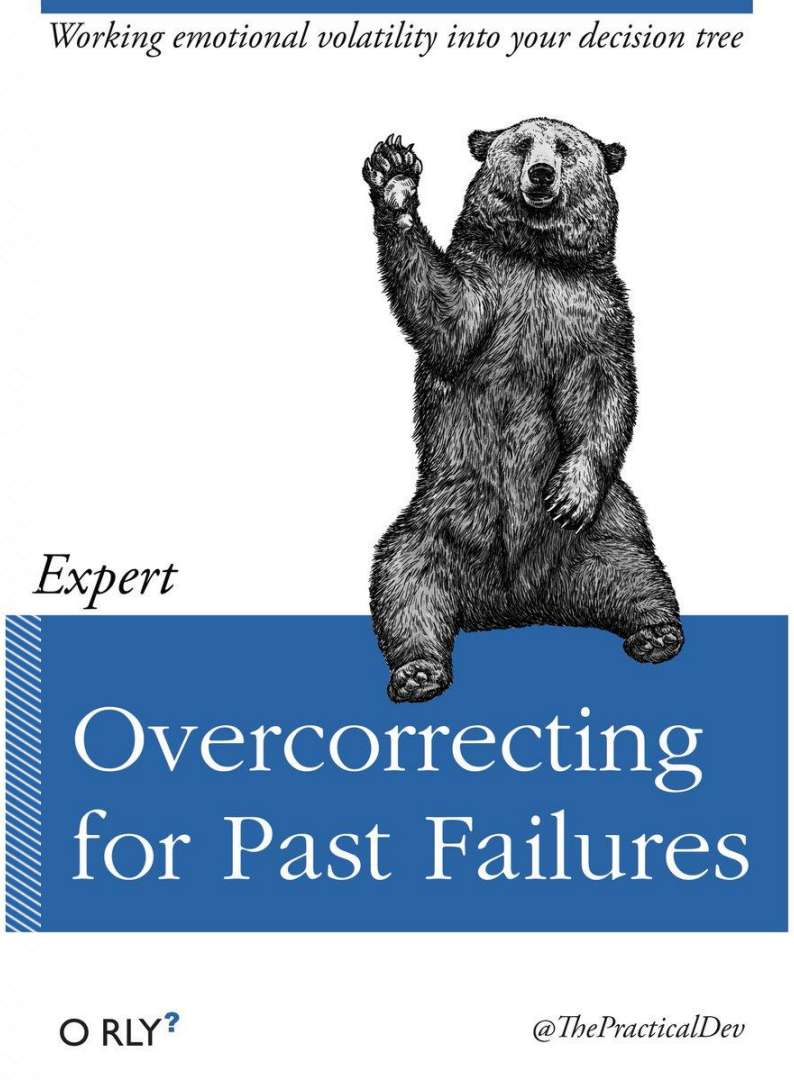
Pre & Post Processing CSS

(how to reach infinite nesting level)

Frontend Junior Program - 2022



Agenda

1. CSS Preprocessors
2. Sass, Less, Stylus
3. Sass installation and usage
4. Sass in details
5. CSS Postprocessors

CSS PREPROCESSORS

Introduction to CSS preprocessing

A CSS preprocessor is a program that lets you generate CSS from the preprocessor's own unique syntax.

Write your code

.scss, .sass, .less

Compile it

with command line
or build tools / bundlers

Done!

