation

SCSS

```
// Can use Ruby/PHP style string insertion

$root: "/images/";

#form
{
  background: url("#{$root}background.jpg");
}
```

CSS

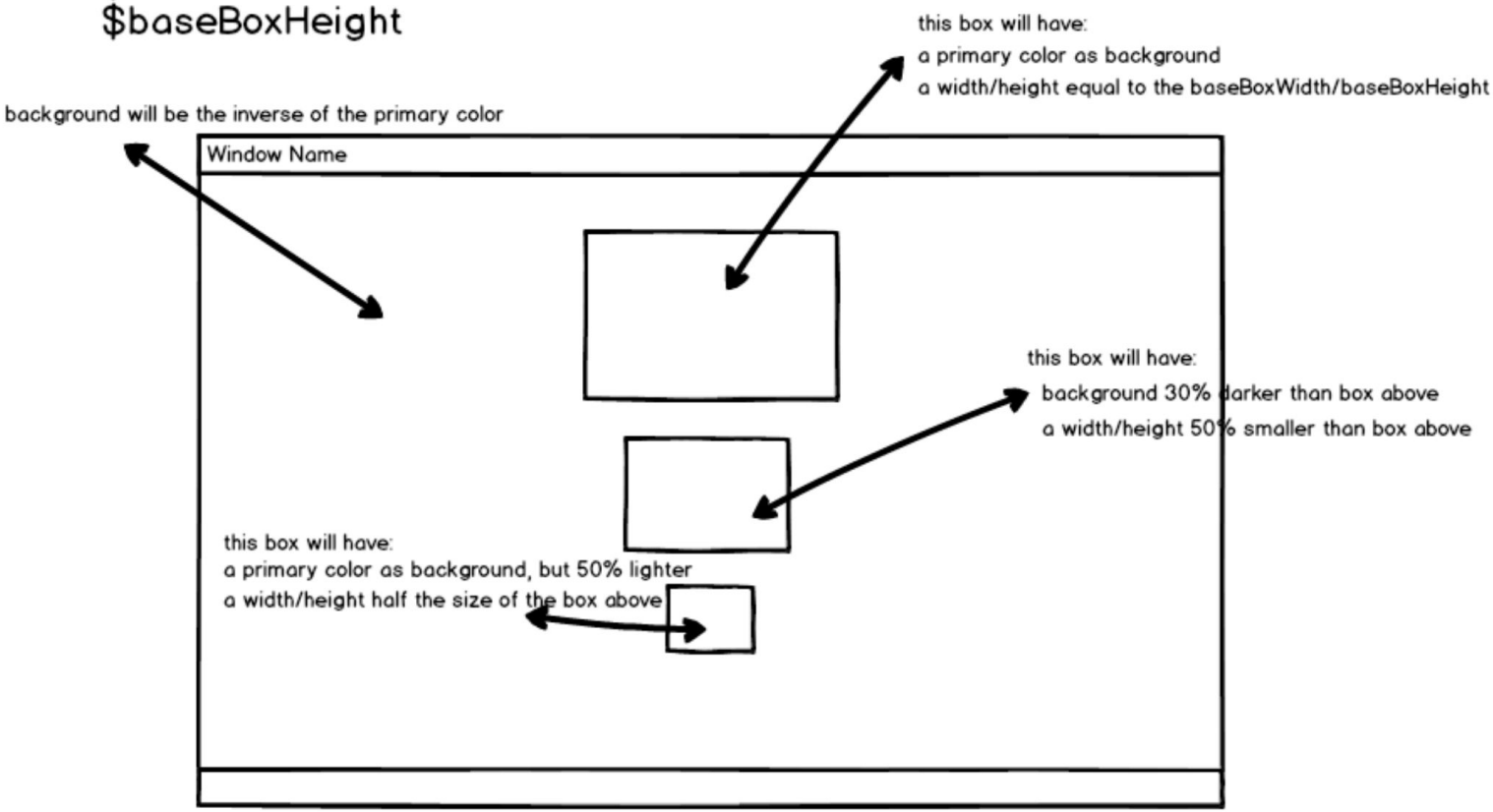
```
#form
{
  background: url("/images/background.jpg");
}
```

Exercise

In pairs, recreate this with sass using vars, functions and operators.

Varibles

$primaryColor
$baseBoxWidth
$baseBoxHeight



Nesting

When writing HTML it has a clear nested and visual hierarchy. CSS, on the other hand, doesn't.
Sass will let you nest your CSS selectors in a way that follows the same visual hierarchy of your HTML.

Be aware that overly nested rules will result in over-qualified CSS that could prove hard to maintain and is generally considered bad practice.

