

# FORMATION SASS

**FRÉDÉRIC GIROD  
ELORRI**



**QU'EST-CE QUE SASS ?**

# SASS ?

- SASS est un préprocesseur CSS



# SASS ?

- SASS est un préprocesseur CSS
- Ce qui va nous permettre d'écrire de la CSS plus simplement et de façon plus organisée
- On appelle ça un préprocesseur CSS car il y a une phase de compilation pour transformer le SASS en CSS

# SASS ?

- Il n'existe pas que SASS dans le monde des préprocesseur CSS

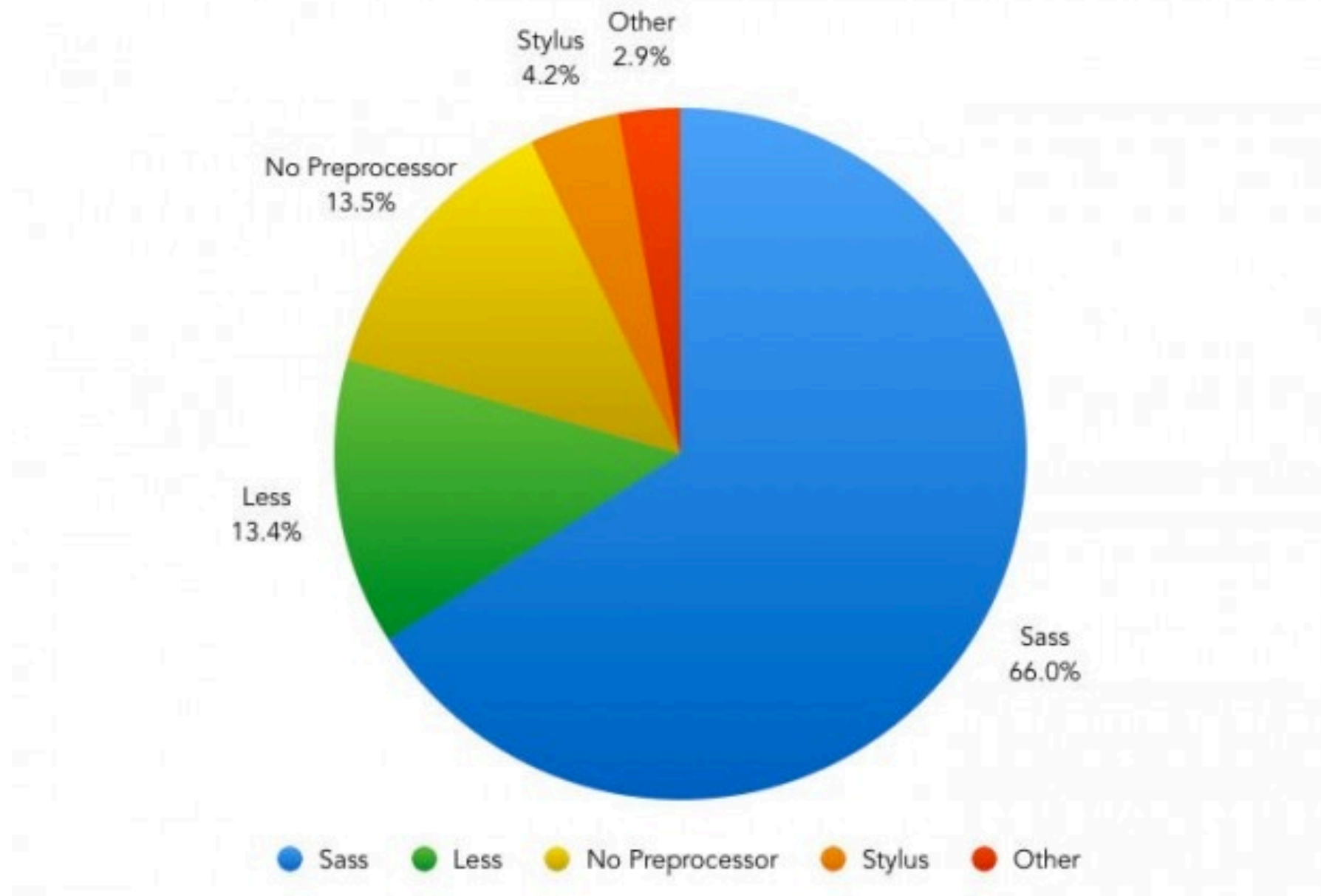
{less}

Sass

stylus

# SASS ?

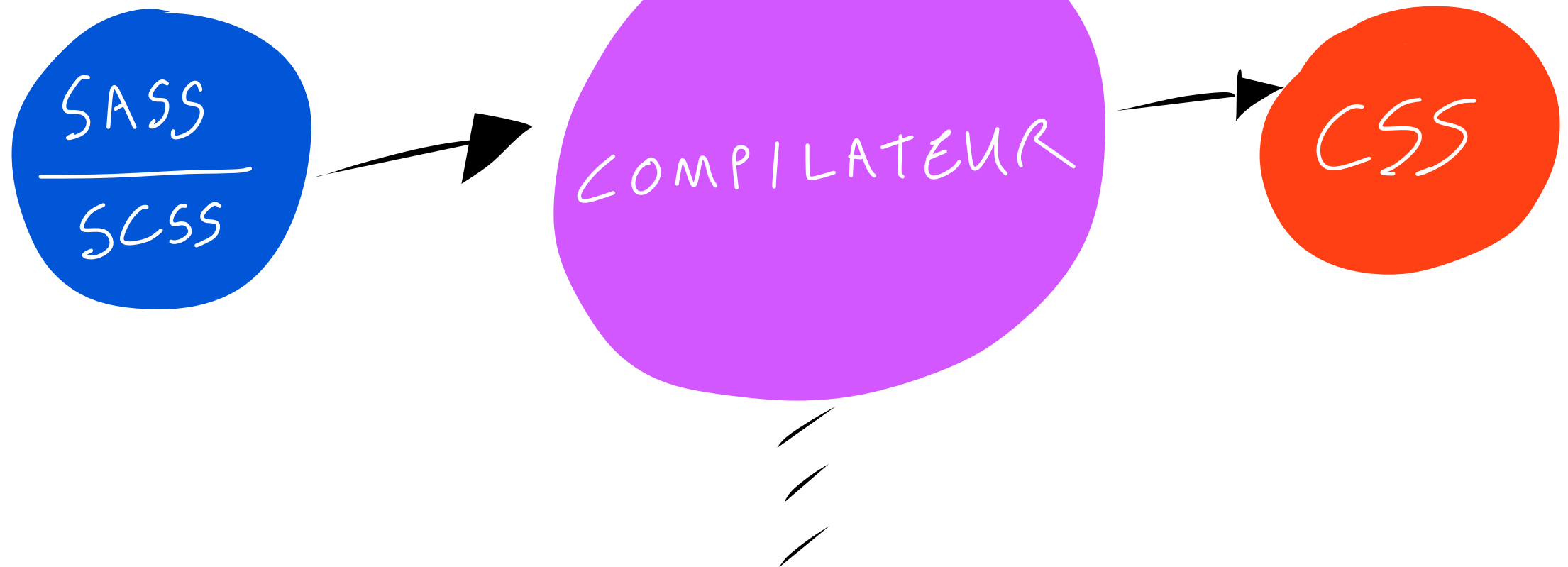
- Il n'existe pas que SASS dans le monde des préprocesseur CSS mais...



