02 - Sass

Variables

Quand on veut définir une variable en CSS on procède de la manière suivante

```
:root {
--color-primary: yellowgreen;
}
```

:root ou html signifie la même chose
ici on va créer une propriété globale pour notre fichier html

```
body {
background-color: var(--color-primary);
}
```

j'appelle donc ma variable pour le mettre comme couleur d'arrière plan du body

ca c'est l'utilisation des variables via CSS, rien de nouveau par rapport a Sass

Maintenant via Sass

```
$color-primary: yellowgreen;
body {
background-color: $color-primary;
}
```

Maps

Dans un projet web, il y a énormément de couleurs ou a minima les couleurs de la charte graphique
Voici une manière de définir les couleurs dans sass et de les réutiliser:

```
<body>
<section>
<h1>Hello in the Sass World !</h1>
```

```
<p>Lorem, ipsum dolor sit amet consectetur adipisicing elit. Expedita, sequi.
Suscipit ullam accusantium quae provident laboriosam at numquam, facilis repellat
optio nihil natus, molestiae deserunt saepe eos assumenda ducimus. Et!</p>
</section>
</body>
```

```
$color-primary: #303960;
$color-secondary: #ea9a96;
$color-tertiary: #f8b24f;
$color-white: #fff;

section {
background-color: $color-primary;
}

h1 {
color: $color-secondary;
}

p {
color: $color-white;
}
```

Voici la meilleure manière avec sass de définir ces couleurs une seule fois

```
$colors: (
"color-primary": #303960,
"color-secondary": #ea9a96,
"color-tertiary": #f8b24f,
"color-white": #fff,
);
```

```
section {
background-color: map-get($map: $colors, $key: color-primary);
}

h1 {
color: map-get($map: $colors, $key: color-secondary);
}

p {
color: map-get($map: $colors, $key: color-white);
}
```

cela nous permet de créer plusieurs dictionnaires ou objets contenant plusieurs propriétés de couleurs, de tailles, de polices
Et ainsi structurer notre code
Mais l'avantage c'est surtout qu'on pourra utiliser les maps quand on fera des boucles et qu'on va itérer sur chacune des valeurs pour l'affecter a une balise

Nesting

On va faire ce qu'on appelle du Element Nesting

```
section {
background-color: map-get($map: $colors, $key: color-primary);
h1 {
color: map-get($map: $colors, $key: color-secondary);
}
p {
color: map-get($map: $colors, $key: color-white);
}
}
```

En fait comme tous les éléments sont des enfants de la balise <section> on peut regrouper tous ces éléments a l'intérieur du sélecteur **section**
Et on voit ce changement s'opérer dans le fichier style.css

style.css

```
section {
background-color: #303960;
}

section h1 {
color: #ea9a96;
}

section p {
color: #fff;
}
```

Quel est l'avantage?
Nous n'avons pas besoin de définir une classe a la balise <h1> ou <p> pour définir un style propre

Je peux avoir d'autres balises h1 dans mon html, seule celle-ci définit en haut aura un style propre

```
<article>
<h1>Article 1</h1>
<p>Lorem ipsum dolor sit amet consectetur, adipisicing elit. Aliquam veniam quidem quas modi necessitatibus ipsa! Quod, ut voluptatem molestias voluptates nemo earum neque at beatae eligendi porro ullam consectetur sed.</p>
</article>
```

De la même manière on peut faire du Class Nesting

```
<section class="section">
<div class="section__content">
<h2 class="section__content--title">Ceci est un titre</h2>
<p class="section__content--paragraphe">Lorem ipsum dolor sit amet consectetur adipisicing elit. Aliquid nihil earum, odio, quasi minima quidem soluta animi culpa fugit unde incidunt officia beatae, similique rerum? Illo soluta dolorem ullam officia.</p>
<a href="" class="section__content--link">Read more</a>
</div>
</section>
```

```
.section {
background: map-get($map: $colors, $key: color-primary);

.section__content {
background: map-get($map: $colors, $key: color-secondary);

.section__content--title {
color: map-get($map: $colors, $key: color-white);
}
}
}
```

Et je peux même ajouter d'autre propriétés

```
.section {
background: map-get($map: $colors, $key: color-primary);
padding: 40px;
.section__content {
background: map-get($map: $colors, $key: color-secondary);
.section__content--title {
color: map-get($map: $colors, $key: color-white);
```

```
}
.section__content--paragraphe {
font-size: 25px;
padding: 20px 0;
}
}
}
```

C'est une façon de faire du class nesting, mais on peut encore faire mieux
En fait on se rend compte, que le mot "section" se répète, que le mot "content" se répète, et un développeur a horreur d'écrire 2 fois le même code

```
.section {
background: map-get($map: $colors, $key: color-primary);
padding: 40px;

&__content {
background: map-get($map: $colors, $key: color-secondary);

&--title {
color: map-get($map: $colors, $key: color-white);
padding: 20px;
}

&--paragraphe {
font-size: 25px;
padding: 20px 0;
}
}
}
```

Le premier & (and) représente "section", le deuxième et le troisième représente "**section__content**"

Mais si on regarde notre fichier style.css

```
.section {
background: #303960;
padding: 40px;
}
.section__content {
background: #ea9a96;
}
.section__content--title {
color: #fff;
padding: 20px;
```

```
}
.section__content--paragraphe {
font-size: 25px;
padding: 20px 0;
}
```

