

SASS



Scss et Sass

Sass est un préprocesseur CSS.

Il existe deux syntaxes, la syntaxe d'origine, Sass et la syntaxe se rapprochant de celle du css, le Scss.

La syntaxe du sass est "indentée", voici une comparaison des deux syntaxes:

```
1  div {
2      background: black;
3      border: 1px solid black;
4      border-radius: 15px;
5      height: 100px;
6
7      p {
8          font-size: 3rem;
9          font-style: italic;
10     }
11 }
```

```
1  div
2      background: black
3      border: 1px solid black
4      border-radius: 15px
5      height: 100px
6
7      p
8          font-size: 3rem
9          font-style: italic
10
```

Installation de dartsass

Dartsass va nous permettre de compiler notre code scss en css.

Afin de l'installer il faut se rendre sur le site officiel de Sass : <https://sass-lang.com>

Nous allons choisir de l'utiliser en ligne de commande. Une fois Dartsass téléchargé et le dossier mis dans notre projet il faut entrer la ligne de commande suivante:

```
./dart-sass/sass ./assets/scss/style.scss ./assets/css/style.css --watch
```

Lancement
de dartsass

chemin de
notre fichier
scss

chemin de
destination
pour le css

option de
compilation a
la sauvegarde

L'importation

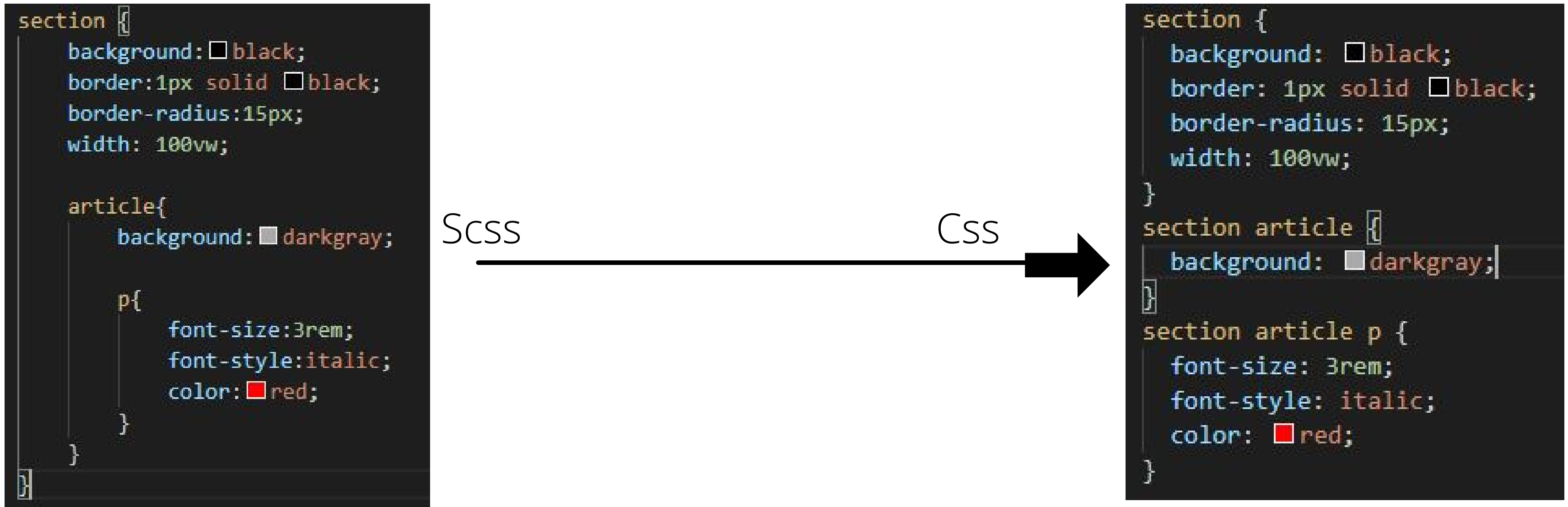
Grace au système d'importation du Sass, il est beaucoup plus simple d'avoir une architecture claire:

```
//IMPORTATION
@import "vars";
@import "fonts";
@import "header";
@import "corpse";
@import "subscribe";
@import "housemade";
@import "infohouserules";
@import "footer";
@import "mediaquery";
```

```
? _corpse.scss
? _fonts.scss
? _footer.scss
? _header.scss
? _housemade.scss
? _infohouserules.scss
? _mediaquery.scss
? _subscribe.scss
? _vars.scss
? style.scss
```

L'imbrication

La syntaxe Scss permet de coder en utilisant un système d'imbrication :



L'imbrication de pseudo-élément / pseudo-classes

Pour imbriquer un pseudo-élément ou pseudo-classe il faut rajouter le caractère & :

```
> div {  
  width: 45vw;  
  
  &:first-child {  
    margin-right: 10px;  
  }  
}
```

Les variables

Les variables permettent d'assigner une valeur a un nom qui commencera par \$

```
//VARIABLE SCSS
$primaryColor: #587D71;
$secondaryColor: #F9ECCC;
$thirdColor: #754668;
$fourthColor: #B5DDA4;
$fifthColor: #4DAA57;
$grey: #D1D1D1;
$shadowDefault: 0px 3px 6px grey;
```

```
header {
  background-color: $primaryColor;
  padding: 10px;
  display: flex;
  justify-content: space-between;
```

