LES PROCESSEURS CSS

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

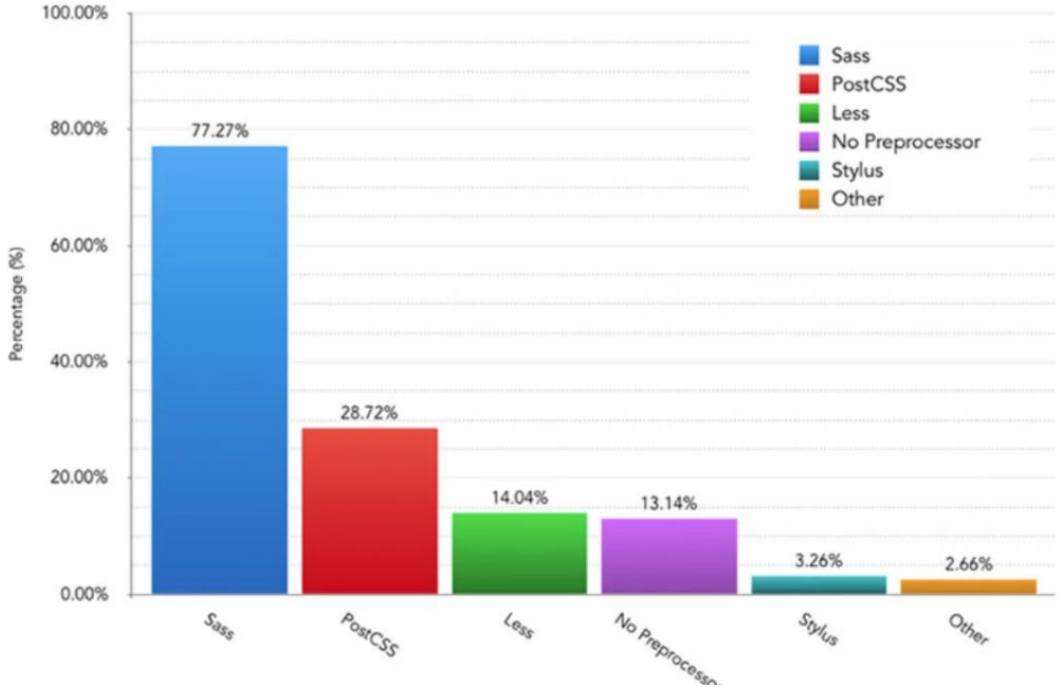
Les préprocesseurs - C’est quoi ?

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 ?

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.

```scss
$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 :

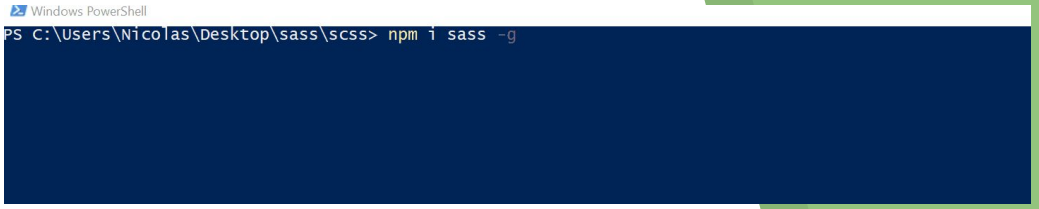
- Installer NodeJS (https://nodejs.org/en/). Prendre la version LTS (plus stable)

(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 :



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**”

.scss

```scss
.button {
    background: cornflowerblue;
    border-radius: 5px;
    padding: 10px 20px;

    &:hover {
        cursor: pointer;
    }

    &:disabled {
        cursor: default;
        background: grey;
        pointer-events: none;
    }
}
```

.sass

```sass
.button
    background: cornflowerblue
    border-radius: 5px
    padding: 10px 20px

    &:hover
        cursor: pointer

    &:disabled
        cursor: default
        background: grey
        pointer-events: none
```

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 !

Les préprocesseurs - SASS

Générer du CSS via SASS :

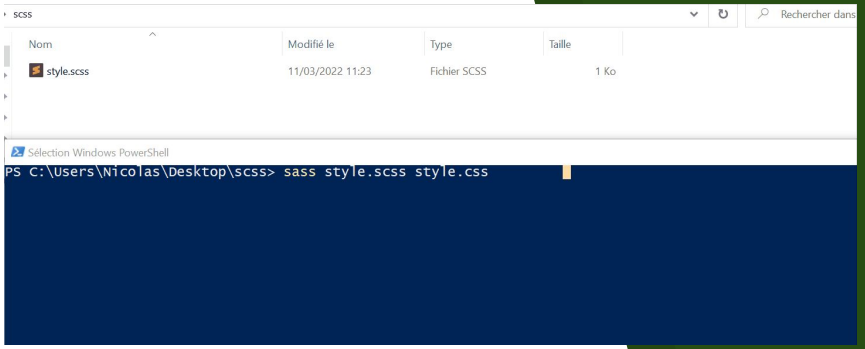
- Ecrivez quelque chose dans votre fichier style.scss :

