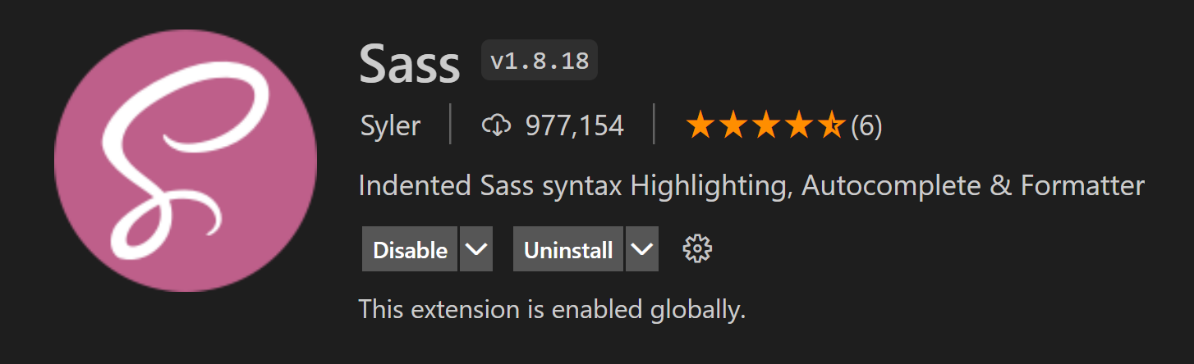
Sass

* It's a pre-processor language.
* We don't have loops, functions, repeat code, keeping files separated, or export-import in CSS. Variables have been added recently to CSS, but it's not as practical as programming languages.
* Sass has all of these. Brower does not run sass files with its specific syntax. It is only used in **development**. In the end, with a command, the sass compiler compiles sass files to CSS files.

**Why Sass makes CSS coding easier?**

* Bootstrap has used Sass to develop its styles for better management and fewer duplicates. Since Bootstrap is open source, you can use Sass to develop it.
* **Sass gives us some functionalities that make development easier. F**or example, teamwork is very practical.

**Extensions**



**comments**

* **/\* \*/** // using this the codes will be compiled and included in the target CSS
* **//**  // using this the codes will not be included

How to install and run Sass

* To use Sass we should have a sass compiler to convert sass and scss files to CSS files.
* To install Sass you should have npm, and for npm, you should have nodejs.

npm i sass -g // with global we can access sass in all of our directories

React:

* npm install node-sass

**Compile Sass to CSS**

* sass –watch sassFileAddress targetAddress // Change the directory to your project. Run this command in cmd. If the –watch flag is set, it watches over changes while the shell is active, so you don't need to run this command every time.

sass styles/sass/style.sass styles/css/main.css

**Sass and CSS**

* two extensions (file format) are used for Sass:
  + sass // there is no need for blocks { } and semicolons ;

body

Background-color : red;

* + scss // the syntax is more similar to CSS, making it less confusing while you have sass syntax.

body {

Background-color : red;

}

* Gulp and webpack.config.js for automation

Nesting

* in CSS we may use one class to select many child elements

ul.list-1{

background: rgb(224,224,224);

}

ul.list-1 li{

border: 1px solid red;

}

ul.list-1 li a.link{

font-size: 1.5rem;

}

* In Sass, you can only use the parent element once and inside its block, select other child elements:

ul.list-1{

background: rgb(224,224,224);

**> li**{ // > says direct child

border: 1px solid red;

// ; **the semicolon** should be here or a.link will not work

**a.link**{

font-size: 1.5rem;

}

}

}

* @media query

ul.list-1{

background: rgb(224,224,224);

> li{ // > says direct child

border: 1px solid red;

a.link{

font-size: 1.5rem;

}

}

**@media only screen and (min-width: 480px)**{

Background: pink;

}

}

Variables

* Variables // containers. That holds values. To be reusable.
* $var : value; // naming rules: numbers and char should not be used

**$**black: #000000;

$text-normal: 1rem;

a.link{

font-size: $text-big;

font-color: $black;

}

* $list : 1,2,3,4,5
* .#{$var}btn // Variables for class names:

$my-name: mycss--;

.#{$my-name}btn {

/\* ... \*/

}

Separate style files and module

* \_filename.scss // \_ tells the Sass that this file will not be a separate CSS file, but the content will be used in another sass file.
* As a convention, Sass files that are only meant to be imported, not compiled on their own, begin with \_ (as in \_code.scss). These are called partials, and they tell Sass tools not to try to compile those files on their own. You can leave off the \_ when importing a partial.

@import "filename";

* With this method, you can import styles and variables.

modules

* When you have the same variables in two different import files, the last one has a higher priority and is affected.
* @use //To use variables with the same name when imported from different files, by this @import rule, you specify the filename: filename.$Varaiable

@import "myvar";

@import "mixins";

li{

border: 1px solid myvar.$black;

color: mixins.$black

}

* This is called modularizing sass files.