On voit qu'on cible toutes les classes
Donc si j'utilise ces classes ailleurs, elles seront affectées par le style défini

Mais ce que je souhaite c'est affecter ces classes qui sont a l'intérieur de cette section en particulière et ainsi de suite
Pour cela on doit modifier le code légèrement

```
.section {
background: map-get($map: $colors, $key: color-primary);
padding: 40px;
&__content {
background: map-get($map: $colors, $key: color-secondary);
#{&}--title {
color: map-get($map: $colors, $key: color-white);
padding: 20px;
}
#{&}--paragraphe {
font-size: 25px;
padding: 20px 0;
}
}
}
```

le #{&} va nous permettre de spécifier pour chaque class, sa généalogie complète qui mène a lui

Notre fichier style.css a bien changer en conséquence

```
.section {
background: #303960;
padding: 40px;
}
.section__content {
background: #ea9a96;
}
.section__content .section__content--title {
color: #fff;
padding: 20px;
}
```

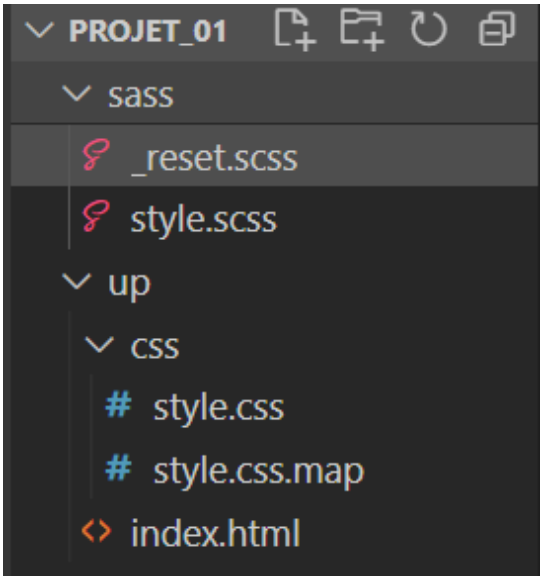
```
.section__content .section__content--paragraphe {
font-size: 25px;
padding: 20px 0;
}
```

Partials et Imports

Les partials se sont des fichiers Sass qui correspondent a des sections de notre Sass final
C'est comme-ci on partager notre fichier complet sass en petits bouts, et qu'on importerait tous ces petits bouts dans le sass final

Les partials sont super pratiques, pour éviter ainsi d'avoir un seul gros fichier contenant une centaine de lignes, devenant illisible, ou difficile a s'y retrouver

On reconnait un partial par le fait que son nom de fichier commence par un Under score (_)



Ce qui reste a faire, c'est de lier ce fichier partial au fichier sass principal, et c'est la qu'on va utiliser les imports

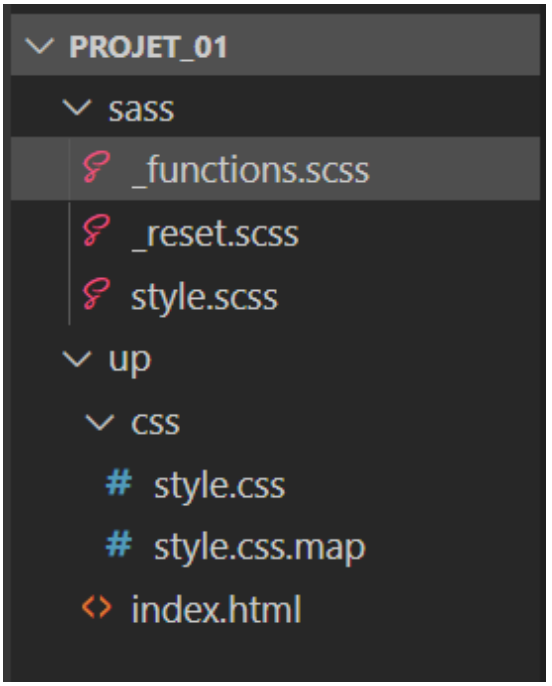
dans style.scss

```
@import 'reset';
```

dans _reset.scss

```
*,
*::before,
*::after {
padding: 0;
margin: 0;
box-sizing: border-box;
}
```

Sass Functions



style.scss

```
@import "reset";
@import "functions";
```

_functions.scss

```
$colors: (
"color-primary": #303960,
"color-secondary": #ea9a96,
"color-tertiary": #f8b24f,
"color-white": #fff,
);
.section {
background: map-get($map: $colors, $key: color-primary);
padding: 40px;
&__content {
```

```
background: map-get($map: $colors, $key: color-secondary);
}
}
```

On voit ici qu'on utilise 2 fois map-get(), et encore on a modifier notre code, sinon on l'utilisait tout a l'heure 4 fois

Une fonction dans Sass est en faite une règle (ou appelée function rule)
la syntaxe :

```
@function getColor($color-name) {
@return map-get($map: $colors, $key: $color-name);
};
```

la fonction a le mot clé @**function**, le nom de la fonction **getColor**, entre parenthèses elle reçoit un argument
et elle renvoi avec le mot clé @return, la valeur qui correspond a la couleur passée en argument