```
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 :



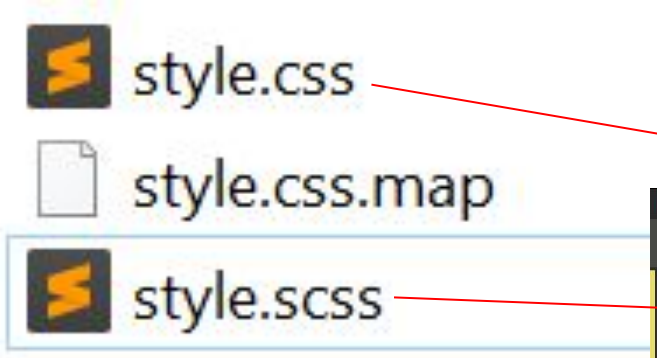
Sass style.scss style.css

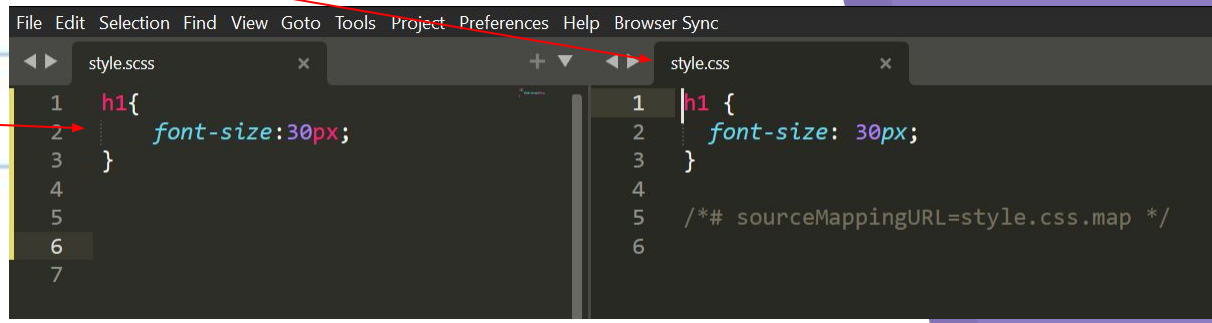
On dit qu'on va faire appel à SASS

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 ?????

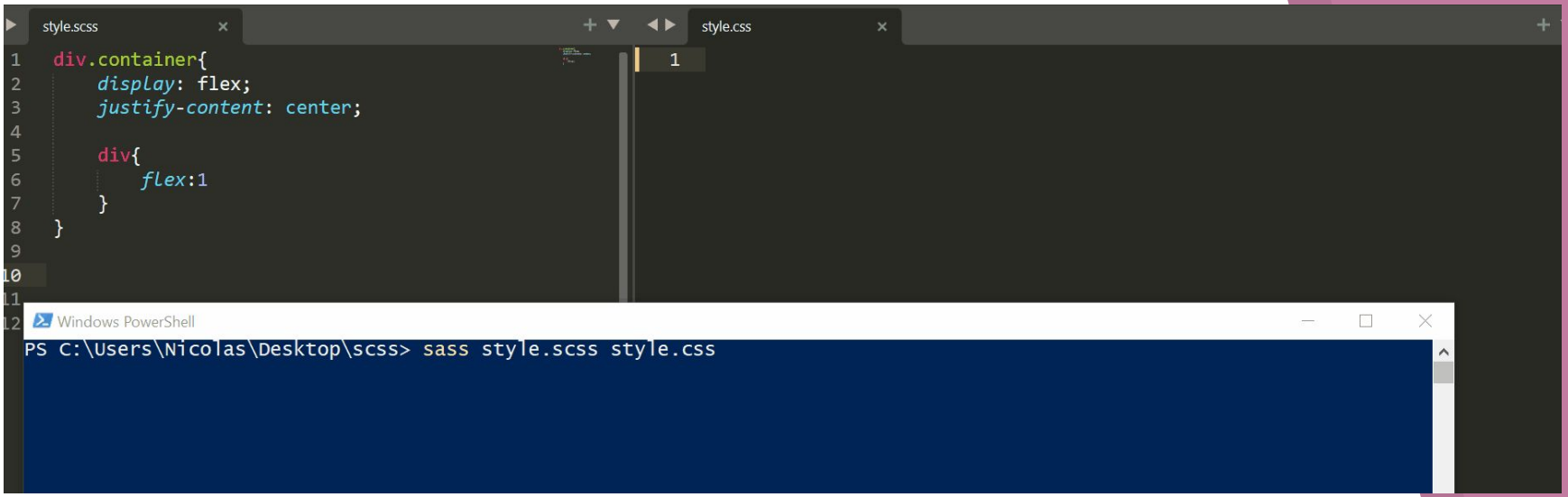
Les préprocesseurs - SASS

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 vrai PUISSANCE de SASS !!!!
Nous allons voir l'imbrication, les variables, les mixins, les fonctions, les boucles.

