

# LES PROCESSEURS

## CSS

Sass



stylus

{less}

# Les préprocesseurs - C'est quoi ?

Un préprocesseur CSS est un outil (ou programme) informatique permettant de générer dynamiquement des fichiers CSS. L'objectif est d'améliorer l'écriture de ces fichiers, en apportant plus de flexibilité au développeur web.

Source : [https://fr.wikipedia.org/wiki/Pr%C3%A9processeur\\_CSS](https://fr.wikipedia.org/wiki/Pr%C3%A9processeur_CSS)

# Les préprocesseurs - C'est quoi ?

## Est-ce nécessaire ?

Les préprocesseurs ne vont pas vous permettre de faire des choses que vous ne pourriez pas faire en CSS, ils vont surtout permettre d'**améliorer** votre **productivité** en vous permettant par exemple d'éviter la répétition et de mieux **organiser**.

# Les préprocesseurs - C'est quoi ?

*stylus*

{less}

*Sass*

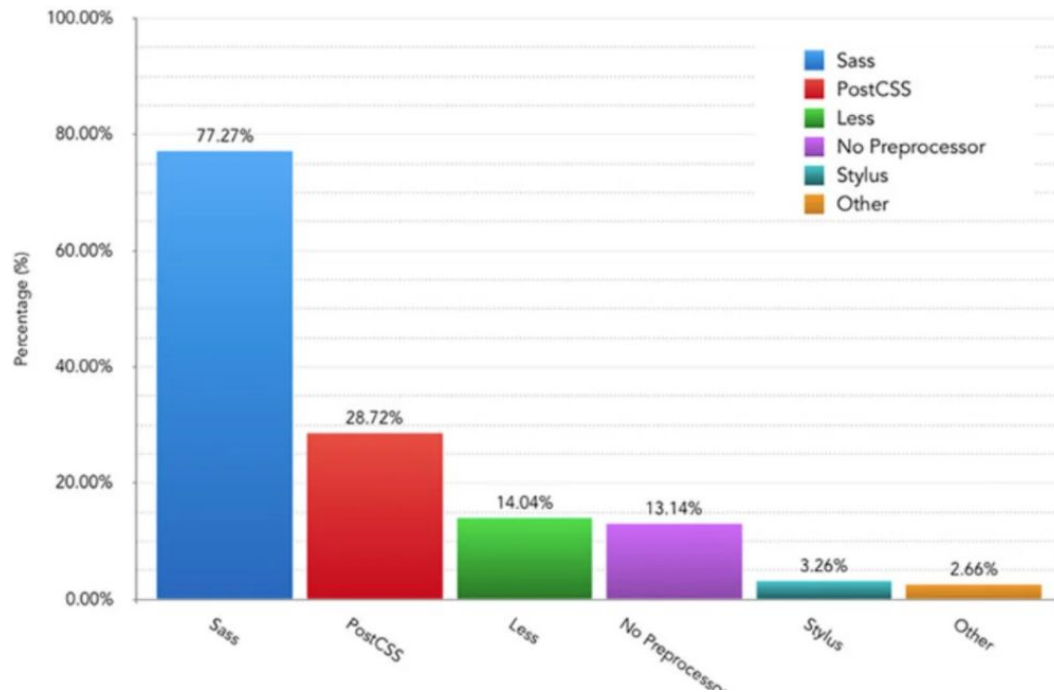
CSS   
CRUSH

*Myth*



PostCSS

# Les préprocesseurs - C'est quoi ?



Source : <https://ashleynolan.co.uk/blog/frontend-tooling-survey-2019-results>

Sass

# Les préprocesseurs - SASS

## SASS, l'incontournable

SASS est un préprocesseur qui permet de travailler avec un langage différent du CSS : le SCSS. Ce langage ressemble à du CSS classique mais ajoute certaines fonctionnalités comme la possibilité d'imbriquer les règles ou la possibilité de créer des fonctions réutilisables.

C'est un des préprocesseurs les plus utilisé aujourd'hui et il est assez simple d'utilisation.

```
$primary: #CCC;

@import "reset";

.btn {
  color: #FFF;
  background-color: $primary;

  &:hover {
    background-color: darken($primary, 10);
  }

  .icon {
    display: inline-block;
  }
}

@for $i from 1 through 3 {
  .spacer-#{ $i } {
    margin: $i * 1rem;
  }
}
```

# Les préprocesseurs - SASS

## Installation de SASS :

- Installer NodeJS (<https://nodejs.org/en/>).  
Prendre la version LTS (plus stable)

(En installant NodeJS, vous aurez accès à sa bibliothèque de modules NPM. SASS en fait parti)



# Les préprocesseurs - SASS

## Installation de SASS :

- Ouvrez l'invite de commande (cmd) ou powershell (windows). Taper la ligne :



```
Windows PowerShell  
PS C:\Users\Nicolas\Desktop\sass\scss> npm i sass -g
```

npm i sass -g

On dit qu'on va chercher un truc dans NPM

"i" veut dire qu'on l'installe (raccourci de "install")

"Sass", c'est le nom du module qu'on installe

"-g" signifie qu'on l'installe en global sur notre ordinateur

# Les préprocesseurs - SASS

## Générer du CSS via SASS :

- Créer un dossier (appelons le scss), puis créer un fichier “**style.scss**”

Et là vous vous dites....attend, c'est SASS ou SCSS.....

En fait, SASS est une syntaxe ET le nom du préprocesseur.

SCSS est une syntaxe uniquement. Le langage SASS c'est pas vraiment clair et apprécié, on lui préfère le SCSS.

Donc, on code en SCSS mais c'est le préprocesseur SASS qui va compiler le code en CSS lisible par les navigateurs. CQFD !

**.SCSS**

```
.button {  
  background: cornflowerblue;  
  border-radius: 5px;  
  padding: 10px 20px;  
  
  @:hover {  
    cursor: pointer;  
  }  
  
  @:disabled {  
    cursor: default;  
    background: grey;  
    pointer-events: none;  
  }  
}
```

**.sass**

```
.button  
  background: cornflowerblue  
  border-radius: 5px  
  padding: 10px 20px  
  
  @:hover  
    cursor: pointer  
  
  @:disabled  
    cursor: default  
    background: grey  
    pointer-events: none
```

# Les préprocesseurs - SASS

## Générer du CSS via SASS :

- Ecrivez quelque chose dans votre fichier style.scss :

```
H1{  
  Font-size:30px;  
}
```

style.scss

```
h1{  
  font-size:30px;  
}
```

# Les préprocesseurs - SASS

## Générer du CSS via SASS :

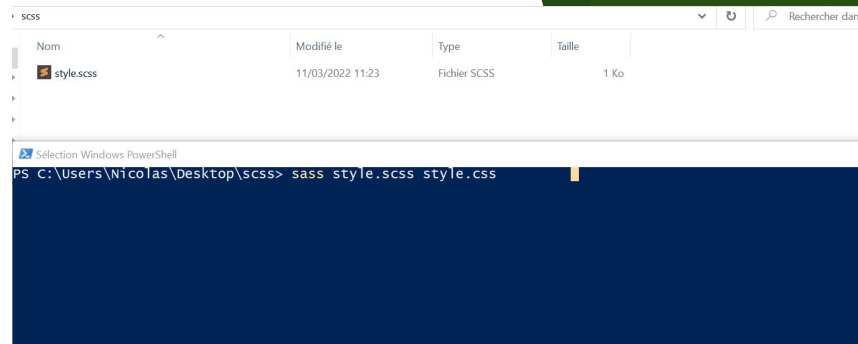
- Ouvrez l'invite de commande (ou powerShell) **DANS** le dossier contenant le fichier style.scss, puis tapez :