Nesting Examples

SCSS

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

  li { display: inline-block; }

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

CSS

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

nav li {
  display: inline-block;
}

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

Referencing Parent Selectors: &

Sometimes it’s useful to use a nested rule’s parent selector in other ways than the default. For instance, you might want to have special styles for when that selector is hovered over or for when the body element has a certain class. In these cases, you can explicitly specify where the parent selector should be inserted using the **&** character.

& Selector

SCSS

```
a.myAnchor {
    color: blue;
    &:hover {
        text-decoration: underline;
    }
    &:visited {
        color: purple;
    }
}
```

CSS

```
a.myAnchor
{
  color: blue;
}
a.myAnchor:hover
{
    text-decoration: underline;
}
a.myAnchor:visited
{
    color: purple;
}
```

Imports

CSS has an import option that lets you split your CSS into smaller, more maintainable portions.
The only drawback is that each time you use `@import` in CSS it creates another HTTP request.
Sass builds on top of the current CSS `@import` but instead of requiring an HTTP request, Sass will take the file that you want to import and combine it with the file you're importing into so you can serve a single CSS file to the web browser.

```
@import "grids.scss";
```

Extends

Using `@extend` lets you share a set of CSS properties from one selector to another. It helps keep your Sass very DRY.

Extends Example

SCSS

```scss
.message {
  border: 1px solid #ccc;
  padding: 10px;
  color: #333;
}

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

CSS

```css
.message, .error {
  border: 1px solid #cccccc;
  padding: 10px;
  color: #333;
}

.error {
  border-color: red;
}
```

Extends Placeholders

Placeholder selectors look like class and id selectors, except the # or . is replaced by %. They can be used anywhere a class or id could, and on their own they prevent rulesets from being rendered to CSS.
However, placeholder selectors can be extended, just like classes and ids. The extended selectors will be generated, but the base placeholder selector will not.

Extends Placeholder Example

SCSS

```
%input-style {
    font-size: 14px;
}

input {
    @extend %input-style;
    color: black;
}
```

CSS

```
input {
  font-size: 14px; }

input {
  color: black; }
```

Exercise

Work together in pairs.
Fork or clone this git repo, and optimize(DRY) it as much as possible through sass.

Mixins

You can think of mixins as a simplified version of constructor classes in programming languages – you can grab a whole group of CSS declarations and re-use it wherever you want to give and element a specific set of styles.

Mixins

SCSS

```scss
@mixin square($size, $color) {
  width: $size;
  height: $size;
  background-color: $color;
}

.small-blue-square {
  @include square(20px, rgb(0,0,255));
}

.big-red-square {
  @include square(300px, rgb(255,0,0));
}
```

CSS

```css
.small-blue-square {
  width: 20px;
  height: 20px;
  background-color: blue;
}

.big-red-square {
  width: 300px;
  height: 300px;
  background-color: red;
}
```

Passing a list...

Passing a variable list is done with adding … to the parameter.

```scss
SCSS

$box : 20px, red;

@mixin square($size, $color) {
  width: $size;
  height: $size;
  background-color: $color;
}
.small-blue-square {
  @include square($box...);
}
```

Functions

Mixins allow you inject name/value pairs into css rules.
Functions allow you to create functional code.

Example

```
// Value calculations
$app-width: 900px;

@function column-width($cols) {
  @return ($app-width / $cols) - ($cols *
5px);
}
.col2 {
  width: column-width(2);
}
.col3 {
  width: column-width(3);
}
```

If/Else

Allows conditional statement and branching.

```
h1{
     @if $size > 14px{
          color: Blue;
     }
     @else if $size < 14px{
          color: Red;
     }
     @else{
          color: Green;
     }
}
```

For Loop

It’s a loop...

```
$page-width:1000px

@for $col from 1 through 4{
     .col#{$col}{
          width: $page-width / $col;
     }
}
```

While Loop

Looping based on a condition

```
$i: 1;
@while $i < 5{
     h#{$i}{
          font-size: $i*4px;
          $i: $i+1;
     }
}
```

Exercise

Create one badass mixin!

You know this css, figure out a great use case for a mixins or a function.

Wow your fellow students and present in 30.

```
@mixin transform-tilt() {
  $tilt: rotate(15deg);

  -webkit-transform: $tilt; /* Ch <36, Saf 5.1+, iOS, An =<4.4.4 */
      -ms-transform: $tilt; /* IE 9 */
          transform: $tilt; /* IE 10, Fx 16+, Op 12.1+ */
}

.frame:hover {
  @include transform-tilt;
}
```

```
.frame:hover {
  -webkit-transform: rotate(15deg);  /* Ch <36, Saf 5.1+, iOS, An =<4.4.4 */
  -ms-transform: rotate(15deg);  /* IE 9 */
  transform: rotate(15deg);  /* IE 10, Fx 16+, Op 12.1+ */
}
```