SASS -L’imbrication (nesting)



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)

style.scss

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

    div{
        flex:1;

        &:hover{
            color: white;
        }
    }
}
```

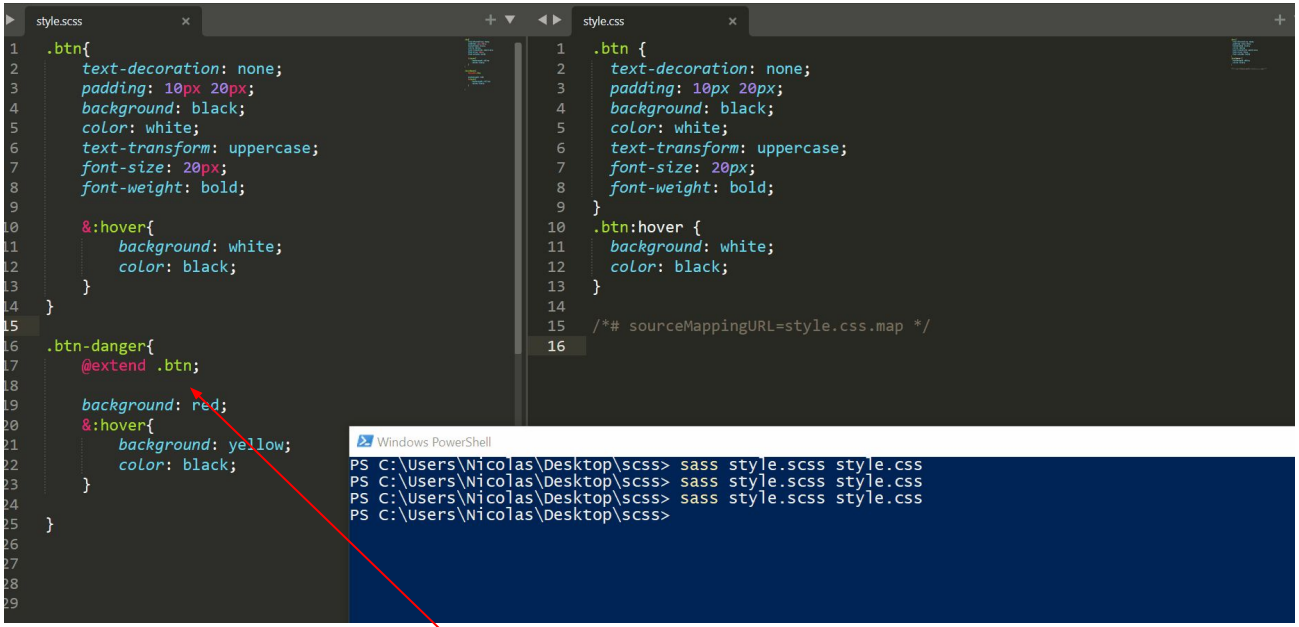
style.css

```css
div.container {
  display: flex;
  justify-content: center;
}
div.container div {
  flex: 1;
}
div.container div:hover {
  color: white;
}

/*# sourceMappingURL=style.css.map */
```

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

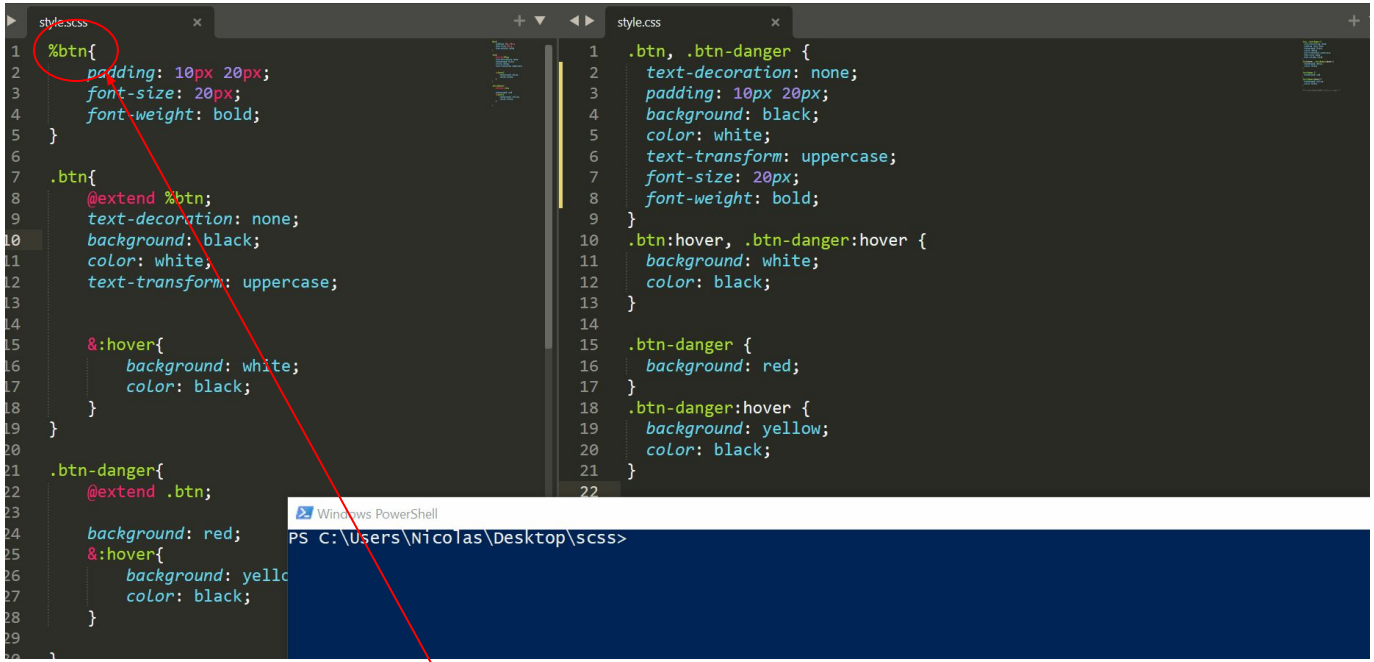
L’imbrication peut avoir ses limites car trop de niveaux d’imbrication peut rendre le code illisible. Je vous conseille 3 niveaux d’imbrication MAX

SASS -L’héritage (@extend)



@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)



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 possibilitées.