Et on peut ainsi l'utiliser

```
.section {
background: getColor(color-primary);
padding: 40px;
&__content {
background: getColor(color-tertiary);
}
}
```

Alors vous vous dites qu'on a simplement remplacer
En faite cette fonction nous permet surtout de ne passer en arguments que la couleur souhaitée, alors que **map-get()** nous demander de passer le dictionnaire de couleurs $colors et la couleur souhaitée

Sass Mixins

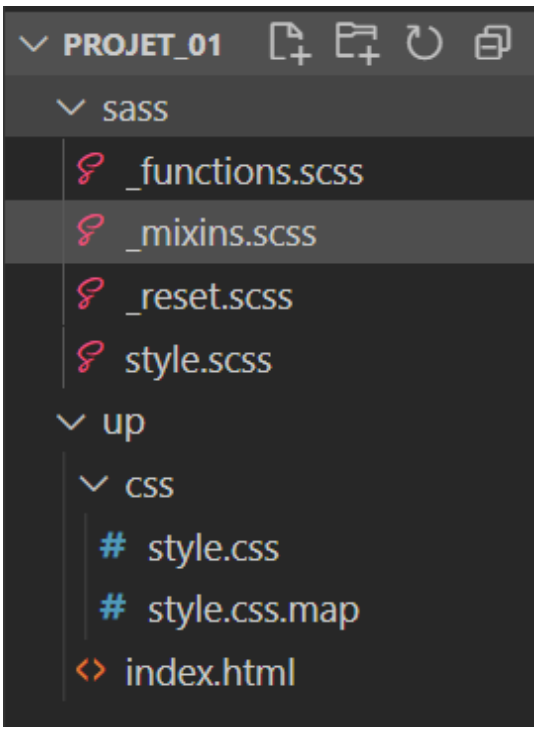
Une mixin est un morceau de code paramétrable et réutilisable n'importe où dans votre fichier SASS. Comme pour une fonction, une mixin peut prendre des paramètres.

La différence entre une mixin et une fonction est le résultat retourné une fois compilé. Une mixin peut vous générer plusieurs lignes de code CSS tandis qu'une fonction vous retournera uniquement une valeur.

On va mettre en place deux sections avec deux classes différentes

```
<body>
<section class="section-1">
<div class="section-1__content">
<h2 class="section-1__content--title">Ceci est un titre</h2>
<p class="section-1__content--paragraphe">Lorem ipsum dolor sit amet consectetur adipisicing elit. Aliquid nihil earum, odio, quasi minima quidem soluta animi culpa fugit unde incidunt officia beatae, similique rerum? Illo soluta dolorem ullam officia.</p>
<a href="" class="section-1__content--link">Read more</a>
</div>
<div class="section-1__content">
<h2 class="section-1__content--title">Ceci est un titre</h2>
<p class="section-1__content--paragraphe">Lorem ipsum dolor sit amet consectetur adipisicing elit. Aliquid nihil earum, odio, quasi minima quidem soluta animi culpa fugit unde incidunt officia beatae, similique rerum? Illo soluta dolorem ullam officia.</p>
<a href="" class="section-1__content--link">Read more</a>
</div>
</section>
<section class="section-2">
<div class="section-2__content">
<h2 class="section-2__content--title">Ceci est un titre</h2>
<p class="section-2__content--paragraphe">Lorem ipsum dolor sit amet consectetur adipisicing elit. Aliquid nihil earum, odio, quasi minima quidem soluta animi culpa fugit unde incidunt officia beatae, similique rerum? Illo soluta dolorem ullam officia.</p>
<a href="" class="section-2__content--link">Read more</a>
</div>
<div class="section-2__content">
<h2 class="section-2__content--title">Ceci est un titre</h2>
<p class="section-2__content--paragraphe">Lorem ipsum dolor sit amet consectetur adipisicing elit. Aliquid nihil earum, odio, quasi minima quidem soluta animi culpa fugit unde incidunt officia beatae, similique rerum? Illo soluta dolorem ullam officia.</p>
<a href="" class="section-2__content--link">Read more</a>
</div>
</section>
</body>
```

On créer un partial _mixins.scss



Notre fichier style.scss contient lui que des imports

```
@import "reset";
@import "functions";
@import "mixins";
```

dans _mixins.scss:

```
.section-1 {
display: flex;
align-items: center;
justify-content: center;
}
```

On va définir une couleur d'arrière-plan, mais souvenez vous qu'on a une fonction qui fait ca

```
.section-1 {
...
background: getColor(color-primary);
padding: 50px;
}
```

Voici donc notre section-1 terminer

```
.section-1 {
```

```
display: flex;
align-items: center;
justify-content: center;
background: getColor(color-primary);
padding: 50px;
margin-bottom: 10px;

&__content:last-child {
margin-left: 200px;
}
}
```

je copie colle ce bout de code pour faire la section-2 (j'enlève juste le margin-bottom de la 2eme section qui ne servira a rien)

```
.section-2 {
display: flex;
align-items: center;
justify-content: center;
background: getColor(color-tertiary);
padding: 50px;

&__content:last-child {
margin-left: 200px;
}
}
```

On voit clairement qu'on répète le code
C'est la qu'on a recours au mixin

```
@mixin section-styling() {
}
```

```
@mixin section-styling() {
display: flex;
align-items: center;
justify-content: center;
padding: 50px;
}
```

