front-end developer_
/webdesigner

Sass

Syntactically Awesome Style Sheets
Préprocesseur CSS

Bruxelles Formation | CEPEGRA | Olivier Céréssia | 2017

webdesigner.cepegra-labs.be

UN PRÉPROCESSEUR CSS, C'EST QUOI ?

L'écriture CSS est probablement la plus statique qui existe. Il n'est pas possible d'y ajouter des boucles, d'envisager des règles d'héritage, de stockage de données dans des variables, de calculs entre les différentes valeurs …

Bref, c'est facile à comprendre mais pas au niveau productivité, on peut faire vraiment beaucoup mieux.

C'est justement là que les préprocesseurs CSS sont intéressants.

Il en existe plusieurs sur le marché : LESS, SASS, Stylus, …

Mais c'est sur SASS que nous nous arrêterons car il est considéré comme le plus complet sur le marché et surtout c'est le plus utilisé à l'heure actuelle.

Les préprocesseurs offrent de nouvelles possibilités en termes de conception CSS, ils accélèrent le développement côté client et facilitent les modifications ultérieures.

Il serait vraiment dommage de se passer d'un service qui n'est là que pour augmenter le confort des développeurs front-end.

Un petit inconvénient néanmoins : il faut apprendre une façon un peu différente d'écrire dans sa feuille de styles.

C'est cette nouvelle syntaxe que nous allons aborder. Nous verrons ensemble les fonctionnalités les plus utiles de SASS.

TABLE DES MATIÈRES

SASS – LA FEUILLE DE STYLE « PROGRAMMABLE »

• Présentation

Sass est un préprocesseur libre et gratuit (ça commence bien ☺).

Sass est l'acronyme de **S**yntactically **A**wesome **S**tyle**S**heet.

Il est le préprocesseur qui a le plus de succès auprès des développeurs front-end probablement pour une raison simple : son installation est super-simple et rapide.

Sass permet d'étendre les possibilités offertes par le langage CSS en donnant la possibilité d'utiliser des variables, des mixins, des fonctions et d'autres techniques qui vont faire en sorte que votre feuille de styles CSS soit plus facile à créer, à maintenir et à étendre.

Sass est un méta-langage utilisé pour générer des feuilles de styles au format CSS, ce qui veut dire que vous continuerez à avoir dans votre application du CSS. La seule différence est que les fichiers d'origine seront, dans la plupart des cas, moins lourds et plus intelligibles.

• Fonctionnalités

Sass va vous permettre toute une série de choses qui, auparavant était assez fastidieux et va donc fortement accroître votre productivité.

Avec Sass vous allez pouvoir :

- Inclure des variables dans vos feuilles de styles
- Imbriquer vos règles CSS les unes dans les autres (on appelle ça le « nesting » en anglais)
- Utiliser les mixins (nous y reviendrons juste après)
- Utiliser des opérations mathématiques et des fonctions dans vos feuilles de styles

• Pourquoi utiliser un préprocesseur Css ?

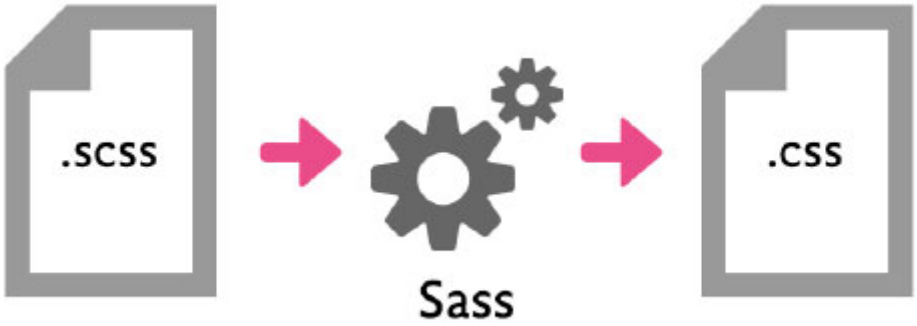
C'est simple. Dans nos feuilles de style, à l'heure actuelle, nous utilisons souvent les mêmes combinaisons de propriétés ensemble, nous répétons souvent des portions de code, nous appelons X fois la même couleur sur différents éléments.