# SASS ?

DEVELOPPEMENT

NAVIGATEUR



# SASS ?

- SASS / SCSS quelles différences ?
  - ➔ SASS a deux syntaxes différentes
  - ➔ Une syntaxe historique avec l'extension `.sass`
  - ➔ L'autre plus récente avec l'extension `.scss`
  - ➔ La syntaxe SCSS (Sassy CSS) est extrêmement proche de celle de la CSS et est la plus utilisée
  - ➔ La syntaxe SCSS rend l'apprentissage de SASS plus simple



# SASS ?

- SASS / SCSS quelles différences ?

```
/* SCSS */  
h1 {  
  color: blue;  
}
```

```
/* SASS */  
h1  
  color: blue
```

# SASS ?

- SASS s'oriente autour de 4 notions clés :
  - ➡ Les variables
  - ➡ La syntaxe imbriquée
  - ➡ Les mixins
  - ➡ Les boucles et conditions

# SASS ?

- Toutes ces notions sont la raison pour laquelle le développement d'une feuille de style sans préprocesseur CSS se fait de plus en plus rare
- Ces fonctionnalités seront peut-être un jour nativement disponible en CSS... (c'est déjà le cas avec les variables malgré des soucis de compatibilité)
- Mais aujourd'hui ce n'est globalement pas le cas et il serait dommage de se passer de ces fonctionnalités !

# SASS ?

**EST CE QUE C'EST PLUS CLAIRE ?**



# LES VARIABLES

# LES VARIABLES

- Comme dans un langage de programmation classique nous allons disposer de la notion de variable
- En gros un emplacement/une « boîte » pour stocker une valeur/une information que l'on pourra réutiliser à plusieurs reprises

# LES VARIABLES

- Les variables en SASS vont être très pratique pour stocker des couleurs par exemple
- Ainsi je stocke une fois la valeur dans une variable qui portera un nom/identifiant unique
- Ensuite j'utilise uniquement le nom/identifiant que j'ai donné à cette variable
- Le gros avantage est : en cas de changement de la couleur au cours du projet, je modifie la valeur affectée à ma variable et le changement va se répliquer automatiquement

# LES VARIABLES

- Quand ? Dès qu'une valeur sera utilisée plus d'une fois
- Comment ?
  - ➔ En deux étapes
  - ➔ Déclaration d'une variable en utilisant la syntaxe :  
`$nom-de-la-variable: valeur;`
  - ➔ Utilisation de la variable en réutilisant le nom/identifiant donné à la variable, par exemple :  
`color: $nom-de-la-variable;`



# LES VARIABLES

- Mise en pratique
- Initialisation du projet de test
  - ➡ Créer un projet `startsass`
  - ➡ Ouvrir le dossier dans Visual Studio Code (ou autre IDE)
  - ➡ Création d'un fichier `index.html`
  - ➡ Création d'un sous-dossier `sass` avec un fichier `style.scss` à l'intérieur de ce dossier

# LES VARIABLES

- Mise en pratique
  - ➡ Créer deux variables qui vont contenir les deux couleurs principales de votre projet de test
  - ➡ Utiliser l'une des couleurs sur le titre de niveau 1 et appliquer l'autre couleur en fond coloré de ce même titre

# LES VARIABLES

**C'EST SUPER... MAIS COMMENT ON COMPILE ??**

# COMPILATION

# COMPILATION

- Deux options s'offrent à vous :
  - ➡ Ligne de commande
  - ➡ Logiciels

# COMPILATION

## EN LIGNE DE COMMANDE

- Installation de SASS
- Si vous utilisez ou connaissez npm (Node Package Manager)
  - `npm install -g sass`
- Sinon sous macOS
  - `brew install sass/sass/sass`

# COMPILATION

## EN LIGNE DE COMMANDE

- Une fois installée, nous pouvons compiler notre code SASS en CSS 🙌
- Une commande simple et unique
  - `sass source.scss destination.css`
  - bien entendu à adapter avec votre arborescence : `sass sass/source.scss css/destination.css`

# COMPILATION

## AVEC DES LOGICIELS

- De nombreux logiciels

➡ Prepros 🔥

➡ Codekit

➡ Koala

➡ Hammer

➡ ...