```
html{
  --main-color: #0F363A;
}


.container div a{
  color:  var(--main-color);
}
```

SASS - Les variables

```scss
$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;
}
```

```scss
.btn, .btn-danger {
  padding: 10px 20px;
  font-size: 20px;
  font-weight: bold;
}

.btn, .btn-danger {
  text-decoration: none;
  background: black;
  color: #0F363A;
  text-transform: uppercase;
}
.btn:hover, .btn-danger:hover {
  background: white;
  color: black;
}
```

Les variables SASS commencent par le symbole “$” (comme php).
EX : $main-color: #0F363A;
Pour utiliser cette variable, color: $main-color;

SASS - Les variables

```scss
$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;
}
```

```css
.btn, .btn-danger {
  padding: 10px 20px;
  font-size: 20px;
  font-weight: bold;
}

.btn, .btn-danger {
  text-decoration: none;
  background: black;
  color: #0F363A;
  text-transform: uppercase;
}
.btn:hover, .btn-danger:hover {
  background: white;
  color: black;
}
```

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

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

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

```scss

.btn, .btn-danger {
  padding: 10px 20px;
  font-size: 20px;
  font-weight: bold;
}

```

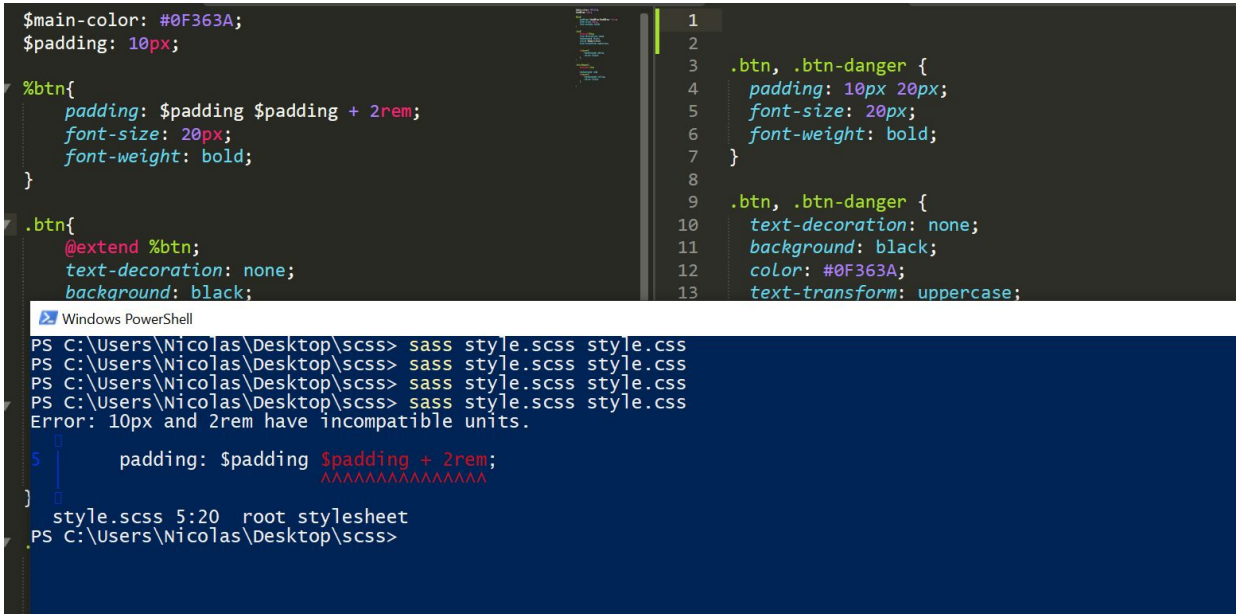
Un autre avantage par rapport aux variables CSS, c’est sa syntaxe plus simple pour certaines opérations.

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



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

SASS - Les variables

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

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

.btn{
    @extend %btn;
```

```css
.btn, .btn-danger {
  padding: 10px 12rem;
  font-size: 20px;
  font-weight: bold;
}

.btn, .btn-danger {
  text-decoration: none;
  background: black;
  color: #0F363A;
  text-transform: uppercase;
```