Les opérations simple

```
body {  
  padding: 4px + 5px;  
  margin: (4+5)px;  
}
```

```
$maVariableA: "Hello ";  
$maVariableB: "World";  
$content: $maVariableA + $maVariableB;  
// résultat "Hello World";
```

```
body {  
  padding: (4px*6);  
  margin: (20px/2);  
}
```


Extend

Extend permet d'éviter les répétitions d'attributs:

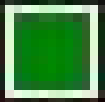
```
.button {  
    background-color: green;  
    width: 200px;  
    height: 75px;  
    font-size: 30px;  
}  
  
.buttonsend {  
    @extend .button;  
    font-style: italic;  
}
```

```
%alert {  
    border: 1px solid white;  
    padding: 30px;  
    color: white;  
    border-radius: 10px;  
    font-family: 'Kulim Park';  
}
```

Mixin

les mixins permettent une sauvegarde de configurations de nos attributs:

```
@mixin center {  
  display: block;  
  text-align: center;  
  margin-left: auto;  
  margin-right: auto;  
}
```

```
div {  
  width: 225px;  
  background:  green;  
  @include center;  
}
```

Mixin et arguments

il est possible de rajouter des arguments dans nos mixins;

```
@mixin flex($just:center, $dir:row, $align) {  
  display: flex;  
  flex-direction: $dir;  
  justify-content: $just;  
  align-items: $align;  
  flex-wrap: wrap;  
}
```

```
div {  
  @include flex(space-between, column, flex-start);  
  width: 45vw;  
}
```

Les fonction natives

Sass contient plusieurs fonction de base. En voici certains exemple :

darken()	permet d'assombrir une couleur d'un % donné
lighten()	permet d'éclaircir une couleur d'un % donné
round()	permet d'arrondir un décimal

<https://sass-lang.com/documentation/modules>

Les fonction natives

Sass contient plusieurs fonction de base. En voici certains exemple :

darken()	permet d'assombrir une couleur d'un % donné
lighten()	permet d'éclaircir une couleur d'un % donné
round()	permet d'arrondir un décimal
nth()	permet de sélectionner les éléments d'une liste

<https://sass-lang.com/documentation/modules>

Les fonction personnalisée

il est possible d'écrire nous même une fonction :

```
@function function-name($args) {  
  @return value;  
}
```

```
@function absolute-center($contenerbox,$width) {  
  @return $contenerbox - $width / 2;  
}
```

```
div{  
  position: absolute;  
  left: absolute-center(300px,100%);  
}
```

les conditions

Voici la syntaxe pour écrire des conditions en scss

```
@if (condition) {  
  //Fonction, propriété, mixin  
} @else if (condition) {  
  // Puis une autre  
} [...] // Autant de @else if (condition) que l'on désire  
@else {  
  // Si aucune condition n'est vérifiée  
}
```

les conditions (opérateurs de comparaisons)

`==` , tester une égalité

`!=` , tester une différence

`>` , tester une supériorité

`<` , tester une infériorité

`>=` , tester une supériorité ou une égalité

`<=` , tester une infériorité ou une égalité

les boucles

Les boucles permettent de répéter des opérations d'un nombre de fois donné.
il existe 3 types de boucles :

La boucle @for

La boucle @while

La boucle @each

la boucle for

```
@for $i from 1 through 6 {  
  .grid-#{ $i } {  
    width: 100px*$i;  
  }  
}
```

```
.grid-1 {  
  width: 100px; }
```

```
.grid-2 {  
  width: 200px; }
```

```
.grid-3 {  
  width: 300px; }
```

```
.grid-4 {  
  width: 400px; }
```

```
.grid-5 {  
  width: 500px; }
```

```
.grid-6 {  
  width: 600px; }
```

la boucle while

```
$i: 0;

@while $i <= 6 {
    .grid-#{ $i } {
        width: 100px*$i;
    }
    $i: $i + 1;
}
```

```
.grid-0 {
    width: 0px; }

.grid-1 {
    width: 100px; }

.grid-2 {
    width: 200px; }

.grid-3 {
    width: 300px; }

.grid-4 {
    width: 400px; }

.grid-5 {
    width: 500px; }

.grid-6 {
    width: 600px; }
```

la boucle each

```
@each $name in mon-site, bootstrap, php, sass {  
  .#{$name}-icon {  
    background-image: url('/assets/#{$name}.png');  
  }  
}
```

```
mon-super-site-icon {  
  background-image: url(/assets/mon-super-site.png); }
```

```
bootstrap-icon {  
  background-image: url(/assets/bootstrap.png); }
```

```
php-icon {  
  background-image: url(/assets/php.png); }
```

```
sass-icon {  
  background-image: url(/assets/sass.png); }
```

Les listes

```
$liste: banane, poire, peche;
```

```
@each $name in $liste {  
  .#{$name}-icon {  
    background-image: url('/assets/#{$name}.png');  
  }  
}
```

```
.banane-icon {  
  background-image: url("/assets/banane.png"); }
```

```
.poire-icon {  
  background-image: url("/assets/poire.png"); }
```

```
.peche-icon {  
  background-image: url("/assets/peche.png"); }
```

Exemple de fonction : nth

```
$liste: banane, poire, peche;
```

```
footer {  
  padding: 50px;  
  background-color: $primaryColor;  
  @include border-radius(15px);  
  content: nth($liste, 1);  
  font-family: 'Roboto';  
}
```

```
footer {  
  padding: 50px;  
  background-color: purple;  
  -webkit-border-radius: 15px;  
  -moz-border-radius: 15px;  
  border-radius: 15px;  
  content: banane;  
  font-family: 'Roboto'; }
```