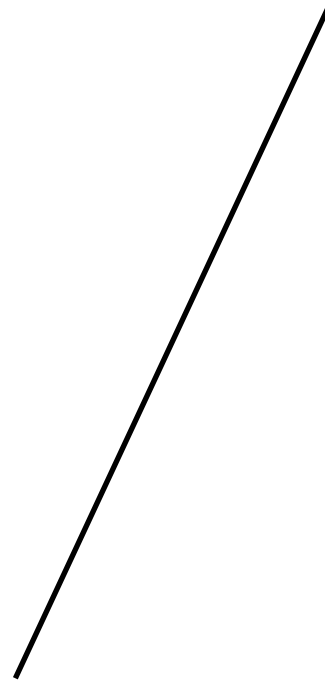
# COMPILATION

## AVEC DES LOGICIELS

- Pour commencer, Koala est le plus simple d'utilisation
- Interface minimaliste
- Il se limite à la compilation d'un fichier SASS en CSS
- Les autres logiciels sont plus complets comme Prepros que nous aurons l'occasion de découvrir un peu plus tard...

# COMPILATION

- Maintenant faites votre choix et compilez votre fichier style.scss



# LA SYNTAXE IMBRIQUÉE

# LA SYNTAXE IMBRIQUÉE

- Comme son nom l'indique la syntaxe imbriquée va nous permettre de faire de l'imbrication
- Imbriquer les sélecteurs les uns dans les autres comme pour une structure HTML avec la notion de parent, enfant et voisin...

# LA SYNTAXE IMBRIQUÉE

- Mais ça ne s'arrête pas là nous allons aussi pouvoir imbriquer :
  - ➔ les pseudo-classes (`:hover`, `:focus`, ...)
  - ➔ les media queries (`@media`)

# LES MIXINS

# LES MIXINS

- Sous ce nom assez mystérieux se cache la notion de fonction pour SASS comme pour un langage de programmation classique
- A la différence d'une fonction « classique » vous pouvez renvoyer plusieurs lignes de CSS avec la mixin
- 2 étapes pour les mixins :
  - ➡ Déclaration d'une mixin
  - ➡ Utilisation d'une mixin

# DÉCLARATION D'UNE MIXIN

- Pour déclarer une mixin on commence par l'instruction `@mixin` suivie du nom que vous voulez donner à la mixin

- Exemples :

```
@mixin text-center {  
    text-align: center;  
}
```

```
@mixin text-blue {  
    color: $blue;  
}
```



# DÉCLARATION D'UNE MIXIN

- Comme une fonction, en JavaScript par exemple, une mixin peut avoir un ou plusieurs paramètres pour véhiculer des informations à la mixin
- Exemples :

```
@mixin text-align($align) {  
    text-align: $align;  
}
```

```
@mixin button($color, $background) {  
    color: $color;  
    background-color: $background;  
}
```

# DÉCLARATION D'UNE MIXIN

- Il est aussi possible de définir une valeur par défaut à nos paramètres de la mixin !
- Exemple :

```
@mixin text-align($align: left) {  
    text-align: $align;  
}
```

# UTILISATION D'UNE MIXIN

- Pour utiliser une mixin il suffit d'utiliser l'instruction `@include` suivie du nom de la mixin
- Exemple :


```
h1 {  
    // Application de la mixin text-center  
    @include text-center;  
}
```

# LES MIXINS

- Mise en pratique
  - ➡ Créer une mixin qui permet de mettre à 0 les marges extérieures et intérieures
  - ➡ Appliquer cette mixin sur le titre de niveau 1

# LES MIXINS

- Mise en pratique



CSS button

- ➔ Créer une mixin qui permet de mettre en forme un lien sous forme de bouton (exemple de mise en forme ci-dessus)
- ➔ Appliquer cette mixin sur la classe btn
- ➔ Rendre la mixin suffisamment modulable pour que l'on puisse l'utiliser sur d'autres classes comme par exemple btn-warning, btn-danger, ...