Sass style.scss style.css

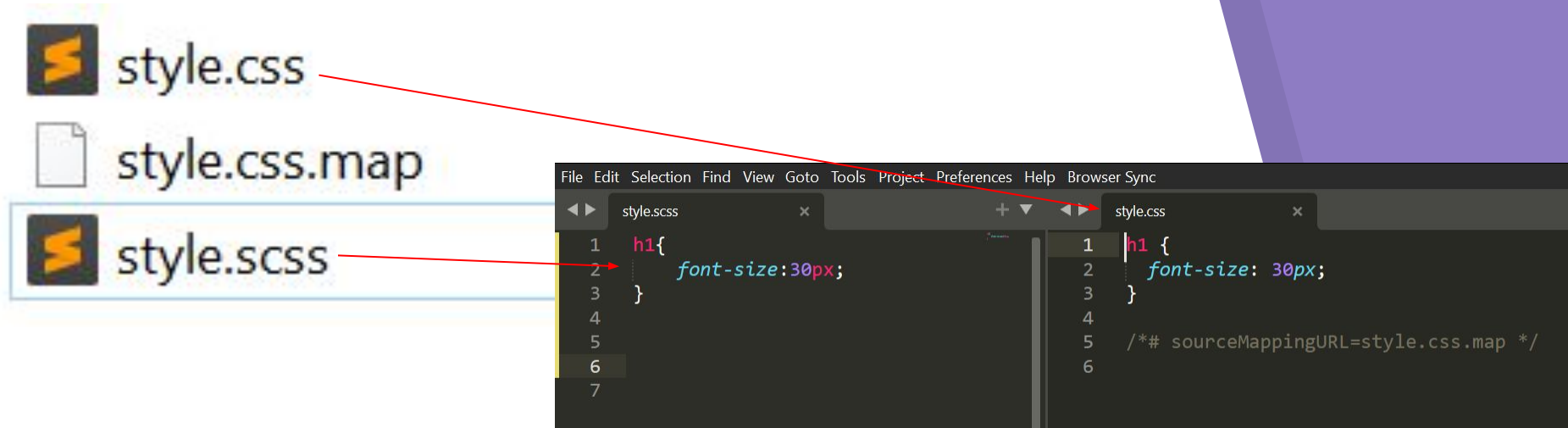
On dit qu'on va faire appel à SASS

On lui dit où est le fichier scss à convertir en css

On lui dit où il doit mettre le fichier css final (donc là, au même niveau que le fichier scss)



# Les préprocesseurs - SASS



SASS vous a généré votre fichier CSS à partir du fichier `style.scss` (et un autre `.map`).

Et là, vous vous dites, attend, attend, attend... TOUT CA pour faire un truc qu'on pouvait faire sans ton put\*\*\* de SASS ?????

# Les préprocesseurs - SASS



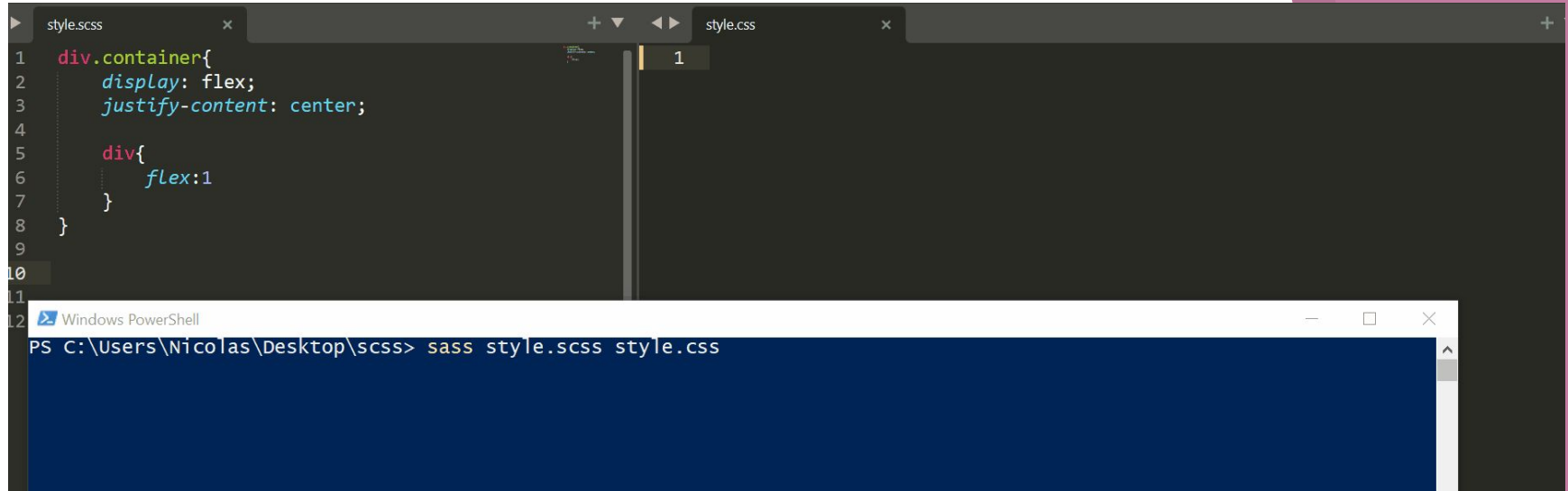
Ne vous inquiétez pas, on fait pas que ça avec SASS sinon ça servirait à rien de prendre tout ce temps pour l'installer...

# Les préprocesseurs - SASS



Il est temps de déployer la vraie PUISSANCE de SASS !!!!  
Nous allons voir l'imbrication, les variables, les mixins, les fonctions, les boucles.

# SASS -L'imbrication (nesting)



The image shows a code editor with two tabs: 'style.scss' and 'style.css'. The 'style.scss' tab is active and contains the following SASS code:

```
1  div.container{  
2    display: flex;  
3    justify-content: center;  
4  
5    div{  
6      flex:1  
7    }  
8  }  
9  
10  
11  
12
```

The 'style.css' tab is also visible and contains the following CSS code:

```
1
```

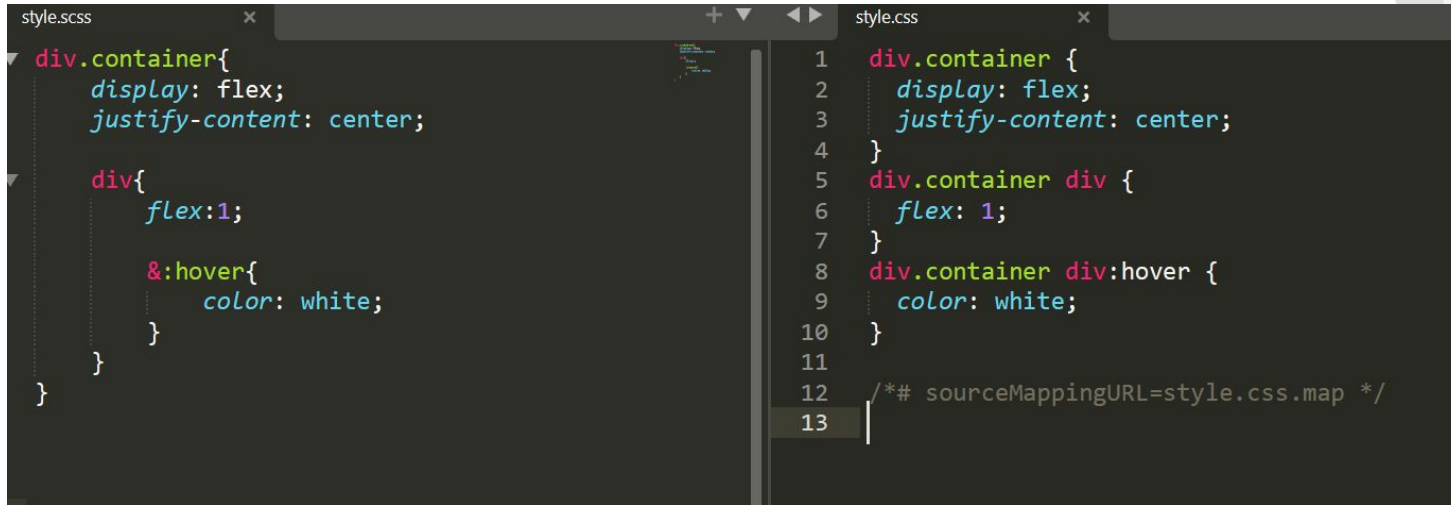
Below the code editor, a Windows PowerShell terminal window is open, showing the command to compile the SASS file:

```
PS C:\Users\Nicolas\Desktop\scss> sass style.scss style.css
```

L'écriture imbriquée (impossible en css) permet de gagner un temps considérable lors de développement ! De plus, elle offre une meilleure organisation du code (plus lisible, on évite les répétitions)



