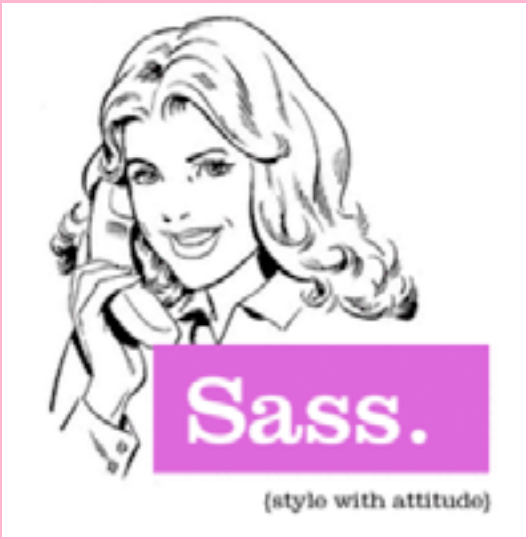

S.yntactically

A.wesome

S.tyle S.heets

CSS Has Problems???

- Large projects yield large amounts of styles
  - Hard to maintain + work on simultaneously
- Multiple stylesheets means multiple server calls
- Repeating yourself for things like colors
  - Accidental duplicate declarations

What is Sass?

- Compiled CSS
- Solves many problems with enterprise-scale styling
- CSS with added utilities, amongst other cool things
- Regular CSS is compatible -- no need to use Sass' features immediately if you're still learning

How can SASS Help?

- Allows for modularizing of CSS in small chunks
  - When compiled, this yields only 1 CSS file

- Utilities for making development easier, such as:
  - Variables
    - Mixins
      - Partials
        - Calculations
          - Imports
            - Nesting

Nesting

- Groups similar chunks of CSS together
- Easier to keep track of*
- Less writing for the developer to do

*Do not do not *do not* **do not *do not*** try to recreate DOM structure -- Nesting is SASS' biggest pitfall.

Nesting

```
nav #homeIcon{
    width: 100px;
    height: 100px;
    float: left;
}
nav .navText{
    color: #FFCCCC;
    font-size: 12px;
}
nav .navText a:hover{
    color: #997A7A;
}
nav .navText a:visited{
    color: #FFA3C2;
}
nav .navText a:active{
    color: #997A7A;
}
nav .navListItems {
    display: inline-block;
}
nav .navListItems .navListItems-large{
    width: 200px;
}

```

```
nav{
    #homeIcon:{
        width: 100px;
        height: 100px;
        float: left;
    }
    .navText{
        color: #FFCCCC;
        font-size: 12px;
        a{
            &:hover{
                color: #997A7A;
            }
            &:active{
                color: #997A7A;
            }
            &:visited{
                color: #FFA3C2;
            }
        }
    }
    .navListItem{
        display: inline-block;
        .navListItem-large{
            width: 200px;
        }
    }
}
```

Partials

- Breaks CSS into smaller files based on modules or functionality

- Are signified by an underscore at the start of the file-name
  - ex: _colors.scss

- Are **not** compiled

- Added to a centralized file via imports

Imports

- Bring outside files in
- Ideally, all of your imports go in 1 centralized file
- Syntax is just @import ‘filename’;
- Leave off “.scss” and “_” before name: Sass knows what to look for
- Do not import into partials: this will cause circular dependencies.

Imports + Partials

_example2_nav.scss

```scss
nav{
    float: left;
    width: 500px;
    height: 120px;
    background-color: #000080;
    #branding{
        background: #fff url('../images/logo.png') no-repeat right top;
        float: left;
    }
}
```

example2_main.scss

```scss
@import 'example2_nav';
@import 'example2_document';
@import 'example2_table';

body{
    width: 100%;
}
h2{
    color: pink;
}
```

_example2_document.scss

```scss
.document{
    font-size: 13px;
    overflow: hidden;
    text-overflow: ellipsis;
    width: 800px;
    height: 300px;
}
```

_example2_table.scss

```scss
.orderTable{
    border:2px dashed #3a4b3d;
    width: 100%;
    .colorHeader{
        color: #00ff00;
    }
}
```

Variables

- Act the same as variables in JavaScript
- Scope is similar to JS'--can be defined in a declaration or globally in a file
  - Imported files also import their global variables
- Syntax: $variableName: value;

Variables

_colors.scss

```scss
$color-accent-negative:     #D26420;
$color-accent-positive:     #00723D;
$color-accent-strong:       #00B8E6;
$color-accent-weak:         #A5E4FF;
```

overview.scss

```scss
@import 'colors';
.overview{
    .error-container{
        color: $color-accent-negative;
        border-color: $color-accent-strong;
    }
    .success-container{
        color: $color-accent-positive;
        border-color: $color-accent-weak;
    }
}
```

Mixins

- Essentially, they're like CSS functions
- Pass some specifications to a mixin and it returns a calculated result for you
- Most often used for vendor-specific styles (gradient, box-shadow, rounded corners, etc)

Mixin Syntax

Declaring a mixin:

```
@mixin mixinName($variables){
    ...
}
```

Using a mixin:

```
.classname{
    @include mixinName(value);
}
```

Mixins

_mixins.scss

```scss
@mixin border-radius($radius){
    -webkit-border-radius: $radius;
    -moz-border-radius: $radius;
    -ms-border-radius: $radius;
    -o-border-radius: $radius;
    border-radius: $radius;
}
@mixin textShadow($opacity, $offset){
    text-shadow: white($opacity) 2px $offset 2px;
}
@mixin opacity($opacity){
    opacity: $opacity;
    $opacity-ie: $opacity * 100;
    filter: alpha(opacity=$opacity-ie);
}
```

button.scss

```scss

@import 'mixins';

.fancyButton{
    @include opacity(0.93);
    @include border-radius(4);
    @include textShadow(0.5, 1);
}
.ghastlyButton{
    @include opacity(0.4);
    @include border-radius(2);
    @include textShadow(1, 2);
}
.circularButton{
    @include border-radius(500);
}
```

Installing Sass

1- Install Ruby: http://www.rubyinstaller.org/
(Macs already have this installed!)

2- Install Sass: http://sass-lang.com/install

Teaching Sublime Sass

1- Install Package Control:
https://sublime.wbond.net/installation#st3

2- Make it Sassy: https://sublime.wbond.net/packag-es/SassBeautify

Exercise

See the other PDF file for instructions for trying out Sass. You will be making a very simple static page, but will be using the concepts covered in this lesson.