.css

Pros & Cons

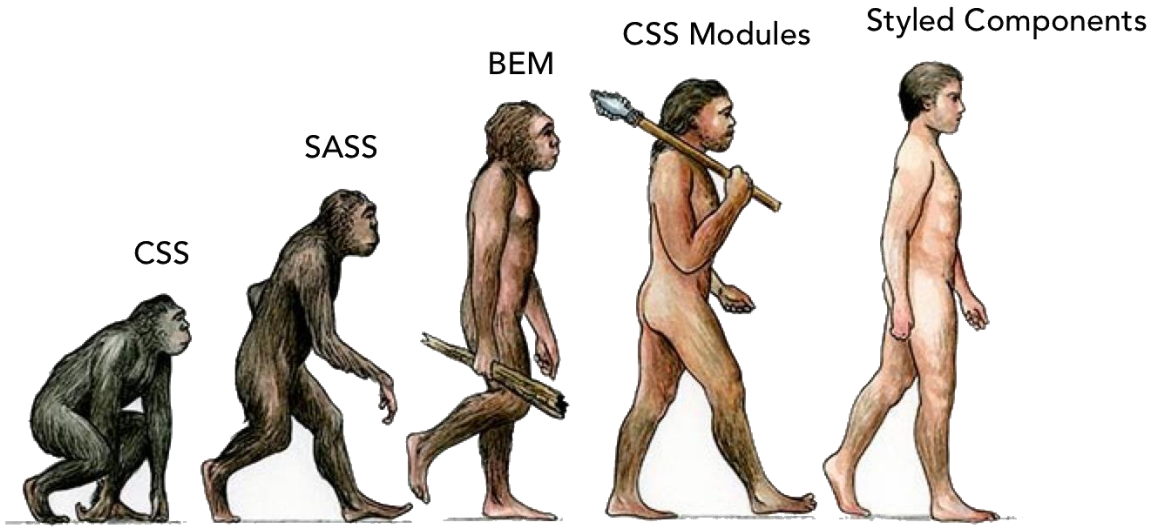
Pros

- Nested
- Modularized
- Powerful expressions
- Reduced redundancy (@mixin, @function)
- Code reuse across multiple projects
- Works wonderfully with BEM

Cons

- Not a native syntax for browsers – needs to be compiled
- Added complexity
- Unusual syntax (.sass, Stylus)
- Error prone patterns (@extend)
- In the era of components (React, Angular, Web Components) different approaches can be more suitable, such as CSS Modules and Styled components

Evolution



Obviously, it is a joke – still, it reflects well on how technologies need to change according to the changes of the environment. It is an evolution - by definition - , however, it does not mean that the new iterations are better or easier to work with.

Why bother?

Necessity

- Preprocessors are still used and an important part of the web technology stack.

Wide view and understanding

- Need to understand the problems it solves, and the motivation behind it.

Focus

- Tools do not matter, focus on fundamentals: simplicity, modularity and scalability.

Is Sass Worth Learning In 2020?

SASS, LESS, STYLUS

Sass, Less, Stylus

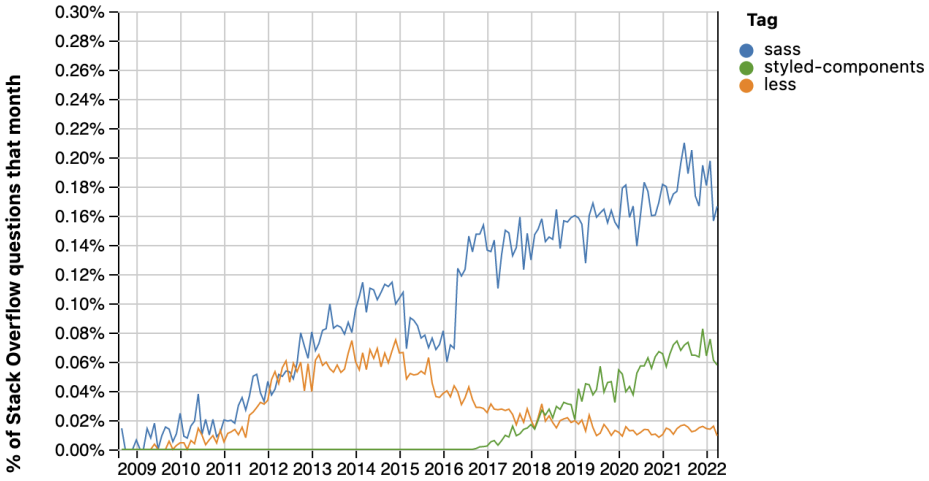
"Sass is the most mature, stable, and powerful professional grade CSS extension language in the world."

It's CSS, with just a little more.

Stylus is an innovative stylesheet language that compiles down to CSS.

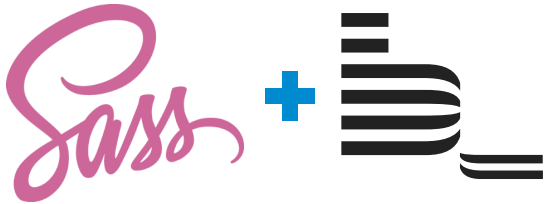
What to learn?