# SASS -L'imbrication (nesting)



The screenshot shows a code editor with two panels. The left panel, titled 'style.scss', contains SASS code using nesting. The right panel, titled 'style.css', shows the compiled CSS output. The SASS code defines a 'div.container' with 'display: flex' and 'justify-content: center'. Inside it, a 'div' is defined with 'flex: 1'. A pseudo-selector '&:hover' is used to set 'color: white;'. The compiled CSS shows the resulting nested selectors: 'div.container { ... }', 'div.container div { ... }', and 'div.container div:hover { ... }'. A source map comment is also visible at the bottom of the CSS panel.

```
style.scss
div.container{
  display: flex;
  justify-content: center;

  div{
    flex:1;

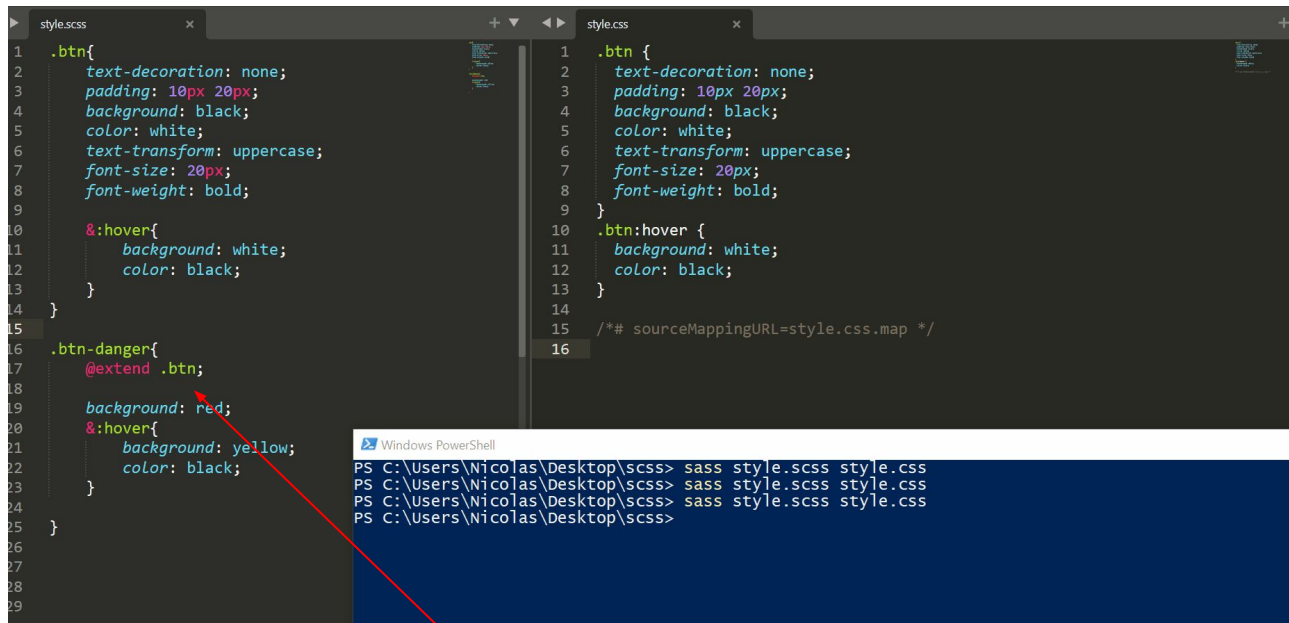
    &:hover{
      color: white;
    }
  }
}

style.css
1  div.container {
2    display: flex;
3    justify-content: center;
4  }
5  div.container div {
6    flex: 1;
7  }
8  div.container div:hover {
9    color: white;
10 }
11
12 /*# sourceMappingURL=style.css.map */
13
```

En SASS, pour ajouter des pseudo-selecteurs / pseudo-classes, on utilise le symbole “&” qui signifie “élément parent”.

L'imbrication peut avoir ses limites car trop de niveaux d'imbrication peut rendre le code illisible. Je vous conseille 3 niveaux d'imbrication MAX

# SASS -L'héritage (@extend)



The screenshot shows a code editor with two panels. The left panel displays SASS code in `style.scss`, and the right panel displays the compiled CSS in `style.css`. A Windows PowerShell terminal window is open at the bottom, showing the command to compile the SASS file.

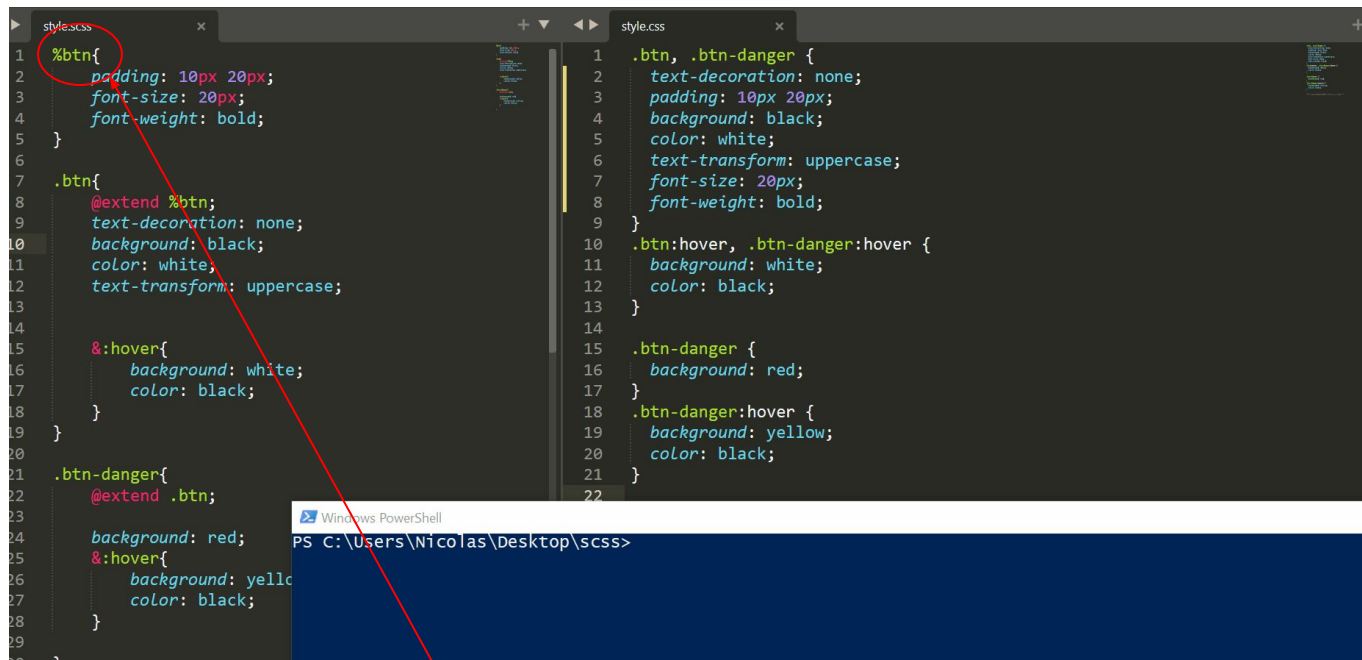
```
style.scss
1 .btn{
2   text-decoration: none;
3   padding: 10px 20px;
4   background: black;
5   color: white;
6   text-transform: uppercase;
7   font-size: 20px;
8   font-weight: bold;
9
10  &:hover{
11    background: white;
12    color: black;
13  }
14 }
15
16 .btn-danger{
17   @extend .btn;
18
19   background: red;
20   &:hover{
21     background: yellow;
22     color: black;
23   }
24 }
25
26
27
28
29
```

```
style.css
1 .btn {
2   text-decoration: none;
3   padding: 10px 20px;
4   background: black;
5   color: white;
6   text-transform: uppercase;
7   font-size: 20px;
8   font-weight: bold;
9 }
10 .btn:hover {
11   background: white;
12   color: black;
13 }
14
15 /*# sourceMappingURL=style.css.map */
16
```

```
Windows PowerShell
PS C:\Users\Nicolas\Desktop\scss> sass style.scss style.css
PS C:\Users\Nicolas\Desktop\scss> sass style.scss style.css
PS C:\Users\Nicolas\Desktop\scss> sass style.scss style.css
PS C:\Users\Nicolas\Desktop\scss>
```

@extend suivi de l'élément à copier permet de dupliquer les propriétés sans avoir à les répéter dans le code. Ex: @extend .btn va ajouter toutes les propriétés de la "class btn" sans que j'ai besoin de tout retaper.