# LES CONDITIONS

# LES CONDITIONS

- Vous connaissez les conditions en programmation ?
- Pour rappel en Algo ça donne :

```
Si nb > 0 alors
    Ecrire("Nombre positif")
FinSi
```

```
Si nb > 0 alors
    Ecrire("Nombre positif")
Sinon Si nb = 0 alors
    Ecrire("Nombre nul")
Sinon
    Ecrire("Nombre négatif")
FinSi
```

# LES CONDITIONS

- En SASS c'est tout pareil, il n'y a que la syntaxe qui change...
- Pour écrire une condition en SASS nous utilisons l'instruction `@if` suivie de la condition à tester

- Exemples :

```
@if $color == blue {  
  @include btn-blue;  
}
```

```
@if $color == blue {  
  @include btn-blue;  
}  
@else if $color == red {  
  @include btn-red;  
}  
@else {  
  @include btn;  
}
```



# LES CONDITIONS

- Mise en pratique
  - ➡ Modifier la mixin button pour faire en sorte que nous ayons la possibilité d'avoir ou non un bouton avec des bords arrondis
  - ➡ Appliquer cette modification sur un des appels à cette mixin

# LES BOUCLES

# LES BOUCLES

- Les boucles permettent de répéter une instruction plusieurs fois par rapport à :
  - ➡ un compteur (la boucle for)
  - ➡ une condition (la boucle while)
  - ➡ une liste d'éléments (la boucle each)

# LA BOUCLE FOR

- Si nous faisons le parallèle avec l'Algo, cette instruction se résume à la boucle Pour :
  - Pour *i* allant de 1 à 10 faire ...
- La boucle for se fait par l'intermédiaire de l'instruction `@for` suivie d'une variable qui va contenir la valeur du compteur elle même suivie de la valeur de début et de fin de la boucle
  - `@for $i from <début> through <fin>`

# LA BOUCLE FOR

- Exemple :

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

- La syntaxe `#{ $i }` permet d'utiliser la valeur pour l'afficher dans un sélecteur (dans un chaîne de manière générale), vous remarquerez que nous n'avons pas besoin de cette syntaxe pour faire des calculs ou appliquer la valeur sur une propriété par exemple

# LA BOUCLE WHILE

- Si nous faisons le parallèle avec l'Algo, cette instruction se résume à la boucle TantQue :
  - TantQue  $i \leq 10$  faire ...
- La boucle while se fait par l'intermédiaire de l'instruction @while suivie d'une condition qui va conditionner le nombre d'itérations dans la boucle
  - @while <condition>

# LA BOUCLE WHILE

- Exemple :

```
@while $i <= 3 {  
    .margin-#{ $i } {  
        margin: 1rem * $i;  
    }  
  
    $i: $i + 1;  
}
```
- Attention à bien incrémenter la valeur de la variable `$i` sinon c'est la catastrophe 😬

# LA BOUCLE EACH

- L'intérêt de la boucle each est de pouvoir parcourir n'importe quelle liste et même des collections
  - Se fait par l'intermédiaire de l'instruction `@each` suivie d'une variable qui va contenir la valeur d'un élément de la liste/collection différent à chaque itération de la boucle
- `@each $item in 1, 2, 3 {...}`



# LA BOUCLE EACH

- A propos des listes, plusieurs notations possibles mais la même finalité :

→ 1, 2, 3

→ 1 2 3

→ (1, 2, 3)

→ (1 2 3)

# LA BOUCLE EACH

- Exemple :

```
@each $color in red, blue, green {  
  .bg-#{$color} {  
    background-color: $color;  
  }  
}
```

# LA BOUCLE EACH

- Peut aussi s'écrire :

```
$colors: red, blue, green;
```

```
@each $color in $colors {  
  .bg-#{ $color } {  
    background-color: $color;  
  }  
}
```