- Robust: Mature, battle-hardened
- Useful: Feature complete (enough)
- Future proof: Largest market share
- As a consequence: Predominantly used in EPAM

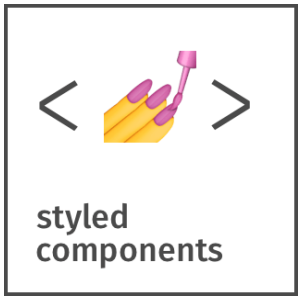


Stack Overflow Trends

Learning path



Sass + BEM



Common parts

Variables

Nesting

Import

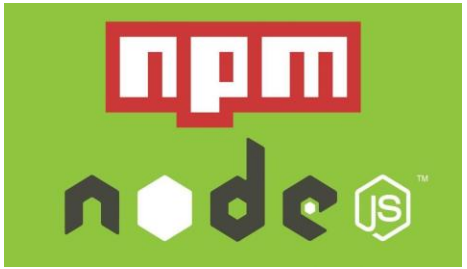
Mixins

Extend / Inheritance

Operators

SASS INSTALLATION AND USAGE

Sass installation - prerequisites



npm (Node Package Manager)

A package manager for JavaScript.

npm is distributed with Node.js- which means that when you download Node.js, you automatically get npm installed on your computer.

Node.js

A JavaScript runtime, that can be used to run js code without a browser.

Download and install node

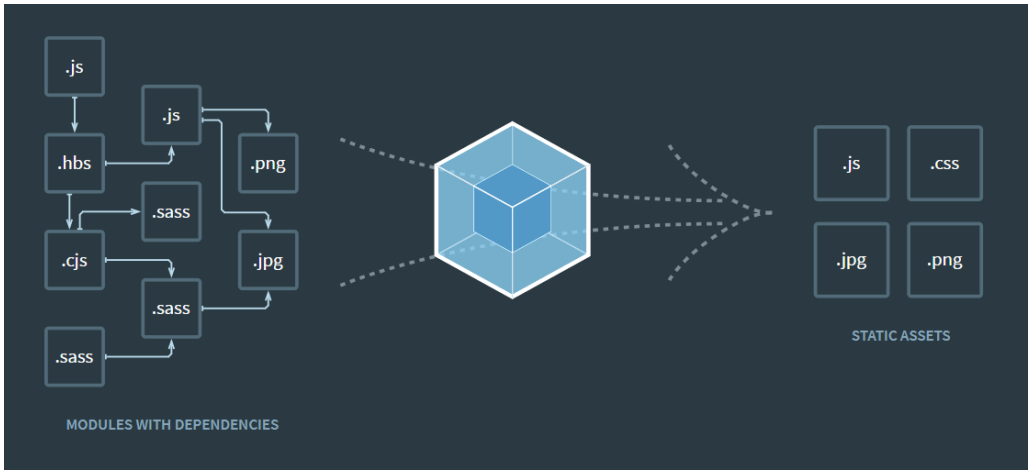
Sass installation – prerequisites (optional)

Webpack

It's all about automation.

The less work you have to do when performing repetitive tasks, such as **compiling, building**, **running unit tests and linters**, etc., the easier your job becomes.

Webpack is a robust module bundler with countless plugins- making it suitable for any project.



Sass installation

We mark each command line example always with a '>' symbol at the beginning. The actual command starts after the symbol.

Make sure you don't include it in your commands!

Standalone version

You can install Sass on Windows, Mac, or Linux by downloading the package for your operating system from GitHub and adding it to your PATH. That's all—there are no external dependencies and nothing else you need to install.

Via npm

If you use Node.js, you can also install Sass using npm by running

```
> npm install -g sass
```

With Webpack

```
> npm install sass-loader sass webpack --save-dev
```

At this point, it is preferred to install with npm.
In future, you will probably use it with Webpack.

Sass: Install Sass (sass-lang.com)

Sass usage

```
# compile input.scss to output.css
$ sass input.scss output.css

# compile all *.scss from a folder and watch for changes
$ sass --watch app/scss:public/css
```

The watch flag tells Sass to watch your source files for changes, and re-compile CSS each time you save your Sass.

Sass would watch all files in the app/scss folder for changes, and compile CSS to the public/css folder.