# SASS -L'héritage (@extend)



The screenshot shows a code editor with two files: `style.scss` and `style.css`. In `style.scss`, a SASS mixin `%btn` is defined with properties `padding: 10px 20px;`, `font-size: 20px;`, and `font-weight: bold;`. This mixin is then extended by the `.btn` class and the `.btn-danger` class. The `.btn` class also has `text-decoration: none;`, `background: black;`, `color: white;`, and `text-transform: uppercase;`. The `.btn-danger` class has `background: red;`. Both classes have a `&:hover` selector that changes the background and color. The `style.css` file shows the resulting CSS, where the `.btn` and `.btn-danger` classes and their hover states are listed. A red circle highlights the `%btn` definition in the SCSS file, and a red arrow points from it to the `background: yellow;` line in the PowerShell terminal. The PowerShell terminal shows the command `PS C:\Users\Nicolas\Desktop\scss>`.

```
1 %btn{
2   padding: 10px 20px;
3   font-size: 20px;
4   font-weight: bold;
5 }
6
7 .btn{
8   @extend %btn;
9   text-decoration: none;
10  background: black;
11  color: white;
12  text-transform: uppercase;
13
14  &:hover{
15    background: white;
16    color: black;
17  }
18 }
19
20 .btn-danger{
21   @extend .btn;
22   background: red;
23   &:hover{
24     background: yellow;
25     color: black;
26   }
27 }
28
29
30
```

```
1 .btn, .btn-danger {
2   text-decoration: none;
3   padding: 10px 20px;
4   background: black;
5   color: white;
6   text-transform: uppercase;
7   font-size: 20px;
8   font-weight: bold;
9 }
10
11 .btn:hover, .btn-danger:hover {
12   background: white;
13   color: black;
14 }
15
16 .btn-danger {
17   background: red;
18 }
19
20 .btn-danger:hover {
21   background: yellow;
22   color: black;
23 }
```

```
Windows PowerShell
PS C:\Users\Nicolas\Desktop\scss>
```

Avec SASS, il est possible de créer des éléments “fantômes”, c’est à dire qui ne seront jamais visibles coté CSS mais réutilisables comme un sélecteur normal. Ils s’écrivent avec le symbole “%” juste avant.

# SASS - Les variables

Il y a un peu de temps, les variables SASS étaient très (très) utiles car elles n'existaient pas en CSS.

Mais de nos jours, elles existent. Cependant, les variables SASS offrent plus de possibilités.

```
html{  
  --main-color: #0F363A;  
}  
  
.container div a{  
  color: var(--main-color);  
}
```


# SASS - Les variables

```
$main-color: #0F363A;

%btn{
  padding: 10px 20px;
  font-size: 20px;
  font-weight: bold;
}

.btn{
  @extend %btn;
  text-decoration: none;
  background: black;
  color: $main-color;
  text-transform: uppercase;
}

1 .btn, .btn-danger {
2   padding: 10px 20px;
3   font-size: 20px;
4   font-weight: bold;
5 }
6
7 .btn, .btn-danger {
8   text-decoration: none;
9   background: black;
10  color: #0F363A;
11  text-transform: uppercase;
12 }
13 .btn:hover, .btn-danger:hover {
14   background: white;
15   color: black;
16 }
```



Les variables SASS commencent par le symbole “\$”  
(comme php).

EX : \$main-color: #0F363A;

Pour utiliser cette variable, color: \$main-color;


# SASS - Les variables

```
$main-color: #0F363A;

%btn{
  padding: 10px 20px;
  font-size: 20px;
  font-weight: bold;
}

.btn{
  @extend %btn;
  text-decoration: none;
  background: black;
  color: $main-color;
  text-transform: uppercase;
}

1 .btn, .btn-danger {
2   padding: 10px 20px;
3   font-size: 20px;
4   font-weight: bold;
5 }
6
7 .btn, .btn-danger {
8   text-decoration: none;
9   background: black;
10  color: #0F363A;
11  text-transform: uppercase;
12 }
13 .btn:hover, .btn-danger:hover {
14   background: white;
15   color: black;
16 }
```



L'avantage par rapport aux variables CSS, c'est la compatibilité avec les navigateurs (ils ne comprennent pas tous les variables CSS). Comme SASS génère du CSS, il n'y a plus de problème à ce niveau.

# SASS - Les variables

```
$main-color: #0F363A;  
$padding: 10px;  
  
%btn{  
  padding: $padding $padding*2;  
  font-size: 20px;  
  font-weight: bold;  
}
```

```
1  
2  
3 .btn, .btn-danger {  
4   padding: 10px 20px;  
5   font-size: 20px;  
6   font-weight: bold;  
7 }  
8
```

Un autre avantage par rapport aux variables CSS, c'est sa syntaxe plus simple pour certaines opérations.

# SASS - Les variables

```
html{  
  --padding: 10px;  
}  
  
.btn, .btn-danger {  
  padding: var(--padding) calc(var(--padding) *2);  
  font-size: 20px;  
  font-weight: bold;  
}
```

La même chose avec CSS....



# SASS - Les variables

```

$main-color: #0F363A;
$padding: 10px;

%btn{
  padding: $padding $padding + 2rem;
  font-size: 20px;
  font-weight: bold;
}

.btn{
  @extend %btn;
  text-decoration: none;
  background: black;
}


```

 Windows PowerShell

```
PS C:\Users\Nicolas\Desktop\scss> sass style.scss style.css
PS C:\Users\Nicolas\Desktop\scss> sass style.scss style.css
PS C:\Users\Nicolas\Desktop\scss> sass style.scss style.css
PS C:\Users\Nicolas\Desktop\scss> sass style.scss style.css
Error: 10px and 2rem have incompatible units.
```

```
padding: $padding $padding + 2rem
AAAAAAAAAAAAAAAAAAAA
```

```
style.scss 5:20  root stylesheet
```

```
PS C:\Users\Nicolas\Desktop\scss>
```

Attention quand vous additionnez des unités, si ce ne sont pas les mêmes, il va vous renvoyer une erreur.

# SASS - Les variables

```
$main-color: #0F363A;
$padding: 10;

%btn{
  padding: $padding + 0px $padding + 2rem;
  font-size: 20px;
  font-weight: bold;
}

.btn{
  @extend %btn;
```

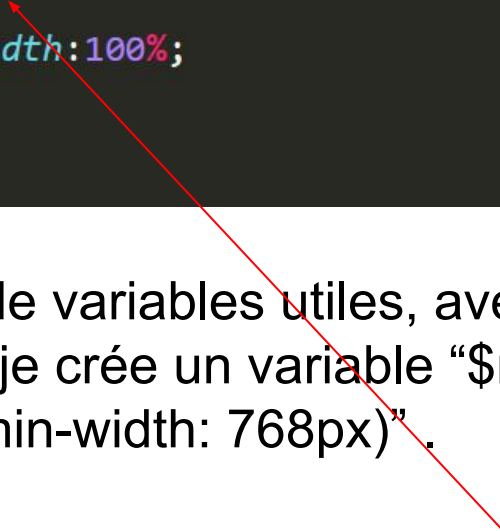
```
1 .btn, .btn-danger {
2   padding: 10px 12rem;
3   font-size: 20px;
4   font-weight: bold;
5 }
6
7 .btn, .btn-danger {
8   text-decoration: none;
9   background: black;
10  color: #0F363A;
11  text-transform: uppercase;
```

Si une valeur n'a pas d'unité, elle prendra la valeur de celle qui en a une lors d'une opération.

