Styling websites faster

An introduction to Sass

Getting started

Run the following commands in your terminal:

```
git clone https://github.com/evantarrh/learn-sass.git
```

```
cd learn-sass
```

```
sudo gem install sass
```

Getting started

Then, run:

```
open index.html
```

There’s already CSS here! So what are we doing here?

THE PROBLEM

CSS: Problems

CSS is outdated and underpowered. It was never designed to work well with large-scale projects.

CSS: Problems

1. No variables
    a. lots of rewriting
    b. difficult to change things, e.g. a theme color

CSS: Problems

1. No variables
   a. lots of rewriting
   b. difficult to change things, e.g. a theme color
2. No way to separate concerns

CSS: Problems

1. No variables
   a. lots of rewriting
   b. difficult to change things, e.g. a theme color
2. No way to separate concerns
3. Selectors (e.g. `section .hero > p > span .code .css`) can get complicated and caterpillar-y

CSS: Problems

1. No variables
    a. lots of rewriting
    b. difficult to change things, e.g. a theme color
2. No way to separate concerns
3. Selectors (e.g. `section .hero > p > span .code .css`) can get complicated and caterpillar-y
4. General lack of features :(

THE SOLUTION

Sass: Magic CSS

1. Variables
2. Separation of concerns
3. Nesting
4. Features

Sass: Magic CSS

1. Variables
2. Separation of concerns
3. Nesting
4. Features
5. Cool factor □

Making Sass work

It’s easy: just start with CSS

Run the following commands in your terminal, from within the `learn-sass` directory:

```
mkdir scss

cp css/index.css scss/index.scss
```

It’s easy: just start with CSS

Run the following commands in your terminal, from within the `learn-sass` directory:

```
mkdir scss
```

```
cp css/index.css scss/index.scss
```

Now, we have two folders, one of which contains `index.css` and the other of which contains `index.scss`.

Compiling Sass

1. Once you’ve changed something in `index.scss`, run this from the `learn-sass` directory:

```
sass scss/index.scss css/index.css
```

This compiles the SCSS from your `scss` directory into `css/index.css`.

Establish a workflow

It’s annoying to have to run that every time you want to change something, though! Fortunately, there exists a perfect solution:

```
sass --watch scss:css
```

Writing Sass

Using variables

Variables should be declared at the top of scss files like so:

```
$gray: #ccc;
```

Variables will always start with a dollar sign.

Using variables

Once you’ve declared the variable, you can use it throughout all scss files.

```
$gray: #ccc;

h1 {
   color: $gray;
}
```

Variables: Benefits

1. Easy to change

```
$gray: #c6c6c6;
```

2. Semantic

```
$light-gray: #eee;
$gray: #aaa;
$dark-gray: #444;
```

Nesting

Wouldn’t it make sense if you could write CSS with a similar structure to your HTML?

```
section {
    background-color: $gray;

    p {
        font-size: 18px;
    }
}
```

Imports

Create a new file in the scss directory, called `_colors.scss`. Inside it, write:

```
$gray: #b0b;
```

Imports

Back in index.scss, delete the “$gray: #ccc” line and replace it with:

```
@import ‘colors’;
```

Imports

Here’s what our two `.scss` files should look like:

`index.scss`

```
@import ‘colors’;

html, body {
    font-size: 18px;
}

...
```

`_colors.scss`

```
$gray: #b0b;
```

Imports

Sass is smart, and any `import` statements will make it search for a .scss file in the same directory that begins with an underscore.

So, the code we wrote automatically incorporates the `_colors.scss` file into the output. Neat!

Imports

But why stop there? Let’s create a `_typography.scss` that will take care of everything font-related.

Imports

Do you see what's happening? Our `index.scss` file is getting smaller and smaller.

An Exercise

Let’s take a few minutes and work to make our `index.scss` look like this (and nothing else!):

```
index.scss

@import ‘colors’;
@import ‘typography’;
@import ‘main’;
```

Reminder—this command will help!

```
sass --watch scss:css
```

Imports: major 🔑

Order of operations matters. See what happens if you put `@import ‘colors’;` at the bottom of your imports.

Responsive Sass

Media queries

One of the best things about separating concerns is that it makes responsive design much easier to pull off. Let’s start by creating a new **module**, `_media_queries.scss`.

Media Queries

```
_media_queries.scss
```

```
$phone-landscape-max: 640px;
$tablet-landscape-max: 1024px;
```

This can be way more detailed for some projects, but this is all we’ll need right now.

Media queries in context

_responsive.scss

```
@media screen and (max-width: $tablet-landscape-max) {
  html, body {
    font-size: 16px;
  }
}

@media screen and (max-width: $phone-landscape-max) {
  html, body {
    font-size: 14px;
  }
}
```

Imports

index.scss

```
@import ‘colors’;
@import ‘typography’;

h1 {
  border-bottom: 1px solid $gray;
  margin-bottom: 1rem;
  margin-top: 5rem;
  padding-bottom: 1rem;
}

h2 {
  padding-bottom: 0.5rem;
  margin-bottom: 2rem;
}
...
```

_typography.scss

```
$sans: Roboto Condensed;
$display: Playfair Display;

html, body {
    font-size: 18px;
}

h1 {
    font-family: $display;
    font-size: 4rem;
    line-height: 4rem;
}

h2 {
    ...
```

Using media queries for good

Now our `index.scss` should look like this:

index.scss

```
@import ‘colors’;
@import ‘typography’;
@import ‘main’;
@import ‘media_queries’;
@import ‘responsive’;
```

Bonus

Mixins

```
@mixin border-radius($radius) {
  -webkit-border-radius: $radius;
     -moz-border-radius: $radius;
      -ms-border-radius: $radius;
          border-radius: $radius;
}

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

Functions

```
a:hover {
    color: lighten($link-blue, 20%);
}

h2 > a {
    color: saturate($link-blue, 40%);
}
```

Math

```
.content {
    width: (900 * 1.5) - 100px;
    max-width: $landscape-portrait-max - 250px;
}
```

Thanks!

@evantarrh