Operators

* + , - , \* , / // you can write units or not, but the **units should be the same**: rem + rem. For example, this is wrong: rem + px

@import "myvar";

@import "mixins";

a.bold{

font-size: mixin.fontSize + .2;

width: 500px / 2px

}

* (var / 2) // divide char is used in CSS for example in font shorthand font: font size/ font-height. So use () if you want it to be treated as an operator

inheritance

* **@extend** %className or. className // Sometimes we have duplicate styles,

// styles that are shared between two other classes

**%mes**{

styles

}

.col{ // in css this will be .col, .warn { styles }

styles

}

.warn{

**@extend %mes; // in final css all the extended of this class will be a selector of the style given**

**@extend .col; // this adds .warn as a selector where .col in defined: .col, .warn { }**

Other styles

}

.note{

@extend %mes;

Other unique styles

}

* In output CSS, we will have:

**// For @extend %mes**

.warn, .note {

styles

}

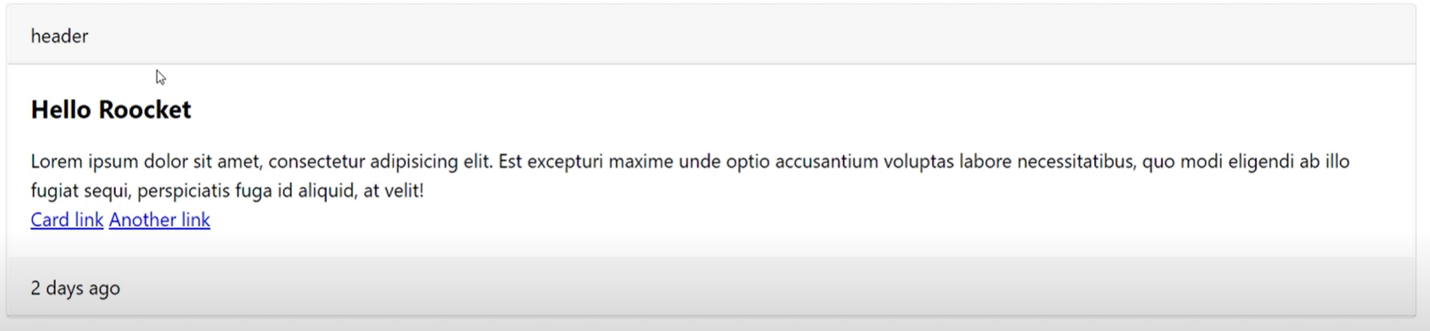
**// For @extend .col**

.col, .warn, .note {

styles

}

Card exercise



* Use one style.scss that imports
  + \_card.scss
  + \_global.scss // like the container and the font family
  + \_reset.scss
  + \_variavbles.scss
* The order of imports is important // if you use some of the codes in one import in another, it should come before that.

mixins

* @mixins name (){} // mixins help to reduce the number of codes we write (in development). Instead of using properties directly, by mixins, you can use a function to set that property.
* #{$var}: // to use variables in property name

**@mixin** border($side , $value) {

border-**#{$side}:** $value;

}

* Prefix // use mixins to not repeat the prefixes for a specific property; [shouldiprefix.com](file:///C:\Users\Alma\Desktop\Pr%20learn\HTML_Css\shouldiprefix.com)

@mixin flex {

display: -webkit-box; // OLD - iOS 6-, Safari 3.1-6, BB7

display: -ms-flexbox; // TWEENER - IE 10

display: -webkit-flex; // NEW - Safari 6.1+. iOS 7.1+, BB10

display: flex; // NEW, Spec - Firefox, Chrome, Opera

}

* @include //To use them:

@include flex;

@include border('bottom', 1px solid black);

* @content //You can pass the block data to the mixin when you call the mixin

@mixin mediaQuery($screen) {

@media only screen and (min-width : $screen) {

@content;

}

}

@include mediaQuery($sm) {

**Background-color: pink;** // this is the @content

}

* Use a different scss file for functions: \_mixins.scss

conditions

* @if x == y {}
* @else if x == y {}
* @else {}

**@mixin** border($side , $value) {

**@if** $side == 'all'{

border**:** $value;

}

**@else** {

Border-#{$side}**:** $value;

}

}

loops

* @for $var from start to end //$var holds the current value of the loop. The end number is not included.

**@for $i from 1 to 10{** // until 9

.padding-#{$i}{

padding: .25rem \* $i;

}

**}**

* @each $var in list // A list is required. Don't use this unless there is no other way.

**$list: 1,2,3,4,5,6,7,8,9,10;**

**@each $i in $list{**

.padding-#{$i}{

padding: .25rem \* $i;

}

**}**

­

Functions

* @function name(){ } // It takes some inputs and does some process on them and gives back an output. It's similar to @mixin, but in the mixin, we don't want to do any process on the inputs, only to replace them. Also, mixin can not have a returned value.
* @return

**@function sum(**$a, $b**){**

@return $a + $b;

}

**@mixin sum(**$a, $b**){**

width: $a + $b;

}

* To use it

width: sum(100px, 200px) // with function

@includes sum(100px, 200px) // with mixin

* If you need to return a **property with its value**, you better use mixins. Functions just return the result.
* Use a different scss file for functions: \_functions.scss

­