# SASS - Les variables

```
$md: "only screen and (min-width: 768px)";

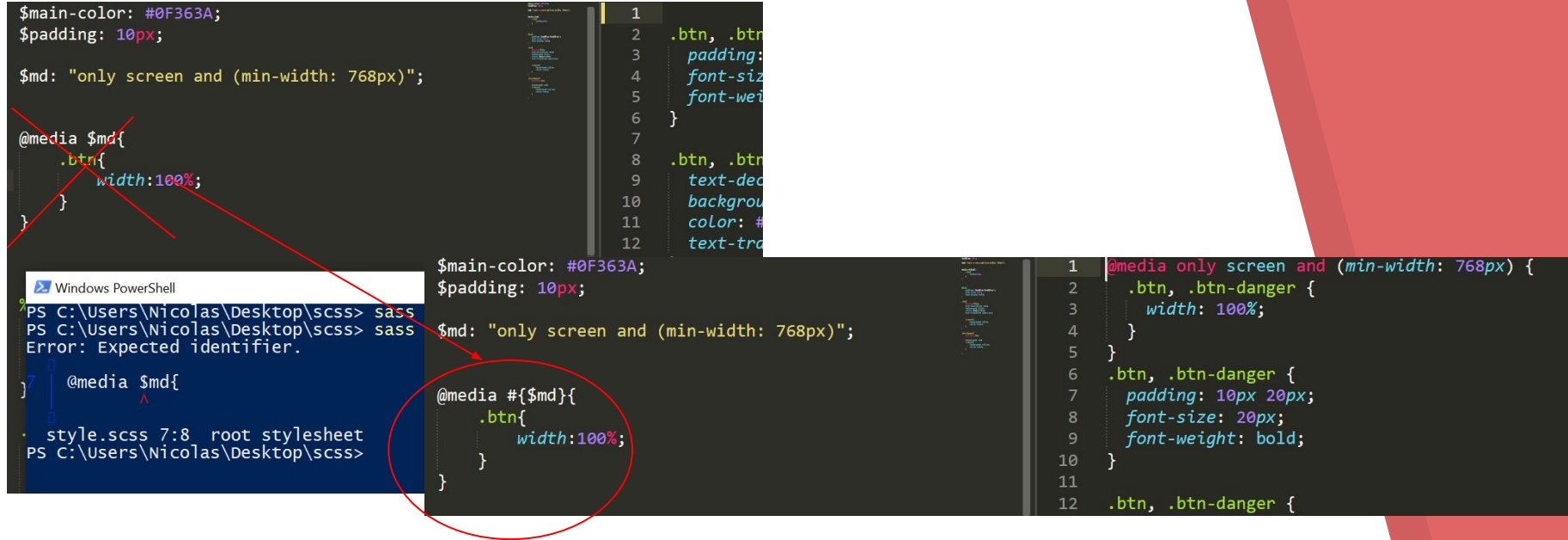
@media $md{
  .btn{
    width:100%;
  }
}
```



Un autre exemple de variables utiles, avec les media queries. Dans cet exemple, je crée un variable “\$md” qui est égale à “only screen and (min-width: 768px)”.

J'appelle cette variables juste en dessous....

# SASS - Les variables



```
$main-color: #0F363A;
$padding: 10px;

$md: "only screen and (min-width: 768px)";

@media $md{
  .btn{
    width:100%;
  }
}
```

Windows PowerShell

```
PS C:\Users\Nicolas\Desktop\scss> sass
PS C:\Users\Nicolas\Desktop\scss> sass
Error: Expected identifier.

7 | @media $md{
  |         ^
  | style.scss 7:8 root stylesheet
PS C:\Users\Nicolas\Desktop\scss>
```

```
$main-color: #0F363A;
$padding: 10px;

$md: "only screen and (min-width: 768px)";

@media #{ $md }{
  .btn{
    width:100%;
  }
}
```

```
1 | @media only screen and (min-width: 768px) {
2 |   .btn, .btn-danger {
3 |     width: 100%;
4 |   }
5 | }
6 | .btn, .btn-danger {
7 |   padding: 10px 20px;
8 |   font-size: 20px;
9 |   font-weight: bold;
10 | }
11 |
12 | .btn, .btn-danger {
```

L'intention est louable (et pratique) mais SASS n'aime pas trop insérer des variables en tant que sélecteur. Pour que ça fonctionne, il faut mettre la variable entre `#{ $ma_variable }`

# SASS - Les variables

```
$medium: "768px";  
$md: "only screen and (min-width: #{ $medium })";  
  
@media #{ $md }{  
  .btn{  
    width: 100%;  
  }  
}
```

Je peux même être encore plus flexible en créant des variables avec les valeurs de “breakpoints”.

# SASS - Les variables

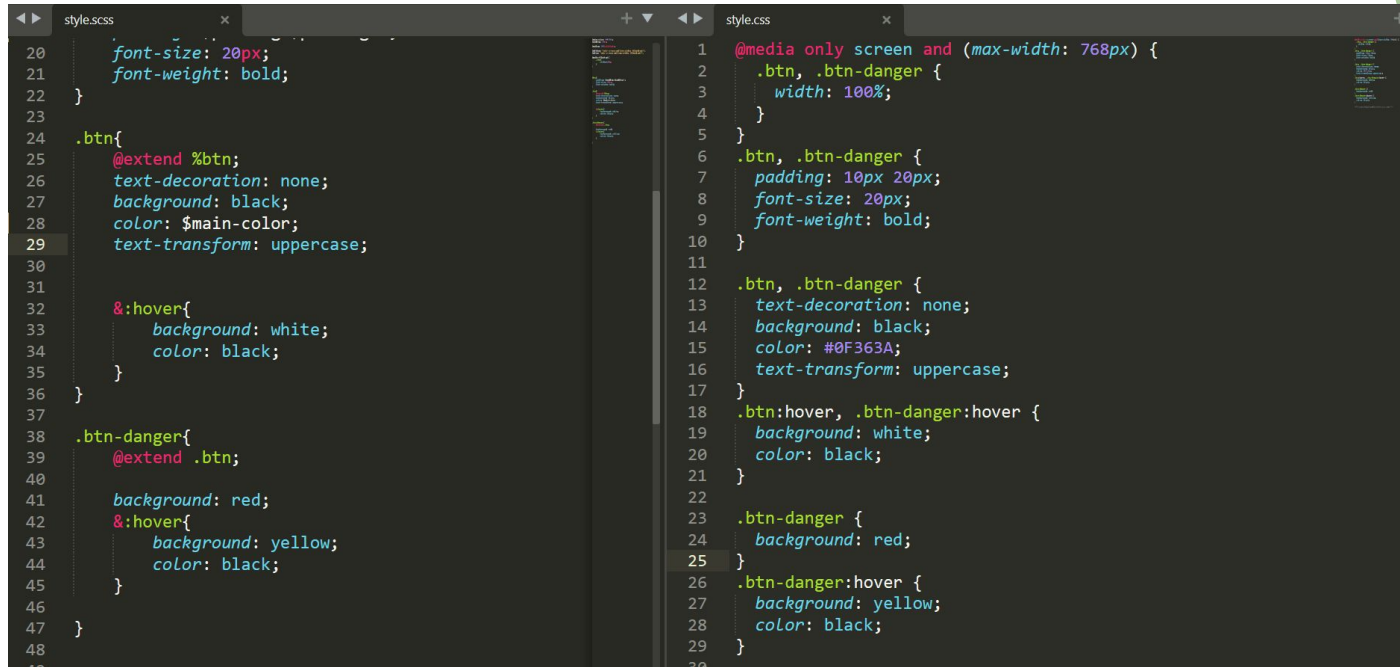
```
$medium: 768px!default;

$md-down: "only screen and (min-width: #{$medium})";
$md-up: "only screen and (max-width: #{$medium})";

@media #{$md-up}{
  .btn{
    width:100%;
  }
}
```

Et si je veux être encore plus précis....