On définit un style précis dont on sait qu'on va le répéter

et ensuite il suffit de l'appeler via la méthode @include

```
.section-1 {
```

```
@include section-styling();

background: getColor(color-primary);
margin-bottom: 10px;
&__content:last-child {
margin-left: 200px;
}
}
.section-2 {
@include section-styling();

background: getColor(color-tertiary);
padding: 50px;
&__content:last-child {
margin-left: 200px;
}
}
```

Donc si je veux affecter les sections en changeant la direction des items du flex-box, il me suffit de l'ajouter au @mixin

```
@mixin section-styling() {
display: flex;
align-items: center;
justify-content: center;
padding: 50px;
flex-direction: column
}
```

Et je peux même passer en paramètre un argument comme ici la direction
Et la définir avec une valeur différente quand j'appelle

Exemple :

```
@mixin section-styling($direction) {
display: flex;
align-items: center;
justify-content: center;
padding: 50px;
flex-direction: $direction;
}
```

```
.section-1 {
@include section-styling(column);

background: getColor(color-primary);
margin-bottom: 10px;
&__content:last-child {
margin-left: 200px;
}
}
.section-2 {
```

```
@include section-styling(row);

background: getColor(color-tertiary);
&__content:last-child {
margin-left: 200px;
}
}
```

Je peux passer autant d'arguments que je le souhaite
et je peux même définir des valeurs par défaut a ces arguments
j'enlève par exemple cette ligne:


```
@mixin section-styling($direction, $m-b : 10px) {
display: flex;
align-items: center;
justify-content: center;
padding: 50px;
flex-direction: $direction;
margin-bottom: $m-b;
}
```

```
.section-1 {
@include section-styling(column);

background: getColor(color-primary);
&__content:last-child {
margin-left: 200px;
}
}

.section-2 {
@include section-styling(row, 0px);
```

```
background: getColor(color-tertiary);
&__content:last-child {
margin-left: 200px;
}
}
```

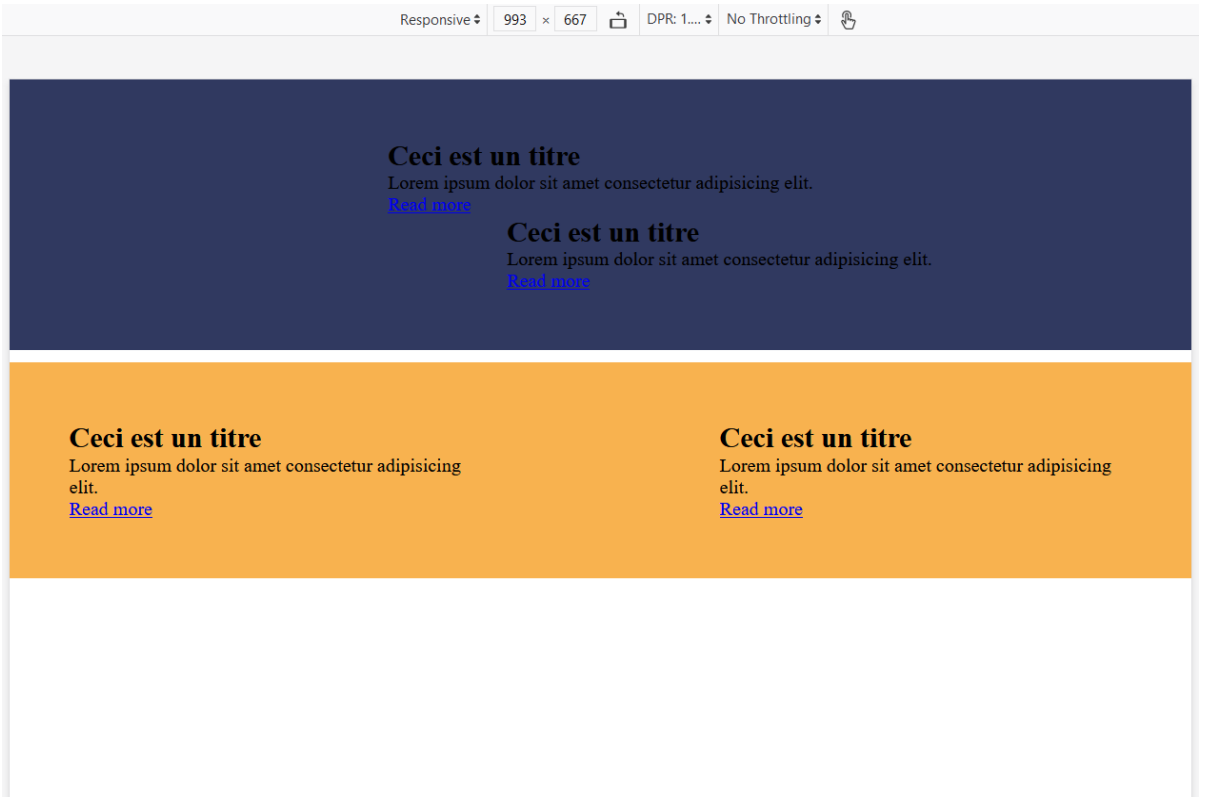
Au premier appel du mixin, je ne lui passe aucun deuxième arguments, et donc le margin-bottom sera par défaut de 10px
au deuxième appel du mixin je lui passe comme argument un m-b de 0px