# LA BOUCLE EACH

- Et donc avec des collections ça donne quoi ? 🤔
  - ➡ Il suffit de reprendre la philosophie d'un tableau associatif en Algo ou en JavaScript par exemple
  - ➡ Une collection fonctionne avec des associations de clés et de valeurs le tout stocké dans une variable

# LA BOUCLE EACH

- Exemple de déclaration d'une collection :

```
$grid: (  
    columns: 12  
);
```

- Pour récupérer une valeur d'une clé d'une collection vous disposez de la fonction map-get prenant en paramètre la collection et la clé de la valeur souhaitée :

```
$columns: map-get($grid, "columns");
```

# LA BOUCLE EACH

- Ainsi avec une boucle each on peut sans problème parcourir une collection :

```
@each $key, $value in $grid {...}
```

- Attention de bien mettre `$key`, `$value` car si vous ne mettez qu'une seule variable vous obtiendrez la concaténation de la clé et de la valeur...

# LA BOUCLE EACH

- Exemple :

```
$text-transform: (  
  text-uppercase: uppercase,  
  text-capitalize: capitalize,  
  text-lowercase: lowercase  
);
```

```
@each $classname, $style in $text-transform {  
  .#{$classname} {  
    text-transform: $style;  
  }  
}
```

# LA BOUCLE EACH

- Mise en pratique
  - ➔ Par l'intermédiaire d'une boucle each générer trois nouveaux styles de bouton portant les classes :
    - btn-success
    - btn-warning
    - btn-danger