# SASS - Les variables



The image shows a side-by-side comparison of SASS and CSS code for button styles. The left pane, titled 'style.scss', contains SASS code with variables like \$main-color and mixins like @extend. The right pane, titled 'style.css', shows the compiled CSS output where variables are replaced with their values and mixins are expanded into standard CSS rules. The code defines styles for a base button class (.btn) and a danger button class (.btn-danger), including font settings, background colors, and hover states.

```
20 font-size: 20px;
21 font-weight: bold;
22 }
23
24 .btn{
25   @extend %btn;
26   text-decoration: none;
27   background: black;
28   color: $main-color;
29   text-transform: uppercase;
30
31   &:hover{
32     background: white;
33     color: black;
34   }
35 }
36
37 .btn-danger{
38   @extend .btn;
39
40   background: red;
41   &:hover{
42     background: yellow;
43     color: black;
44   }
45 }
46
47 }
48
```

```
1 @media only screen and (max-width: 768px) {
2   .btn, .btn-danger {
3     width: 100%;
4   }
5 }
6
7 .btn, .btn-danger {
8   padding: 10px 20px;
9   font-size: 20px;
10  font-weight: bold;
11 }
12
13 .btn, .btn-danger {
14   text-decoration: none;
15   background: black;
16   color: #0F363A;
17   text-transform: uppercase;
18 }
19
20 .btn:hover, .btn-danger:hover {
21   background: white;
22   color: black;
23 }
24
25 .btn-danger {
26   background: red;
27 }
28
29 .btn-danger:hover {
30   background: yellow;
31   color: black;
32 }
```

Un truc cool avec SASS, la conversion RGBA

# SASS - Import

```
$main-color: #0F363A;  
$padding: 10px;  
  
$medium: 768px!default;  
  
$md-down: "only screen and (min-width: #{ $medium })";  
$md-up: "only screen and (max-width: #{ $medium })";  
  
@media #{ $md-up }{  
  .btn{  
    width: 100%;  
  }  
}
```

responsive.scss

```
1 $medium: 768px!default;  
2  
3 $md-down: "only screen and (min-width: #{ $medium })";  
4 $md-up: "only screen and (max-width: #{ $medium })";
```

```
@import "libs/responsive";
```

```
$main-color: #0F363A;  
$padding: 10px;
```

Pour une meilleure organisation du code, il est possible de le découper dans plusieurs fichiers et de les importer (@import) dans le dossier principal.



# SASS - Import

The diagram illustrates the SASS import functionality. It shows a file explorer with a folder named **atomes** and a file named **style.scss**. A red arrow points from **style.scss** to a code editor showing the content:

```
@charset 'utf-8';  
@import 'atomes';  
/* NE RIEN RAJOUTER => L
```

Another red arrow points from the **atomes** folder to a list of SCSS files in a sidebar:

- \_404.scss
- \_accueil.scss
- \_animated\_elements.scss
- \_atomes.scss
- \_base.scss
- \_color.scss
- \_contact.scss
- \_cursor.scss
- \_equipe.scss
- \_helper.scss
- \_index.scss
- \_loader.scss
- \_manifeste.scss
- \_menu.scss
- bootstrap\_modifier.scss

Each file in the sidebar is associated with a date and time, the file type (Fichier SCSS), and its size in Ko.

File	Date	Type	Size
_404.scss	08/07/2020 13:08	Fichier SCSS	1 Ko
_accueil.scss	21/09/2021 21:20	Fichier SCSS	1 Ko
_animated_elements.scss	08/07/2020 13:08	Fichier SCSS	1 Ko
_atomes.scss	14/10/2021 11:40	Fichier SCSS	15 Ko
_base.scss	25/06/2020 11:42	Fichier SCSS	3 Ko
_color.scss	25/06/2020 11:42	Fichier SCSS	2 Ko
_contact.scss	10/09/2021 23:25	Fichier SCSS	2 Ko
_cursor.scss	27/09/2021 14:25	Fichier SCSS	2 Ko
_equipe.scss	25/06/2020 11:42	Fichier SCSS	1 Ko
_helper.scss	25/06/2020 11:42	Fichier SCSS	1 Ko
_index.scss	08/07/2020 13:08	Fichier SCSS	1 Ko
_loader.scss	08/07/2020 13:08	Fichier SCSS	1 Ko
_manifeste.scss	25/06/2020 11:42	Fichier SCSS	16 Ko
_menu.scss	21/09/2021 19:20	Fichier SCSS	4 Ko
bootstrap_modifier.scss	25/06/2020 11:42	Fichier SCSS	1 Ko

Il est possible d'importer le contenu d'un fichier (scss, sass,css) ou même un dossier entier (grace au fichier index.scss) ! Au final SASS va créer un seul fichier CSS. Pratique !

# SASS - mixins



```
style.scss  x  colors.scss  x  mixins.scss  x  style.css  x  responsive.
@import "libs/responsive";
@import "libs/mixins";

1  @mixin rotate10{
2      transform: rotate(10deg);
3  }
4
5
```

Une mixin commence par “@mixin” et ensuite le nom que vous voulez lui donner. @mixin rotate10 (je vous cache pas que c’est pour faire une rotation de 10 degrés....). Enfin, on ouvre les accolades pour mettre le code à exécuter.

# SASS - mixins

```
.btn{  
  @include rotate10;  
  @extend %btn;  
  text-decoration: none;  
  background: black;  
  color: rgba($main-color,.5);  
  text-transform: uppercase;  
  
  &:hover{
```

```
8      font-size: 20px;  
9      font-weight: bold;  
10   }  
11  
12   .btn, .btn-danger {  
13     transform: rotate(10deg);  
14     text-decoration: none;  
15     background: black;  
16     color: rgba(15, 54, 58, 0.5);  
17     text-transform: uppercase;  
18   }
```

Et pour l'appeler, il faut faire “@include” et le nom de la mixin à appeler.  
Et là, vous vous dites, mais c'est idiot, c'est comme le “@extends”.....  
Oui.....MAIIIIIIIIIS.....

# SASS - mixins

```
.btn{
  @include rotate10;
  text-decoration: none;
  background: black;
  color: rgba($main-color,.5);
  text-transform: uppercase;

  &:hover{
    background: white;
    color: black;
  }
}

.btn-danger{
  @extend .btn;

  background: red;
  &:hover{
    background: yellow;
    color: black;
  }
}

1 @media only screen and (max-width: 768px) {
2   .btn, .btn-danger {
3     color: black;
4   }
5 }
6 .btn, .btn-danger {
7   transform: rotate(10deg);
8   text-decoration: none;
9   background: black;
10  color: rgba(15, 54, 58, 0.5);
11  text-transform: uppercase;
12 }
13 .btn:hover, .btn-danger:hover {
14   background: white;
15   color: black;
16 }
17
18 .btn-danger {
19   background: red;
20 }
21 .btn-danger:hover {
22   background: yellow;
23   color: black;
24 }
25
26 /*# sourceMappingURL=style.css.map */
27
```

Il y a 2 principales différences :

- La première est que le “@include” injecte du code à l’endroit souhaité alors que le “@extend” surcharge le CSS (il ajoute des sélecteurs un peu partout)

# SASS - mixins

```
@mixin rotate($rotation){  
  transform: rotate($rotation);  
}
```

```
.btn{  
  @include rotate(20deg);  
  text-decoration: none;  
  background: black;  
}  
  
.btn, .btn-danger {  
  transform: rotate(20deg);  
  text-decoration: none;
```

Il y a 2 principales différences :

- La deuxième c'est qu'il est possible de lui faire passer des paramètres...

EX: je veux faire une rotation mais je veux mettre une valeur à la volée que je décide.

# SASS - mixins

```
@mixin rotate($rotation:10deg){  
  transform: rotate($rotation);  
}
```

```
.btn{  
  @include rotate;  
  @include rotate(50deg);  
  text-decoration: none;  
  background: black;  
  color: rgba($main-color, .5);  
}  
  
.btn, .btn-danger {  
  transform: rotate(10deg);  
  transform: rotate(50deg);  
  text-decoration: none;  
}
```

D'ailleurs, il est possible de lui passer une valeur par défaut. En gros, si je mets aucune valeur ce sera 10 degrés, sinon, la valeur que je choisis.

# SASS - mixins

```
@mixin mq($size, $minmax:"max", $type:"screen"){  
  @media only #{ $type } and (#{ $minmax }-width: $size){  
    @content;  
  }  
}
```

```
@include mq($medium){  
  header{  
    background: black;  
  }  
}
```

```
@media only screen and (max-width: 768px) {  
  header {  
    background: black;  
  }  
}
```

Une petite mixin que j'aime beaucoup... qui gère les média queries (mq)

# SASS - Functions

```
@function rem($size , $base:16){  
  @return 1rem * calc($size / $base);  
}
```

Les mixins et les fonctions se ressemblent beaucoup. La différence c'est que la fonction va renvoyer une valeur alors que la mixin va renvoyer des règles CSS. La fonction a obligatoirement un “@return” alors que la mixin non.