SASS IN DETAILS

Playground

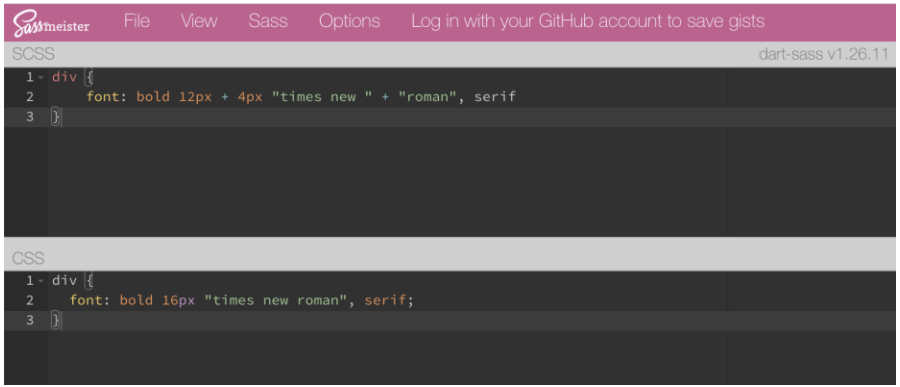
SassMeister is a playground for Sass.

Add some Sass and SassMeister will show you the rendered CSS.

SassMeister supports

- *both Sass and SCSS syntaxes*
- *all output styles, and*
- *an ever-expanding list of Sass libraries.*

https://www.sassmeister.com/

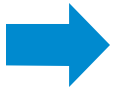


Variables

.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;
}
```

Sass variables are prefixed with the $ symbol, otherwise the syntax similar to a CSS property.

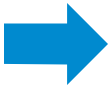
Sass: Variables

Operators

.scss

```
article {
    float: left;
    width: 600px / 960px * 100%;
}

div {
    font: bold 12px + 4px "times new " + "roman", serif
}
```



.css

```
article {
  float: left;
  width: 62.5%;
}

div {
  font: bold 16px "times new roman", serif;
}
```

- == and != are used to check if two values are the same.
- +, -, *, /, and % have their usual mathematical meaning for numbers, with special behaviors for units that matches the use of units in scientific math.
- <, <=, >, and >= check whether two numbers are greater or less than one another.
- and, or, and not have the usual boolean behavior. Sass considers every value “true” except for false and null.
- +, -, and / can be used to concatenate strings.

Sass: Operators

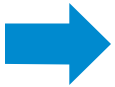
Nesting

.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;
}
```

Sass: Nesting

Remember, nesting could mean inheritance.

Inheritance means that the code depends on other code parts, therefore, it is less reusable, more error prone and less future proof.

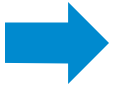
Basically, we try to avoid the Cascading part of Cascading Style Sheets.

In this case the component does not depend on where it is placed in DOM.

Parent selector

.SCSS

```scss
.alert {
  // The parent selector can be used to add pseudo-classes to the outer selector.
  &:hover {
    font-weight: bold;
  }
  // It can also be used to style the outer selector in a certain context,
  // such as a body set to use a right-to-left language.
  [dir=rtl] & {
    margin-left: 0;
    margin-right: 10px;
  }
  // You can even use it as an argument to pseudo-class selectors.
  :not(&) {
    opacity: 0.8;
  }
}
```



.CSS

```css
.alert:hover {
  font-weight: bold;
}

[dir=rtl] .alert {
  margin-left: 0;
  margin-right: 10px;
}

:not(.alert) {
  opacity: 0.8;
}
```

The & symbol references the parent selector.

When a parent selector is used in an inner selector, it is replaced with the corresponding outer selector. This happens instead of the normal nesting behavior.

Sass: Parent selector

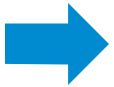
Selector combinators

.scss

```scss
ul > {
  li {
    list-style-type: none;
  }
}

h2 {
  + p {
    border-top: 1px solid gray;
  }
}

p {
  ~ {
    span {
      opacity: 0.8;
    }
  }
}
```



.css

```css
ul > li {
  list-style-type: none;
}

h2 + p {
  border-top: 1px solid gray;
}

p ~ span {
  opacity: 0.8;
}
```

You can nest selectors that use combinators as well. You can put the combinator at the end of the outer selector, at the beginning of the inner selector, or even all on its own in between the two.

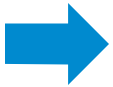
Sass: Selector combinators

Nesting, adding suffixes – a BEM example

.SCSS

```scss
.block {
  &__element {
    background: rebeccapurple;

    &--modifier {
      background: white;
      border-color: lightgray;
    }
  }
}
```



.CSS

```css
.block__element {
  background: rebeccapurple;
}

.block__element--modifier {
  background: white;
  border-color: lightgray;
}
```

You can also use the parent selector to add extra suffixes to the outer selector. This is particularly useful when using a methodology like BEM that uses highly structured class names.

As long as the outer selector ends with an alphanumeric name (like class, ID, and element selectors), you can use the parent selector to append additional text.