Notez notre fichier style.css combien de lignes il comporte, et voyez comment en quelques lignes de sass, on pu générer toutes ces lignes

On peut également utiliser mixin pour définir des règles d'affichage en fonction de la taille de l'écran

```
$bpDesktop: 1000px;
@mixin media-desktop {
@media screen and (max-width: $bpDesktop) {
@content;
}
}
```

Avant l'utilisation du mixin



```
.section-2 {
@include media-desktop() {
}
}
```

A l'intérieur de ce block de code, nous allons justement remplace le contenu (représenter par $content)

```
.section-2 {
@include media-desktop() {
flex-direction: column;
}
}
```

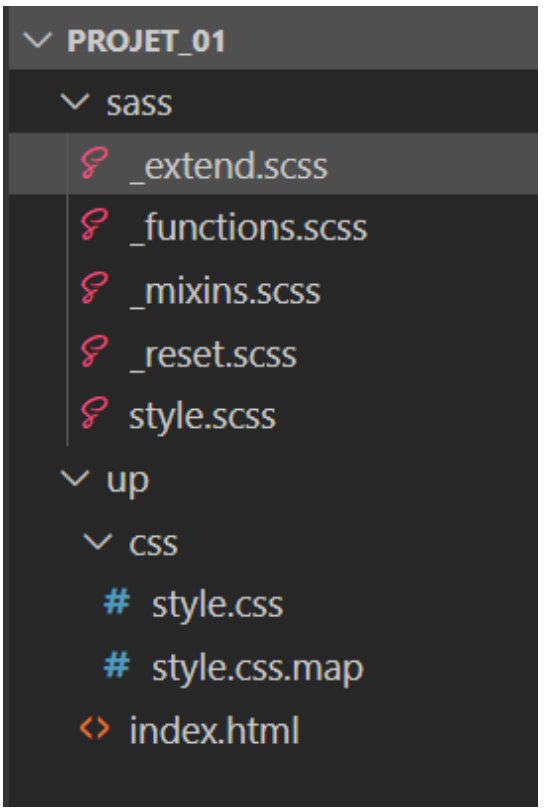
```
.section-2 {
@include media-desktop() {
flex-direction: column;

&__content:last-child {
margin-left: 0;
}
}
}
```

On peut en faire de même pour la section-1

```
.section-1 {
@include media-desktop() {
&__content:last-child {
margin-left: 0px;
}
}
}
```

@extend rule




```
@import "reset";
@import "functions";
// @import "mixins";
@import "extend";
```

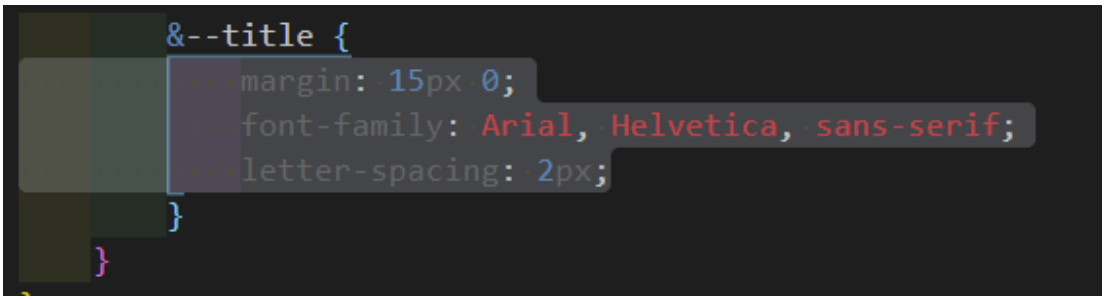
On met en commentaire le import de mixins

```
.section-1 {
background-color: getColor(color-primary);

&__content {
padding: 20px;
color: getColor(color-white);

&--title {
margin: 15px 0;
font-family: Arial, Helvetica, sans-serif;
letter-spacing: 2px;
}
}
}
```

Maintenant admettons que nous souhaitions que le paragraphe ait comme style ces 3 lignes de code contenu dans le sélecteur du titre pour le paragraphe



Pour cela on va utiliser **@extend**

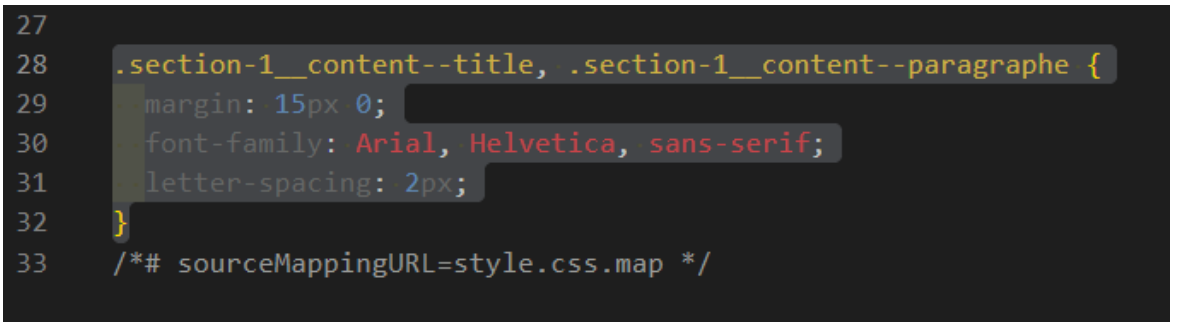
```
.section-1 {
background-color: getColor(color-primary);

&__content {
padding: 20px;
color: getColor(color-white);

&--title {
margin: 15px 0;
font-family: Arial, Helvetica, sans-serif;
letter-spacing: 2px;
}

&--paragraphe {
@extend .section-1__content--title;
}
}
}
```