La fonction commence par “@function”, puis son nom (comme la mixin), enfin entre accolades du code et un “@return” qui renvoie une valeur.



# SASS - Functions

```
text-transform: uppercase;  
font-size: rem(24);
```

```
11 color: rgba(15, 54, 58, 0.5);  
12 text-transform: uppercase;  
13 font-size: 1.5rem;  
14 }
```



Les fonctions s'appellent plus simplement que les mixins....

De plus, les fonctions ne peuvent s'appeler que comme valeur d'une propriété alors que les mixins comme règles CSS

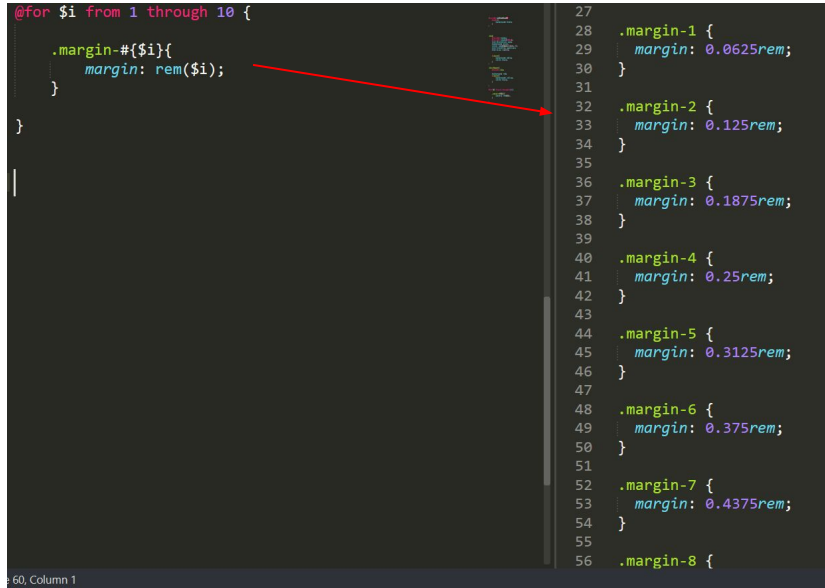
# SASS - Les boucles

```
@for $i from 1 to 13 {  
  
}
```

Voici la syntaxe de la boucle “@for”.

La variable \$i peut être renommée ainsi que la valeur de départ (1) et de fin (10). Entre les accolades, on met le code à dupliquer...10 fois pour le coup...

# SASS - Les boucles



The image shows a code editor with two panels. The left panel contains a SASS loop: `@for $i from 1 through 10 {  
 .margin-#{$i}{  
 margin: rem($i);  
 }  
}`. A red arrow points from the `rem($i);` line to the right panel. The right panel shows the expanded CSS output, listing classes `.margin-1` through `.margin-8` with their corresponding margin values in rem (e.g., `.margin-1 { margin: 0.0625rem; }`). The status bar at the bottom left of the editor indicates '60, Column 1'.

```
@for $i from 1 through 10 {  
  .margin-#{$i}{  
    margin: rem($i);  
  }  
}
```

```
27  
28 .margin-1 {  
29   margin: 0.0625rem;  
30 }  
31  
32 .margin-2 {  
33   margin: 0.125rem;  
34 }  
35  
36 .margin-3 {  
37   margin: 0.1875rem;  
38 }  
39  
40 .margin-4 {  
41   margin: 0.25rem;  
42 }  
43  
44 .margin-5 {  
45   margin: 0.3125rem;  
46 }  
47  
48 .margin-6 {  
49   margin: 0.375rem;  
50 }  
51  
52 .margin-7 {  
53   margin: 0.4375rem;  
54 }  
55  
56 .margin-8 {
```

Et voilà, comment une petite boucle “@for” génère plusieurs lignes de CSS.

# SASS - Les boucles

```
@each $variable in $listDeValeurs {  
  
}
```

Et ça, c'est la boucle `@each`. Cette boucle n'a ni besoin de début ni de limite de fin, juste d'une liste à parcourir.

# SASS - Les boucles

```
$colors: blue red orange;  
  
@each $color in $colors {  
  
}
```

Donc là, nous allons extraire chaque couleur (\$color) de la liste (\$colors).

Dans cet exemple, seul “\$color” est modifiable car “\$colors” correspond au nom de la liste.

# SASS - Les boucles

```
$colors: blue red orange;  
  
@each $color in $colors {  
  .color-#{$color}{  
    color: $color;  
  }  
}
```

```
30  
31 .color-blue {  
32   color: blue;  
33 }  
34  
35 .color-red {  
36   color: red;  
37 }  
38  
39 .color-orange {  
40   color: orange;  
41 }  
42  
43  
44
```

TADA !

# SASS - Les boucles

```
$categories:  
  plane #fff,  
  boat #000,  
  car #FF0000;  
  
@each $vehicule, $color in $categories {  
  
}
```

Il est possible d'avoir des listes un peu plus complexes...  
Après mon “@each”, j’ai 2 variables (\$vehicule et \$color) qui correspondent aux 2 “colonnes” de ma liste.

# SASS - Les boucles

```
$categories:  
  plane #fff,  
  boat #000,  
  car #FF0000;  
  
@each $vehicule, $color in $categories {  
  .#{$vehicule}{  
    color: $color;  
  }  
}
```

```
35 .plane {  
36   color: #fff;  
37 }  
38  
39 .boat {  
40   color: #000;  
41 }  
42  
43 .car {  
44   color: #FF0000;  
45 }  
46  
47
```

(Re) TADA !



# SASS - Les boucles

```
$categories:
  plane #fff yy,
  boat #000 uu,
  car #FF0000 ii;

@each $vehicule, $color, $1 in $categories {
  .#{$vehicule}-#{ $1 }{
    color: $color;
  }
}
```

```
30
31
32 .plane-yy {
33   color: #fff;
34 }
35
36 .boat-uu {
37   color: #000;
38 }
39
40 .car-ii {
41   color: #FF0000;
42 }
43
44
45
```

The image shows a SASS code editor with two panels. The left panel contains the source SASS code. A red circle highlights the list of categories: `plane #fff yy, boat #000 uu, car #FF0000 ii;`. Another red circle highlights the loop variable `$1` in the `@each` loop. A red arrow points from this `$1` to the right panel. The right panel shows the compiled CSS output. A red circle highlights the generated class names: `.plane-yy`, `.boat-uu`, and `.car-ii`, demonstrating how the loop variable `$1` is interpolated into the class name.

Si je rajoute une “colonne”, je rajoute une variable dans ma boucle  
“@each”

# SASS - Les boucles

```
$sm: 576px!default;  
$md: 768px!default;  
$lg: 992px!default;  
$xl: 1200px!default;  
$xxl: 1400px!default;
```

```
$prefixes:
```

```
  sm $sm,  
  md $md,  
  lg $lg,  
  xl $xl,  
  xxl $xxl;
```

```
$classes: row, column, row-reverse, column-reverse;
```

Attention, ca va piquer un peu.... J'ai créé une liste "prefixe valeur" (\$prefixes) et une autre liste de valeur "flex" (que j'ai appelé \$classes, je sais pas pourquoi...en même temps il est minuit neuf donc bon.....)