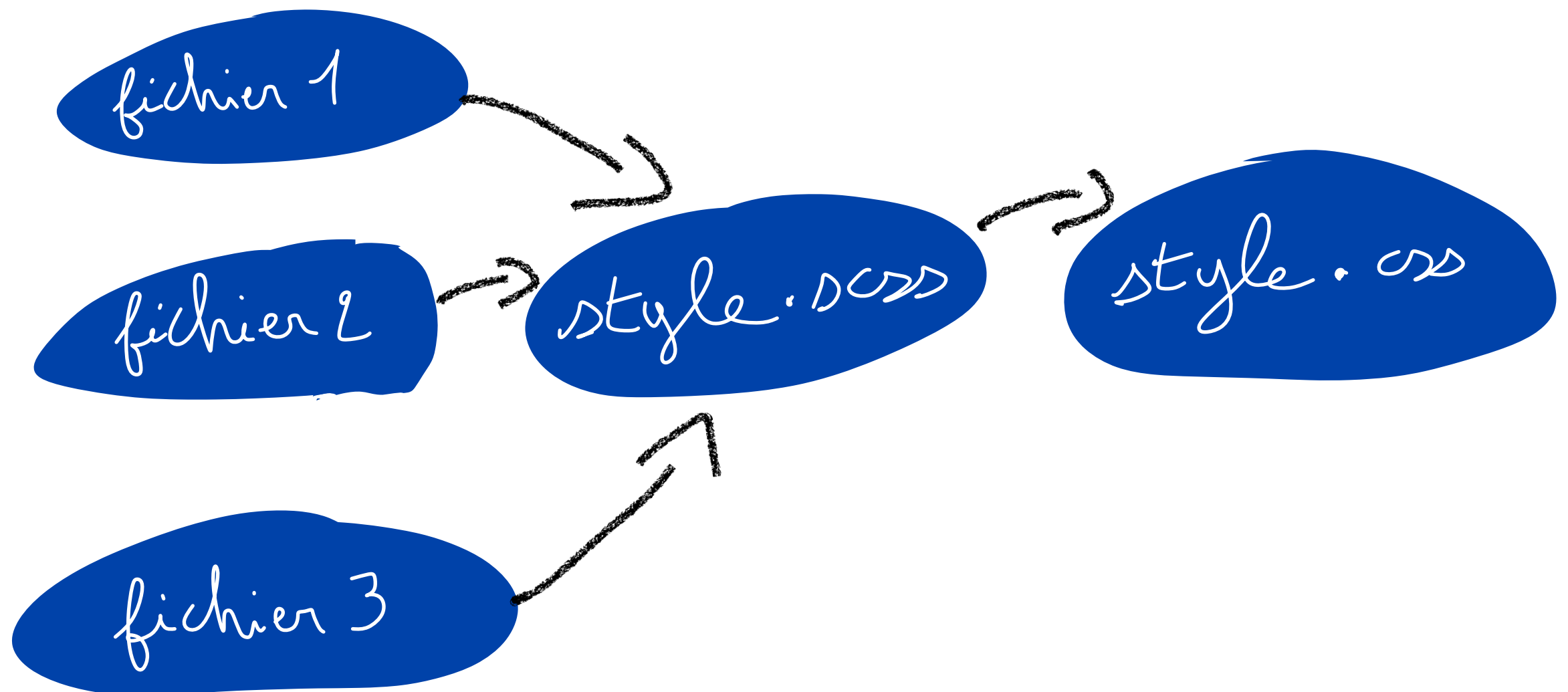
# LES FICHIERS PARTIELS

# LES FICHIERS PARTIELS

- Sous ce nom se cache l'idée de diviser en plusieurs fichiers notre feuille de style
- Dans l'optique notamment de pouvoir réutiliser certains fichiers d'un projet à un autre
- En CSS, la division de feuilles de style est déconseillée car elle augmente le nombre de requêtes au serveur...
- Mais avec SASS aucun problème car tous ces fichiers seront compilés en un seul fichier CSS !

# LES FICHIERS PARTIELS

- Avec SASS ce n'est donc pas un problème et je vous encourage à le faire



# CRÉATION D'UN FICHIER PARTIEL

- Le nom du fichier doit être précédé d'un underscore \_ pour que SASS comprenne qu'il s'agit d'un fichier partiel
- Par exemple si je veux créer un fichier partiel pour mettre toutes mes variables SASS à l'intérieur, je vais créer un fichier nommé `_variables.scss`

# IMPORT D'UN FICHIER PARTIEL

- Quand nous importons un fichier nous n'avons pas besoin d'ajouter :
  - ➔ le tiret \_ précédant le nom du fichier
  - ➔ l'extension .scss.
- SASS comprend parfaitement qu'il doit aller chercher un fichier partiel même sans ces informations
- La syntaxe pour importer un fichier partiel est :  
`@import "monfichierpartiel";`

# LES FICHIERS PARTIELS

- Mise en pratique
  - ➡ Créer des fichiers partiels pour :
    - les variables
    - les mixins
  - ➡ Importer ces fichiers partiels dans le fichier principal `style.scss`

**ALLER PLUS LOIN**

# AVOCODE

**POUR EXPORTER DU CODE SASS DIRECTEMENT  
DEPUIS UNE MAQUETTE GRAPHIQUE !**





# FRAMEWORK SASS

- Il existe des frameworks pour SASS les plus connus étant :
  - ➔ Compass (<http://compass-style.org/>)
  - ➔ Bourbon (<https://www.bourbon.io/>)
- C'est quoi l'idée ?
  - ➔ Dans le cas de Compass et Bourbon, avoir une bibliothèque de mixins prête à l'emploi...

# FRAMEWORK SASS

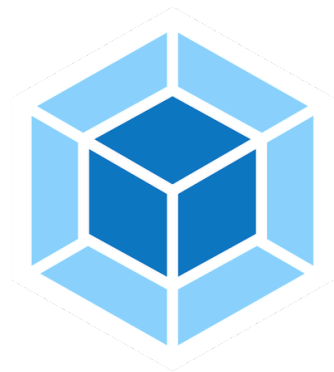
- Bourbon a également fait des petits avec :
  - ➔ Neat (<https://neat.bourbon.io/>)
    - Un système de grille en SASS
  - ➔ Bitters (<https://bitters.bourbon.io/>)
    - Structuration d'un projet en SASS couplé à Bourbon
  - ➔ Refills (<https://refills.bourbon.io/>)
    - Composants prêt à l'emploi