On peut voir ainsi dans le fichier style.css que c'est bien appliquer aux deux sélecteurs



et on peut l'appliquer au paragraphes de la section-2

```
.section-2 {
&__content {
```

```
&--paragraphe {
@extend .section-1__content--title;
}
}
}
```

Une deuxième méthode pour utiliser étendre des styles a d'autres sélecteurs
la méthode s'appelle Placeholder rulesets

On utilise généralement le placeholder rulesets pour fixer des règles générales pour les textes, les images, les formulaires
Et cette fois-ci on utilise le % pour le différencier

```
%text-styling {
margin: 15px 0;
font-family: Arial, Helvetica, sans-serif;
letter-spacing: 2px;
}
```

Et on l'utilise toujours via @extend

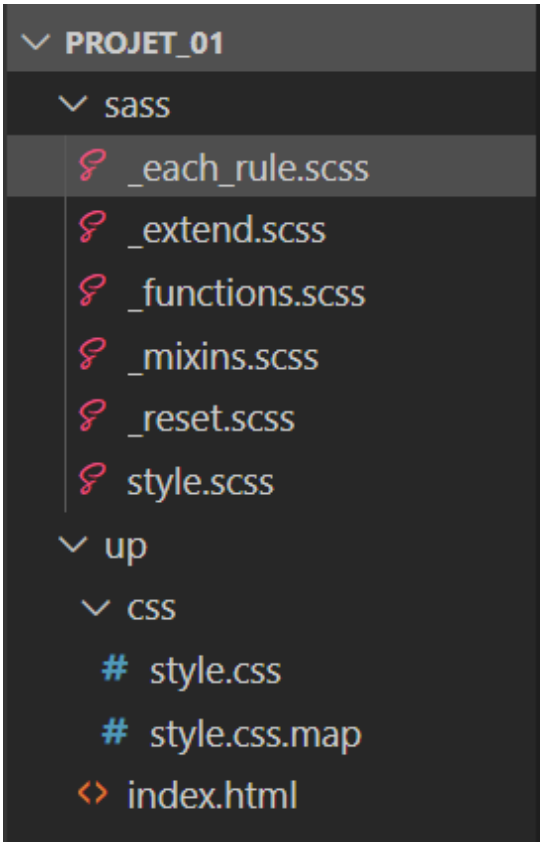
```
.section-1 {
background: getColor(color-primary);
&__content {
padding: 20px;
color: getColor(color-white);
&--title {
@extend %text-styling;
}
&--paragraphe {
@extend %text-styling;
}
}
}
```

@each rule

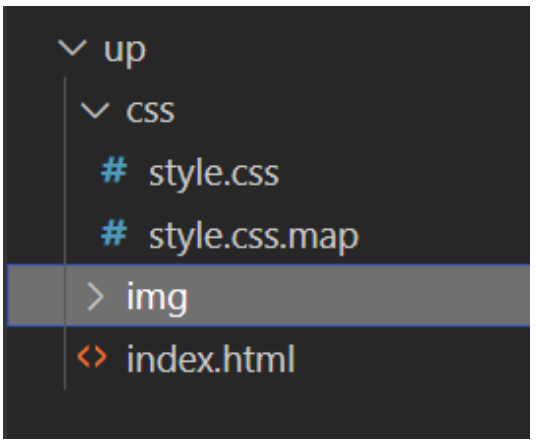
@each rule va nous permettre de répéter le style en utilisant une liste
c'est comme itérer dans une liste et effectuer la même tache pour chaque élément de la liste

```
<body>
<section>
<div class="ballet"></div>
<div class="cat"></div>
<div class="gramophone"></div>
<div class="sparrow"></div>
<div class="sunset"></div>
<div class="turkey"></div>
</section>
</body>
```

Nous avons 6 div, imaginez que cela représente des témoignages, des cartes ou des membres d'une équipe



On créer un dossier "img" dans le dossier "up"




```
section {
div {
height: 30rem;
width: 50rem;
}
}

section {
.ballet {
background-image: url("../img/ballet.jpg");
}
}
```

Attention: pour le chemin d'accès a l'image, il ne faut pas oublier que le code sass est convertis en code css, et qu'il est compiler dans le fichier style.css
donc le chemin doit partir depuis le fichier style.css qui se trouve dans "up/css/style.css"
Or, pour aller depuis style.css a l'image, on doit monter d'un cran et aller ensuite dans le dossier "img"
d'où le chemin:

```
"../img/ballet.jpg"
```

```
section {
.ballet {
background-image: url("../img/ballet.jpg");
background-repeat: no-repeat;
background-size: cover;
background-position: center;
}
.cat {
```

```
background-image: url("../img/cat.jpg");
background-repeat: no-repeat;
background-size: cover;
background-position: center;
}
}
```