Ce côté répétitif devrait ne pas exister. Nous ne devrions avoir qu'à faire une modification pour qu'elle soit valable sur tous les éléments sur lesquels cette règle est appliquée. Et c'est là que Sass va nous faciliter la vie (entre autre).

Sass : qu'est-ce que c'est ?

C'est un préprocesseur CSS, mais ça je vous l'avais déjà dit. Un préprocesseur, en gros, c'est une sorte d'étape intermédiaire entre vous et votre fichier .css final.

Les feuilles de styles classiques ne permettent pas l'utilisation de variables, de mixins (blocs réutilisables de styles), etc. alors que Sass le permet, mais il permet bien plus encore …



En gros, à partir de maintenant, nous allons écrire nos feuilles de styles non plus avec une extension .css mais bien avec une extension .scss qui va permettre à Sass de convertir toutes les nouvelles fonctionnalités que nous allons ajouter à nos anciens fichiers .css.

Ensuite, Sass sert de moulinette pour faire la traduction de toutes ces nouvelles choses et vous offre un fichier .css tout à fait classique mais compilé avec toutes vos règles bien appliquées.

Magique !

Bon, par contre, la conversion de l'un vers l'autre ne se fait pas de manière magique. Nous allons devoir faire des choses nouvelles qui vont peut-être vous faire un peu peur !

Installation de Sass

Installation en ligne de commande

Le guide présent sur le site officiel de Sass est parfait pour installer Sass pour la première fois.

Bonne nouvelle pour les utilisateurs Mac : Sass est codé en Ruby (et Ruby est installé par défaut sur Mac – pas sur Windows). Sur Windows, vous aurez donc une étape supplémentaire : vous allez devoir installer Ruby.

Pour les utilisateurs Windows donc, rendez-vous sur https://rubyinstaller.org/ et téléchargez la dernière version stable de Ruby. Laissez tous les choix par défaut et le tour est joué. Il n'est pas nécessaire d'installer le programmer supplémentaire proposé à la fin de l'installation.

Pour Mac et Windows, une fois que vous avez installé Ruby, il vous faut maintenant installer Sass sur votre machine et la bonne nouvelle c'est que ce n'est pas compliqué :

```
gem install sass

ou sur Mac : sudo gem install sass
```

Une fois l'installation terminée, vérifiez que vous avez bien installé Sass :

```
Sass –v
```

Cette commande vérifie quelle version de Sass est installée sur votre machine.

Et voilà c'est tout ! Vous allez pouvoir utiliser sass dans vos projets dès à présent.

Une alternative pour les personnes qui ont peur de la ligne de commande

Honnêtement ce serait une mauvaise idée de passer par cette solution : elle n'est pas très professionnelle et, même si elle semble plus conviviale, elle ne reflète pas la réalité du terrain.

Néanmoins, il est bon de savoir que des solutions embarquent tout dans un pack avec une interface pour tout faire à votre place. Voici les deux solutions (gratuites) présentées sur le site officiel de Sass :

- Koala (http://koala-app.com/)
- Scout-App (http://scout-app.io/)

• Mon premier fichier .scss

C'est un moment émotionnellement chargé que celui de créer votre premier fichier au format Sass. Pour cela, vous vous placez dans le dossier du projet sur lequel vous travaillez et vous allez lancer la commande suivante :

```
sass chemin/style.scss chemin/style.css
```

Ce que cette commande fait est simple : elle prend, en entier, le fichier style.scss et le convertit en fichier style.css.

Un problème avec cette commande : il faudrait la répéter à chaque fois que vous faites un changement et là, franchement, on perd tout l'intérêt du gain en productivité que Sass est sensé nous apporter.

Donc une autre commande est plus utile :

```
sass --watch chemin-entrant/style.scss:chemin-sortant/style.css
```

Avec cette ligne, une nouvelle fonctionnalité intéressante entre en jeu : le watch(ing). Watch qui, en anglais, veut dire « surveiller ». Quand vous lancez cette commande, Sass passe en mode surveillance sur la première partie de votre commande et, dès qu'un changement est effectué sur le fichier en question, Sass le transforme et le « sauvegarde » dans le chemin de destination, magique !

```
>>> Sass is watching for changes. Press Ctrl-C to stop.
```

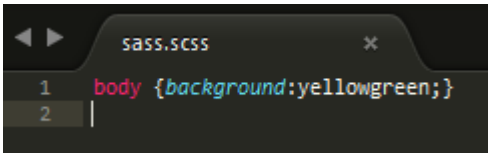
Vous devriez voir quelque chose comme ça, ce qui précise que la « surveillance » est en marche ! Youpie.