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

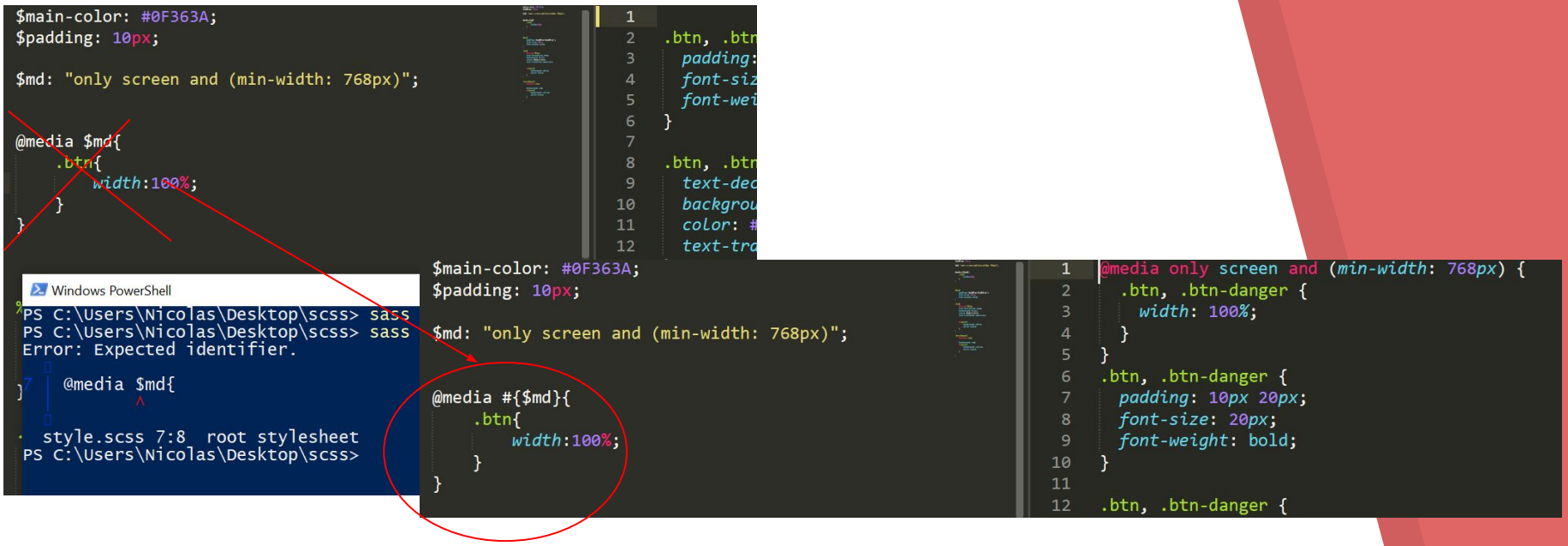
SASS - Les variables

```
$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



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

style.scss

```scss
    font-size: 20px;
    font-weight: bold;
}

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


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

.btn-danger{
    @extend .btn;

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

}
```

style.css

```css
@media only screen and (max-width: 768px) {
  .btn, .btn-danger {
    width: 100%;
  }
}
.btn, .btn-danger {
  padding: 10px 20px;
  font-size: 20px;
  font-weight: bold;
}

.btn, .btn-danger {
  text-decoration: none;
  background: black;
  color: #0F363A;
  text-transform: uppercase;
}
.btn:hover, .btn-danger:hover {
  background: white;
  color: black;
}

.btn-danger {
  background: red;
}
.btn-danger:hover {
  background: yellow;
  color: black;
}
```

Un truc cool avec SASS, la conversion RGBA

SASS - Import