Et vous avez compris, on répète l'opération pour chaque div
on constate que les noms des fichiers images ont le même nom que les class de chaque div

on constate que ces 3 lignes sont identiques

la seule chose qui change, c'est le nom du fichier qui pour le coup a le même nom que la classe de la div (soit-dit en passant, c'est une mauvaise pratique)
On pourrait utiliser un placeholder ruleset, mais ici le @each rule fera mieux l'affaire et plus rapidement

```
$bg-pics : (ballet, cat, gramophone, sparrow, sunset, turkey);

@each $bg-div in $bg-pics {
.#{$bg-div} {
background-image: url("../img/#{$bg-div}.jpg");
background-repeat: no-repeat;
background-size: cover;
background-position: center;
}
}
```

$bg-div est ce qu'on appelle un itérateur, c'est lui qui va prendre a tour de rôle comme valeur chaque élément de la liste $bg-pics
Pour utiliser sa valeur on l'appelle de la manière suivante:

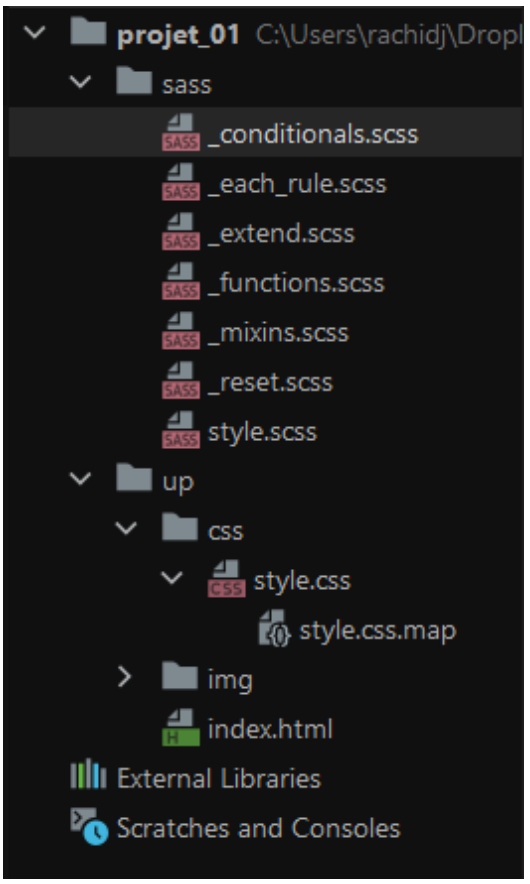
```
#{$bg-div}
```

le code suivant marcherait tout autant:

```
@each $bg-div in (ballet, cat, gramophone, sparrow, sunset, turkey) {
.#{$bg-div} {
background-image: url("../img/#{$bg-div}.jpg");
background-repeat: no-repeat;
background-size: cover;
background-position: center;
}
}
```

on ne passe pas par une variable contenant la liste

Sass Conditionals | @if Rule



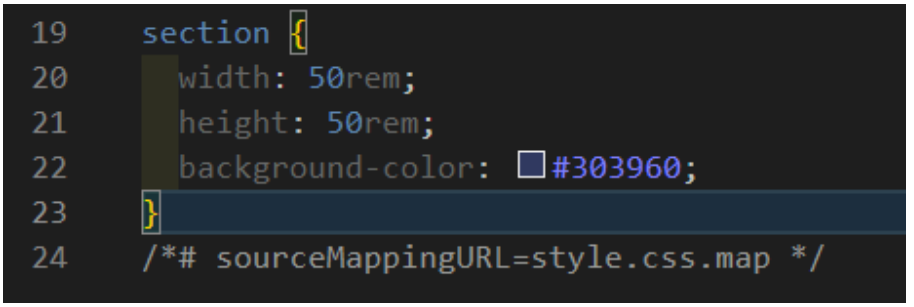

```
@import "reset";
@import "functions";
// @import "mixins";
// @import "extend";
//@import "each_rule";
@import "conditionals";
```

```
@mixin section($size) {
width: $size;
height: $size;
@if $size < 200 {
background-color: getColor(color-primary);
} @else if($size == 250px) {
background-color: getColor(color-secondary);
} @else {
background-color: green;
}
}
```

```
section {
@include section(50rem);
```

```
}
```

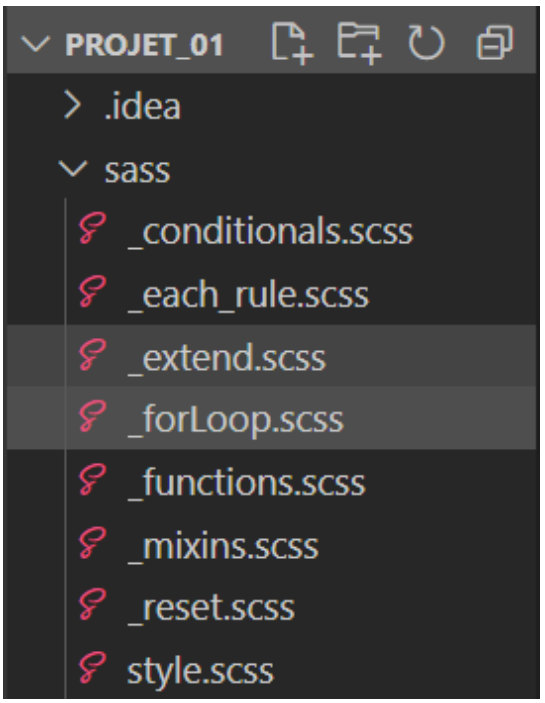
Et on peut voir l'implémentation de ce code dans le fichier style.css