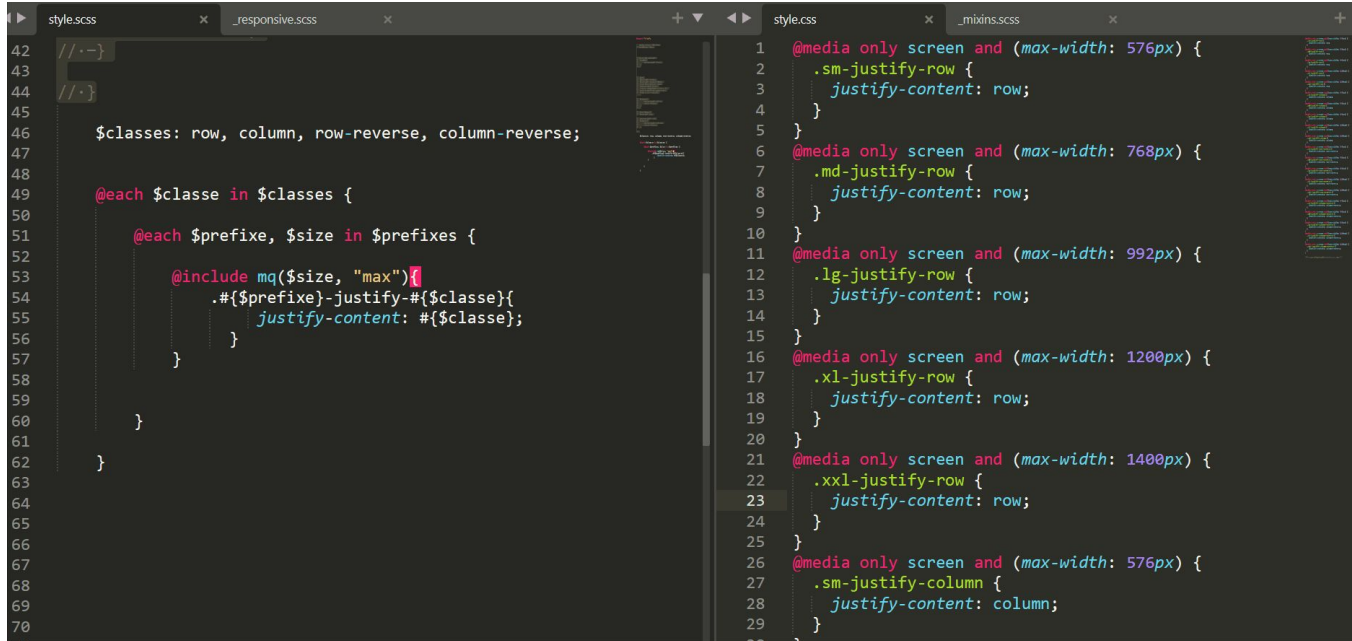
# SASS - Les boucles

```
$classes: row, column, row-reverse, column-reverse;

@each $classe in $classes {
  @each $prefixe, $size in $prefixes {
    @include mq($size, "max"){
      .#{$prefixe}-justify-#{$classe}{
        justify-content: #{$classe};
      }
    }
  }
}
```

Et voici 2 boucles “@each” imbriquées qui appelle une mixin pour générer des dizaines de lignes CSS rapidement...

# SASS - Les boucles



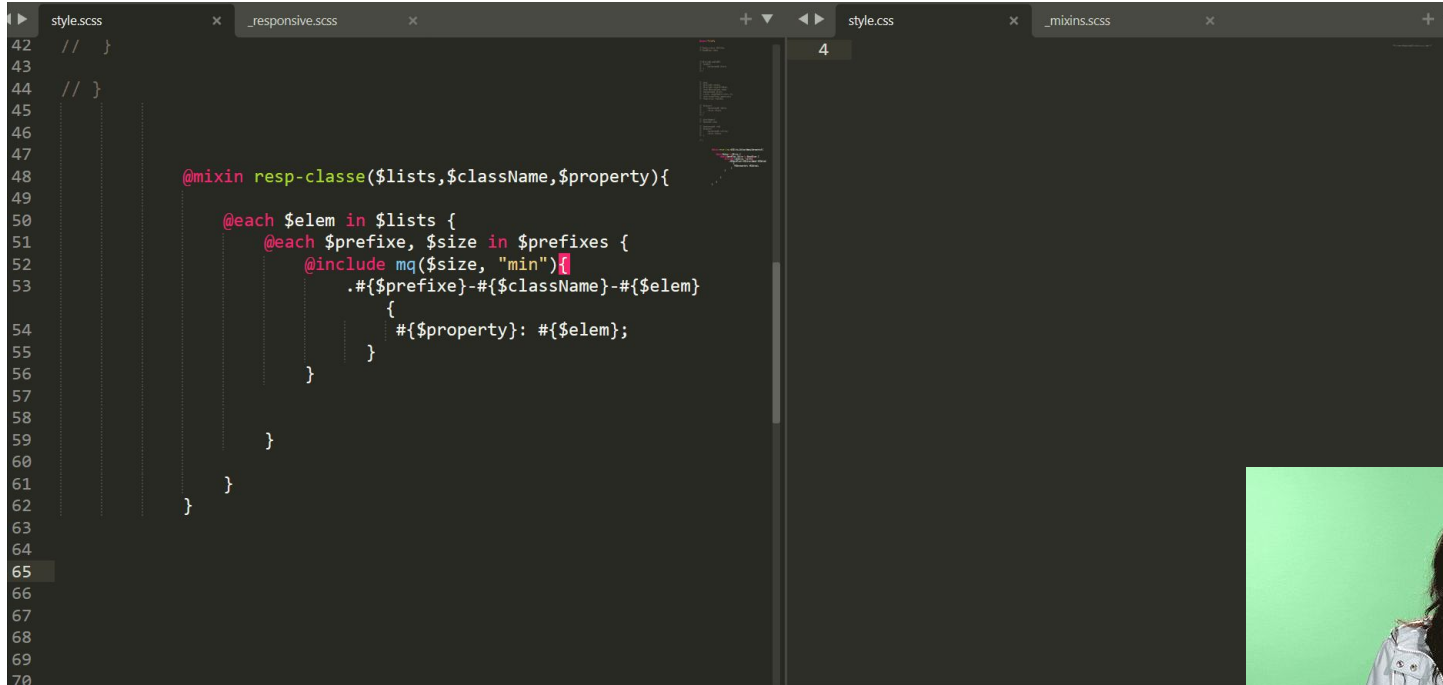
The image shows a code editor with two files open: `style.scss` and `_mixins.scss`. The `style.scss` file on the left uses a loop to generate styles for different classes and prefixes. The `_mixins.scss` file on the right uses media queries to generate styles for different screen widths.

```
42 //.-}
43 //.-}
44 //.-}
45
46 $classes: row, column, row-reverse, column-reverse;
47
48 @each $classe in $classes {
49   @each $prefixe, $size in $prefixes {
50     @include mq($size, "max") {
51       .#{$prefixe}-justify-#{$classe} {
52         justify-content: #{$classe};
53       }
54     }
55   }
56 }
57
58 }
59
60 }
61
62 }
63
64 }
65
66 }
67
68 }
69
70 }
```

```
1 @media only screen and (max-width: 576px) {
2   .sm-justify-row {
3     justify-content: row;
4   }
5 }
6 @media only screen and (max-width: 768px) {
7   .md-justify-row {
8     justify-content: row;
9   }
10 }
11 @media only screen and (max-width: 992px) {
12   .lg-justify-row {
13     justify-content: row;
14   }
15 }
16 @media only screen and (max-width: 1200px) {
17   .xl-justify-row {
18     justify-content: row;
19   }
20 }
21 @media only screen and (max-width: 1400px) {
22   .xxl-justify-row {
23     justify-content: row;
24   }
25 }
26 @media only screen and (max-width: 576px) {
27   .sm-justify-column {
28     justify-content: column;
29   }
30 }
```

TADADADADA !

# SASS - Les boucles



```
42 // }
43
44 // }
45
46
47
48 @mixin resp-classe($lists,$className,$property){
49
50     @each $elem in $lists {
51         @each $prefixe, $size in $prefixes {
52             @include mq($size, "min"){
53                 .#{$prefixe}-#{$className}-#{$elem}
54                 {
55                     #{$property}: #{$elem};
56                 }
57             }
58         }
59     }
60 }
61
62 }
```



Et là, je fais une mixin de tout le code qui me génère aléatoirement des classes utilitaires de tous les formats d'écran !

# SASS - Les boucles

```
@include resp-classe([row, column, row-reverse,  
column-reverse],justify,justify-content);
```

Ma mixin s'appelle "resp-classe" et en tant que mixin je dois faire un "@include" pour l'appeler.

[row, column, row-reverse, column-reverse] : la liste des valeurs

Justify : Le nom que je choisis pour ma classe (j'aurais pu l'appeler billy ou pateEnCroute)

Justify-content : le nom de la propriété CSS