```scss
$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%;
    }
}
```

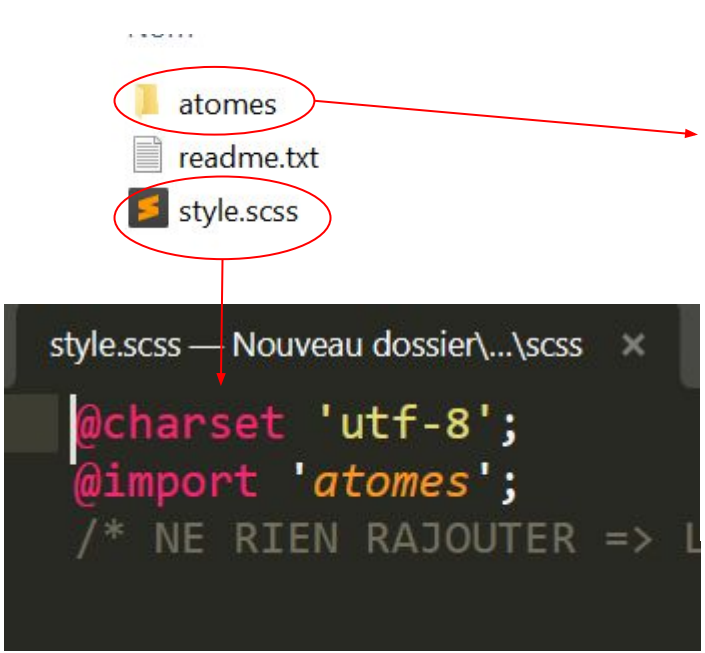
responsive.scss

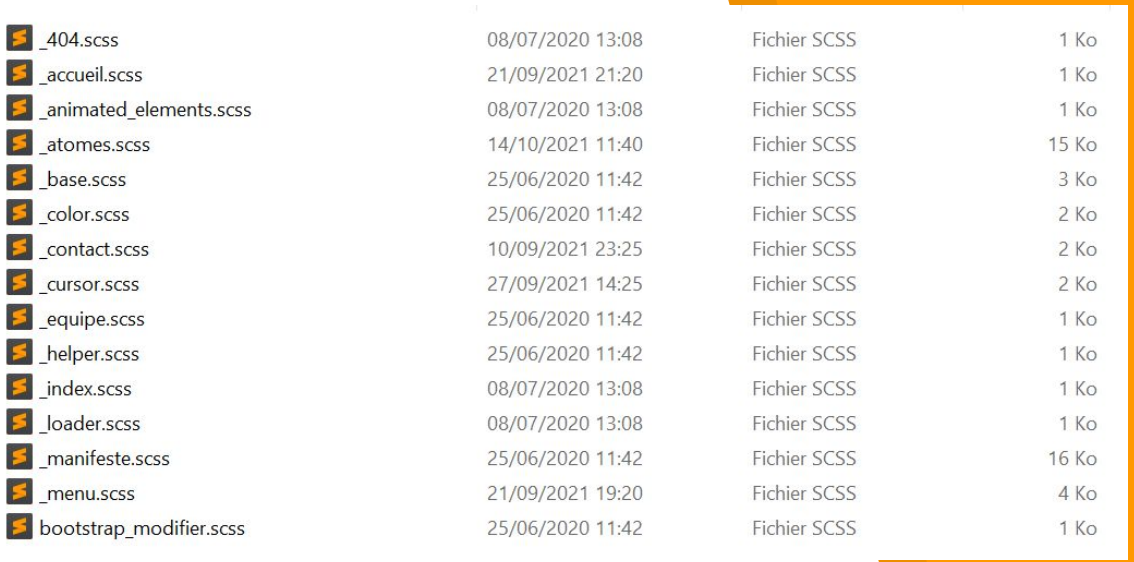
```scss
1  $medium: 768px!default;
2
3  $md-down: "only screen and (min-width: #{$medium})";
4  $md-up: "only screen and (max-width: #{$medium})";
```

```scss
@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





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

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{
```

```
    font-weight: bold;
}

.btn, .btn-danger {
    transform: rotate(10deg);
    text-decoration: none;
    background: black;
    color: rgba(15, 54, 58, 0.5);
    text-transform: uppercase;
}
```

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;
    }

}
```

```
@media only screen and (max-width: 768px) {
  .btn, .btn-danger {
    color: black;
  }
}
.btn, .btn-danger {
  transform: rotate(10deg);
  text-decoration: none;
  background: black;
  color: rgba(15, 54, 58, 0.5);
  text-transform: uppercase;
}
.btn:hover, .btn-danger:hover {
  background: white;
  color: black;
}

.btn-danger {
  background: red;
}
.btn-danger:hover {
  background: yellow;
  color: black;
}

/*# sourceMappingURL=style.css.map */

```

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

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

```scss
.btn{
    @include rotate(20deg);
    text-decoration: none;
    background: black;
```

```scss
}
.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

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

```scss
.btn{
    @include rotate;
    @include rotate(50deg);
    text-decoration: none;
    background: black;
```

```scss
    }
}
.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

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

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

```css
@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 functions se ressemble beaucoup. La différence c'est que la function va renvoyer une valeur alors que la mixin va renvoyer des règles CSS. La function a obligatoirement un "@return" alors que la mixin non.

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

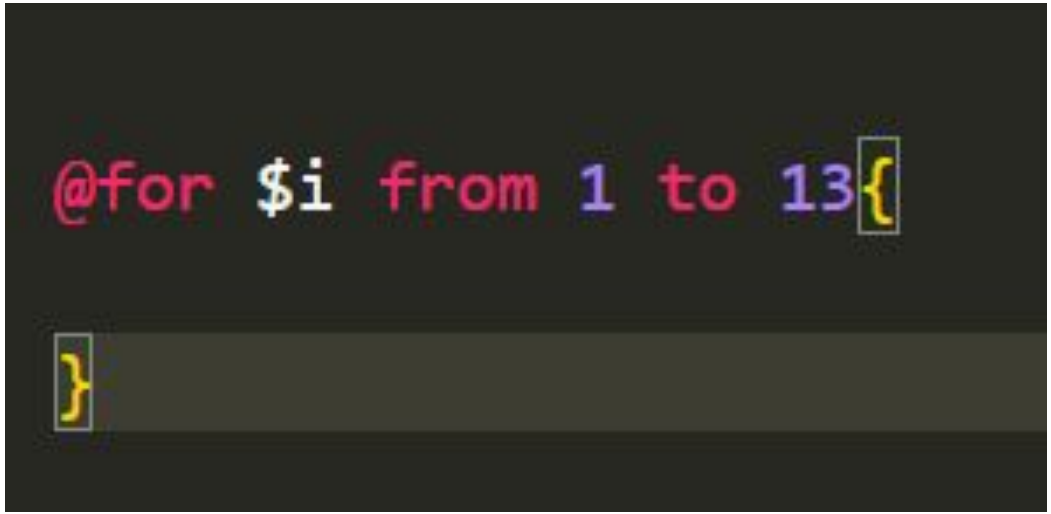
SASS - Functions