Sass For Loops

```
div*6.gallery__content-$
```

```
<body>
<section class="gallery">
<div class="gallery__content-1"></div>
<div class="gallery__content-2"></div>
<div class="gallery__content-3"></div>
<div class="gallery__content-4"></div>
<div class="gallery__content-5"></div>
<div class="gallery__content-6"></div>
</section>
</body>
```




```
.gallery {
div[class^= "gallery__content"] {
height: 30rem;
width: 30rem;
}
}
```

```
@for $img from 1 through 6 {
.gallery__content-#{$img} {
background-image: url("../img/gallery-#{$img}.jpg");
background-repeat: no-repeat;
background-position: center;
background-size: cover;
}
}
```

#{...} est appeler le template literal, il permet d'acceder a la valeur d'une variable

Sass While Loop

```
.gallery {
div[class^= "gallery__content"] {
```

```
height: 30rem;
width: 30rem;
}
}
```

```
$img : 1;
@while $img < 7 {
.gallery__content-#{$img} {
background-image: url("../img/gallery-#{$img}.jpg");
background-repeat: no-repeat;
background-position: center;
background-size: cover;
}
$img : $img + 1;
}
```

Attention si on oublie le

```
$img : $img + 1;
```

On aura une boucle infini, en effet si on augmente pas de 1 a chaque tout, la valeur de $img reste tjrs egale a 1 et donc 1 est tjrs < a 7, la boucle ne se termine jamais

Structure d'un projet avec SASS
Le pattern 7-1

The architecture known as the *7–1 pattern* (7 folders, 1 file), is a widely-adopted structure that serves as a basis for large projects. You have all your partials organized into 7 different folders, and a single file sits at the root level (usually named main.scss) to handle the imports — which is the file you compile into CSS.

Here's a sample 7–1 directory structure, I've included some examples of files that would sit inside of each folder:

```
sass/
|
|– abstracts/ (or utilities/)
| |– _variables.scss // Sass Variables
| |– _functions.scss // Sass Functions
| |– _mixins.scss // Sass Mixins
|
```

```
|- base/
| |- _reset.scss // Reset/normalize
| |- _typography.scss // Typography rules
|
|- components/ (or modules/)
| |- _buttons.scss // Buttons
| |- _carousel.scss // Carousel
| |- _slider.scss // Slider
|
|- layout/
| |- _navigation.scss // Navigation
| |- _grid.scss // Grid system
| |- _header.scss // Header
| |- _footer.scss // Footer
| |- _sidebar.scss // Sidebar
| |- _forms.scss // Forms
|
|- pages/
| |- _home.scss // Home specific styles
| |- _about.scss // About specific styles
| |- _contact.scss // Contact specific styles
|
|- themes/
| |- _theme.scss // Default theme
| |- _admin.scss // Admin theme
|
|- vendors/
| |- _bootstrap.scss // Bootstrap
| |- _jquery-ui.scss // jQuery UI
|
`- main.scss // Main Sass file
```

Abstracts (or utilities): holds Sass tools, helper files, variables, functions, mixins and other config files. These files are meant to be just helpers which don't output any CSS when compiled.

Base: holds the boilerplate code for the project. Including standard styles such as resets and typographic rules, which are commonly used throughout your project.

Components (or modules): holds all of your styles for buttons, carousels, sliders, and similar page components (think widgets). Your project will typically contain a lot of component files — as the whole site/app should be mostly composed of small modules.

Layout: contains all styles involved with the layout of your project. Such as styles for your header, footer, navigation and the grid system.

Pages: any styles specific to individual pages will sit here. For example it's not uncommon for the home page of your site to require page specific styles that no other page receives.

Themes: this is likely not used in many projects. It would hold files that create project specific themes. For example if sections of your site contain alternate color schemes.

Vendors: contains all third party code from external libraries and frameworks — such as Normalize, Bootstrap, jQueryUI, etc.
However, there is often a need to override vendor code. If this is required, its good practice to create a new folder called vendors-extensions/ and then name any files after the vendors they overwrite. The filevendors-extensions/_bootstrap.scss would contain all your Bootstrap overrides — as editing the vendor files themselves, isn't generally a good idea.

Main.scss: This file should only contain your imports! For example..

```
@import 'abstracts/variables';
@import 'abstracts/functions';
@import 'abstracts/mixins';

@import 'vendors/bootstrap';
@import 'vendors/jquery-ui';

@import 'base/reset';
@import 'base/typography';

@import 'layout/navigation';
@import 'layout/grid';
@import 'layout/header';
@import 'layout/footer';
@import 'layout/sidebar';
@import 'layout/forms';

@import 'components/buttons';
@import 'components/carousel';
@import 'components/slider';

@import 'pages/home';
@import 'pages/about';
@import 'pages/contact';
```

```
@import 'themes/theme';
@import 'themes/admin';
```