Et quand vous faites un changement sur votre fichier « source » voici ce qu'il se passe :

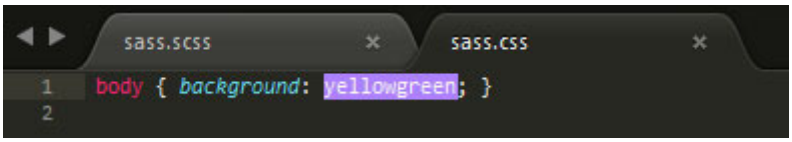
```
>>> Change detected to: scss/styles.scss
      write css/style.css
      write css/style.css.map
```

Génial ! Il a repéré qu'il y a eu un changement dans le fichier styles.scss et du coup, il me crée, à la volée, sans que je lui demande rien du tout, mon fichier .css ! C'est facile non ? ☺

Donc si vous écrivez ceci dans votre fichier .scss :



Il va le traduire dans un fichier .css qui portera le même nom mais qui aura donc bien l'extension .css (et non plus .scss) :



Super ! Notre « traducteur » de fichiers Sass fonctionne à merveille, maintenant nous allons enfin pouvoir découvrir les différentes fonctionnalités qui vont bouleverser votre quotidien !

• Créer son environnement de travail modularisé

Créons ensemble une structure de dossiers et de fichiers que l'on va pouvoir réutiliser pour tous nos projets.

Cette structure pourrait comprendre, dans le dossier au nom de notre projet, un dossier src et un dossier dist (source et distribution).

Dans src nous allons mettre tous nos fichiers de travail écrits en SASS, nos fichier js non-minifiés, nos images non compressées, etc.

Alors que dist sera votre dossier de production. C'est-à-dire qu'à peu de choses près, vous pourrez prendre et déposer ce dossier sur votre serveur distant et tout devrait fonctionner. C'est votre site fini.

Vous pouvez déjà créer des fichiers .html vierges et inclure les fichiers habituels que vous liez à vos projets, si vous en avez (travailler en CDN est une meilleure solution).

Pour les fichiers Sass, une pratique courante est de tout « découper » en plein de petits fichiers.

Mais pourquoi cela ?

C'est simple !

D'abord pour pouvoir facilement s'y retrouver et savoir exactement quel fichier vous allez devoir éditer pour effectuer une modification.

Ensuite, ce côté « modulaire » est aussi une excellente façon de pouvoir reprendre une feuille sass dans un projet et de l'utiliser dans un autre projet. Je pense principalement aux Mixins, à des feuilles Sass pour vos boutons, pour le reset, etc.

Au final, tout sera de toute façon réuni à l'intérieur d'un seul et même fichier.

Voici un exemple d'architecture de fichiers Sass que l'on pourrait utiliser (source : http://www.sitepoint.com/architecture-sass-project/), vous n'êtes évidemment pas obligés de vous y tenir, mais c'est une bonne base de réflexion pour votre propre architecture.

```
sass/
|
|- base/
|   |- _reset.scss       # Reset/normalize
|   |- _typography.scss  # Typography rules
|   ...                  # Etc…
|
|- components/
|   |- _buttons.scss     # Buttons
|   |- _carousel.scss    # Carousel
|   |- _cover.scss       # Cover
|   |- _dropdown.scss    # Dropdown
|   |- _navigation.scss  # Navigation
|   ...                  # Etc…
|
|- helpers/
|   |- _variables.scss   # Sass Variables
|   |- _functions.scss   # Sass Functions
|   |- _mixins.scss      # Sass Mixins
|   |- _helpers.scss     # Class & placeholders helpers
|   ...                  # Etc…
|
|- layout/
|   |- _grid.scss        # Grid system
|   |- _header.scss      # Header
|   |- _footer.scss      # Footer
|   |- _sidebar.scss     # Sidebar
|   |- _forms.scss       # Forms
|   ...                  # Etc…
|
|- pages/
|   |- _home.scss        # Home specific styles
|   |- _contact.scss     # Contact specific styles
|   ...                  # Etc…
|
```

```
|- themes/
|   |- _theme.scss        # Default theme
|   |- _admin.scss        # Admin theme
|   ...                   # Etc…
|
|- vendors/
|   |- _bootstrap.scss    # Bootstrap
|   |- _jquery-ui.scss    # jQuery UI
|   ...                   # Etc…
|
|
`- main.scss              # primary Sass file
```

Un brin d'explications sur cette architecture de fichiers.