```
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  }
```

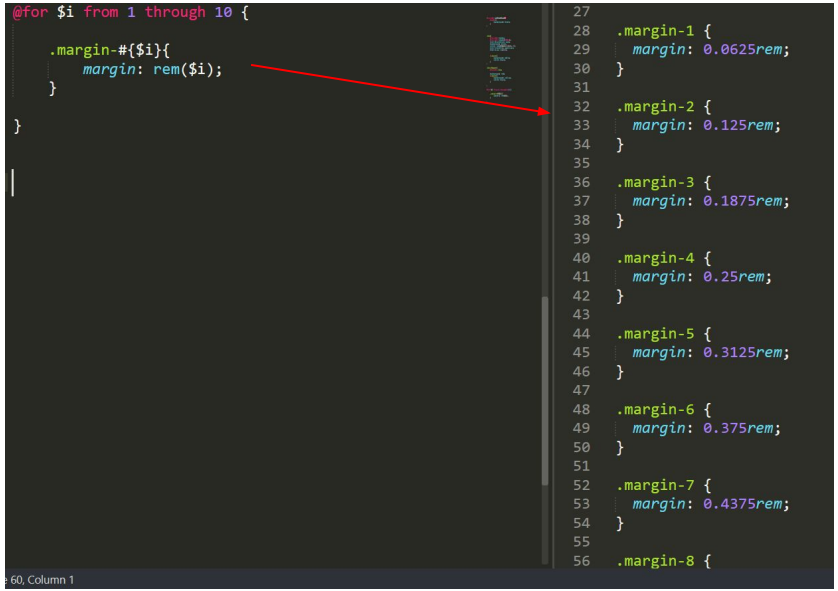
Les functions s’appelle plus simplement que les mixins….
De plus, les functions ne peuvent s’appeler que comme valeur d’une propriété alors que les mixins comme règles CSS

SASS - Les boucles



Voici la syntaxe de la boucle “@for”.
La variable $i peut être renommée ainsi que la valeur de dépar (1) et de fin (10). Entre les accolades, on met le code à dupliquer…10 fois pour le coup…

SASS - Les boucles



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.

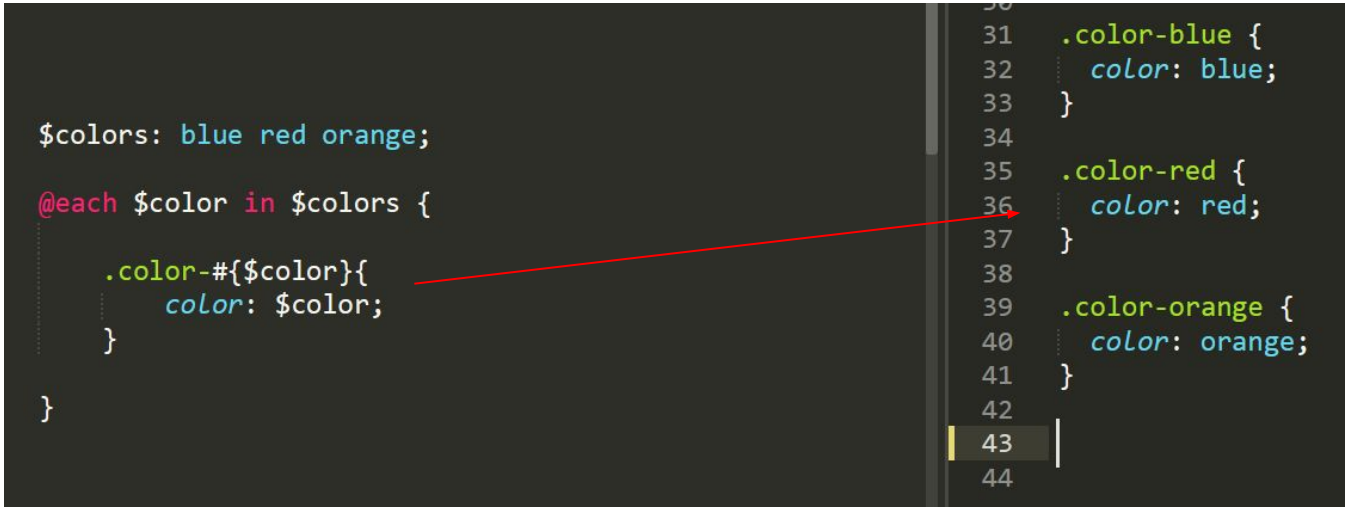
SASS - Les boucles

```
$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



```scss
$colors: blue red orange;

@each $color in $colors {

    .color-#{$color}{
        color: $color;
    }

}
```

```css
.color-blue {
  color: blue;
}

.color-red {
  color: red;
}

.color-orange {
  color: orange;
}
```

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.