No inheritance… *makes the elements be dependent on the block only. So, you can easily move them across the block when providing changes to the interface.*

The changes of the block DOM structure would not need corresponding changes to the CSS code.

Sass: Adding suffixes

@mixin

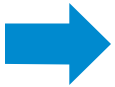
.scss

```scss
@mixin reset-list {
  margin: 0;
  padding: 0;
  list-style: none;
}

@mixin horizontal-list {
  @include reset-list;

  li {
    display: inline-block;
    margin: {
      left: -2px;
    }
  }
}

nav ul {
   @include horizontal-list;
}
```



.css

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

nav ul li {
  display: inline-block;
  margin-left: -2px;
}
```

Mixins allow you to define styles that can be re-used throughout your stylesheet. They make it easy to avoid using non-semantic classes like .float-left, and to distribute collections of styles in libraries.

Sass: @mixin and @include

@extend and inheritance

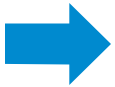
.scss

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

.success {
    @extend .message;
    border-color: green;
}

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

.warning {
    @extend .message;
    border-color: yellow;
}
```



.css

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

.success {
  border-color: green;
}

.error {
  border-color: red;
}

.warning {
  border-color: yellow;
}
```

Sass: @extend

Seriously: Don't use @extend!

.scss

```scss
h4 {
  color: #c09;
  font-size: 1.2rem;
  font-weight: 100;
}

label {
  @extend h4;
  cursor: pointer;
}

.header h4 {
  font-weight: bold;
}

.footer h4 {
  display: inline;
}
```

.css

```css
h4, label {
  color: #c09;
  font-size: 1.2rem;
  font-weight: 100;
}

label {
  cursor: pointer;
}

.header h4, .header label {
  font-weight: bold;
}

.footer h4, .footer label {
  display: inline;
}
```

[Don't use extend!]

Remember!

Don't use extend!

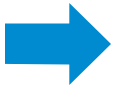
Placeholder selectors

.scss

```scss
%message {
    padding: 10px;
    color  : #333;
}

.success {
    @extend %message;
    border-color: green;
}

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



.css

```css
.success,
.error {
  padding: 10px;
  color  : #333;
}

.success {
  border-color: green;
}

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

Sass has a special kind of selector known as a “placeholder”. It looks and acts a lot like a class selector, but it starts with a % and it's not included in the CSS output.

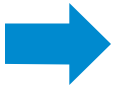
Sass: Placeholder Selectors

@each

.scss

```scss
$sizes: 40px, 50px, 80px;

@each $size in $sizes {
    .icon-#{$size} {
        height: $size;
        width: $size;
    }
}
```



.css

```css
.icon-40px {
  height: 40px;
  width: 40px;
}

.icon-50px {
  height: 50px;
  width: 50px;
}

.icon-80px {
  height: 80px;
  width: 80px;
}
```

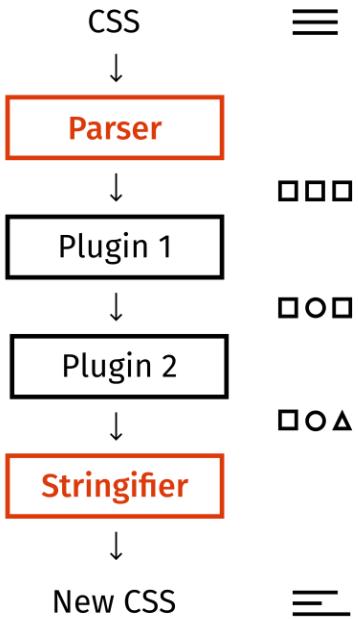
The @each rule makes it easy to emit styles or evaluate code for each element of a list or each pair in a map.

It's great for repetitive styles that only have a few variations between them.

Sass: @each

CSS POSTPROCESSORS

CSS Postprocessors



A post-processor applies changes to a CSS file after it's been hand-coded or generated by a pre-processor.

Use post-processors to tweak it and improve it.

Get more out of our CSS than we could do it by ourselves.

Deconfusing Pre- and Post-processing

PostCSS…

- *is **NOT** a style preprocessor like Sass or Less.*

  It does not define a custom syntax and semantics, it's not actually a language.

  PostCSS works with CSS and can be easily integrated with the tools described above. That being said, any valid CSS can be processed by PostCSS.

- *is a tool for CSS syntax transformations.*

  It allows you to define custom CSS like syntax that could be understandable and transformed by plugins.

PostCSS

edu_hu@epam.com