# AUTOMATISATION DES TACHES

- On appelle ça plus communément l'utilisation d'un task runner
- Un peu comme pour la simple compilation d'un fichier SASS en CSS vous allez avoir 2 solutions :
  - ➡ Utilisation d'un logiciel
  - ➡ Tout faire à la « mano »

# AUTOMATISATION DES TACHES

- Si vous voulez tout faire à la « mano » il va falloir se tourner vers des solutions comme :



**webpack**



**GRUNT**



# AUTOMATISATION DES TACHES

- Mais à quoi ça sert au juste ?



# AUTOMATISATION DES TACHES

- Mais à quoi ça sert au juste ?
  - ➡ Compilation SASS
  - ➡ Minification de fichiers CSS, JS, ...
  - ➡ Optimisation des images
  - ➡ Live server
  - ➡ Live reload
  - ➡ ...

# AUTOMATISATION DES TACHES

- Tout ça se fait via des fichiers de configuration écrit en JavaScript, exemples :
  - ➡ webpack.js
  - ➡ gulpfile.js

# AUTOMATISATION DES TACHES

- Mais lequel choisir ?
  - ➡ Si vous faites du React (ou Angular) orientez vous vers Webpack car il est très orienté application
  - ➡ Autrement Gulp fait très bien le job
  - ➡ Grunt est un peu plus verbeux et moins accessible à mon sens...



# AUTOMATISATION DES TACHES

- Exemple avec Webpack

➡ dans un fichier webpack.config.js

```
➡ module.exports = {  
  module: {  
    rules: [  
      {  
        test: /\.scss$/,  
        use: [  
          { loader: "css-loader" },  
          { loader: "sass-loader" }  
        ]  
      },  
    ]  
  }  
};
```

# AUTOMATISATION DES TACHES

- Exemple avec Gulp

➡ dans un fichier gulpfile.js

```
➡ const gulp = require('gulp');  
   const sass = require('gulp-sass');  
  
   gulp.task('styles', () => {  
       return gulp.src('sass/**/*.scss')  
           .pipe(sass().on('error', sass.logError))  
           .pipe(gulp.dest('./css/'));  
   });
```

➡ Ne reste plus qu'à lancer la commande `gulp styles`

# AUTOMATISATION DES TACHES

- Et avec un logiciel alors ?
- Souvenez-vous nous avons parlé de CodeKit, Prepros & co au moment d'évoquer Koala
- Découvrons Prepros qui va nous permettre de faire à peu près tout ce que permet les task runners JS vu précédemment




**[HTTP://PREPROS.IO](http://prepros.io)**

# AUTOMATISATION DES TACHES

**GLOBALEMENT QUELQUE SOIT LA SOLUTION PRIVILÉGIÉE,  
L'ESSENTIEL EST DE NE PLUS PERDRE DE TEMPS AVEC CES TÂCHES  
RÉCURRENTES ET CONSOMMATRICE DE TEMPS !**

**PRATIQUE**

# BOUTONS RÉSEAUX SOCIAUX

- Objectif :
  - ➔ Nous désirons créer une fonctionnalité SASS, réutilisable d'un projet à un autre qui va nous permettre de mettre en forme des boutons pour les réseaux sociaux
  - ➔ D'un site à un autre les réseaux sociaux peuvent être différent
  - ➔ Deux apparences possibles, exemple : 
  - ➔ Nous choisirons la meilleure solution à la fin !

# MA GRILLE BOOTSTRAP



- Objectif :
  - ➡ Nous désirons créer notre propre système de grille en interne
  - ➡ Réutiliser au maximum les notions vues
  - ➡ Si nécessaire bien entendu, sous entendu ne mettez pas une condition ou autre pour faire beau...
  - ➡ Chacun est libre de choisir ses noms pour ses classes, sa façon de procéder, etc... nous choisirons la meilleure solution à la fin !