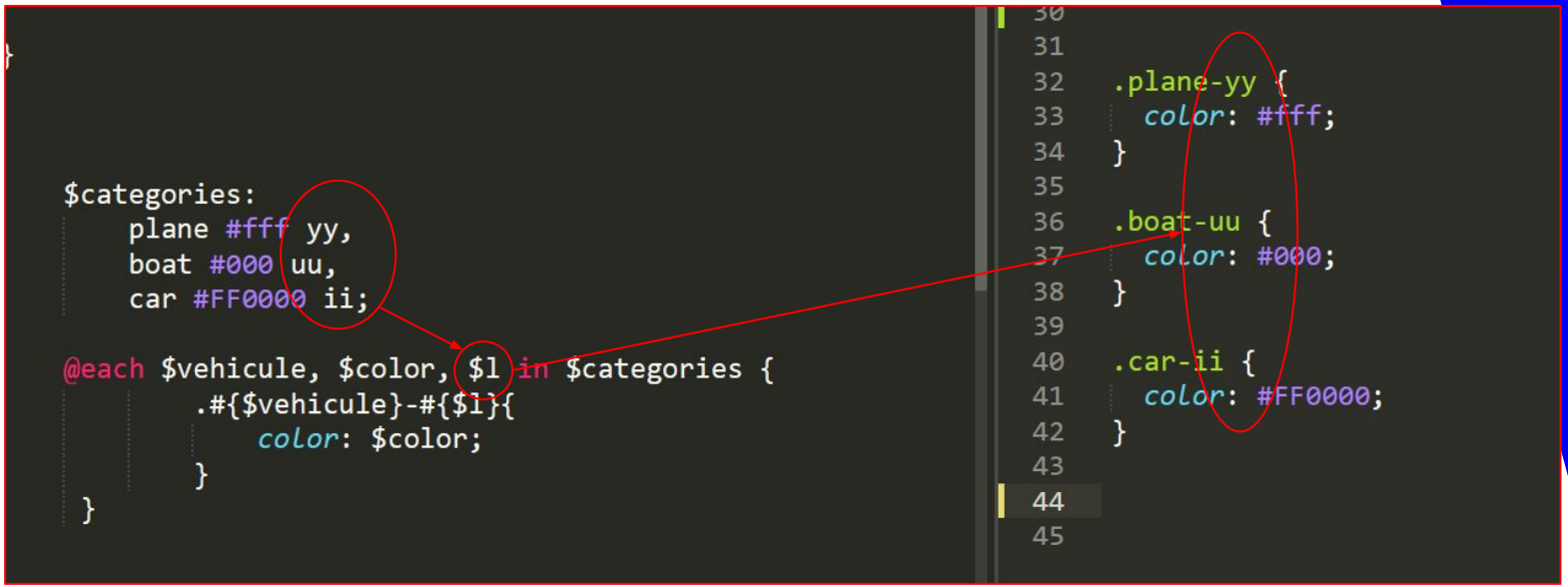
SASS - Les boucles

```scss
$categories:
    plane #fff,
    boat #000,
    car #FF0000;

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

```css
.plane {
  color: #fff;
}

.boat {
  color: #000;
}

.car {
  color: #FF0000;
}
```

(Re) TADA !

SASS - Les boucles



```
}


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

@each $vehicule, $color, $l in $categories {
        .#{$vehicule}-#{$l}{
            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
```

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

style.scss _responsive.scss

```scss
//-}

//}

    $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};
                }
            }


        }

    }
```

style.css _mixins.scss

```css
@media only screen and (max-width: 576px) {
  .sm-justify-row {
    justify-content: row;
  }
}
@media only screen and (max-width: 768px) {
  .md-justify-row {
    justify-content: row;
  }
}
@media only screen and (max-width: 992px) {
  .lg-justify-row {
    justify-content: row;
  }
}
@media only screen and (max-width: 1200px) {
  .xl-justify-row {
    justify-content: row;
  }
}
@media only screen and (max-width: 1400px) {
  .xxl-justify-row {
    justify-content: row;
  }
}
@media only screen and (max-width: 576px) {
  .sm-justify-column {
    justify-content: column;
  }
}
```

TADADADADA !

SASS - Les boucles

style.scss | _responsive.scss | style.css | _mixins.scss

```
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}-#{$elem}
                                {
54                               #{$property}: #{$elem};
55                            }
56                      }
57
58
59                  }
60
61              }
62          }
63
64
65
66
67
68
69
70
```

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