Dans `base/`, vous trouverez des fichiers que vous modifierez rarement comme votre reset (ou normalize), vos appels à vos typo, etc.

Dans `components/`, vous trouverez un ensemble de petits modules indépendants du genre : buttons, carousel, navigation, dropdown, etc.

Dans `helpers/`, vous trouverez des règles qui ne sont là que pour vous aider, c'est-à-dire qu'on ne définit rien de la structure de notre site ici. On met en place des variables, des fonctions et des mixins qu'on va pouvoir utiliser durant toute la durée de notre projet.

Dans `layout/`, vous trouverez votre mise en page générale, alors que components concerne des petits modules, layout, lui, s'occupe des grandes dispositions d'éléments sur vos pages (grille de mise en page, header, footer, siderbar, formulaires, etc.)

Dans `pages/`, vous trouverez les styles spécifiques à certaines pages de votre site, on en a dans tous nos projets.

Si vous travaillez sur un très grand site, alors `themes/` pourrait être utile. Dans les autres cas vous ne l'utiliserez pas.

Dans `vendors/`, vous trouverez les feuilles de styles externes qui sont issues des outils que vous allez utiliser. Par exemple on peut retrouver les fichiers Sass pour Bootstrap, jquery-ui, fontawesome, etc.

Tout cela n'a qu'un but : gagner du temps. Dès lors il est logique de créer un dossier de travail « de base » pour tous vos projets Web.

• Les variables

C'est une des nouveautés majeures introduites par Sass : l'utilisation de variables à l'intérieur même de vos feuilles de styles.

Dans quels cas cela va être intéressant ? Pour l'utilisation de couleurs sur votre site par exemple, quand on doit respecter une charte qui comporte 3 ou 4 couleurs, il suffit de placer chacune d'elles dans une variable et de les appeler par leur nom. C'est plus facile à retenir qu'un code Hexadécimal et c'est super-facile à maintenir en une fois, vous pourrez changer toutes les couleurs de votre site, sans même passer par un rechercher/remplacer ! Yahoo !

Petit exemple pratique issu du site officiel de Sass :

```
$font-principale:    Helvetica, sans-serif;
$couleur-principale: #333;

body {
  font: 100% $font-principale;
  color: $couleur-principale;
}
```

Vous aurez remarqué la présence d'un dollar ($) qui précède directement le nom de la variable (que vous pouvez choisir librement). Ensuite les deux-points (:) qui introduisent la valeur de la variable.

Vous écrivez ces variables en ouverture de votre document .scss et elles seront dispo pour tout votre document.

On peut y stocker ce qu'on veut évidemment, des valeurs hexadécimales mais aussi un font-family par exemple ! Tout est possible.

Et pour l'appeler, il suffit alors de l'appeler de la même façon que vous l'avez déclarée, mais à l'endroit où vous voulez la voir écrite, c'est simple non ?!

Le résultat, dans votre fichier .css sera évidemment le suivant :

```
body {
  font: 100% Helvetica, sans-serif;
  color: #333;
}
```

On remarquera la disparition des premières lignes de déclaration des variables qui, lors de la compilation du fichier .css, disparaissent, et leurs valeurs sont redistribuées à chaque fois que vous les avez appelées dans votre code css ! C'est cool !

Encore plus fort, il est possible avec Sass, à partir d'une couleur donnée, de la rendre plus sombre ou plus claire et vous n'avez qu'à demander de combien de % vous désirez éclaircir ou assombrir votre couleur :

```
$couleur-principale: #ea4c89;

a {color: $couleur-principale;
   &:hover {
            color: darken($couleur-principale, 30%);
           }
}
```

Ce code générera le code css suivant :

```
a {color: #ea4c89;}
a:hover {color: #8d1040;}
```

La couleur a donc été « assombrie » de 30% par rapport à la couleur originale que vous aviez indiquée.

Et c'est possible de l'éclaircir également avec la valeur lighten à la place de darken !

• L'imbrication des sélecteurs css (nesting)

C'est une nouvelle façon d'écrire vos sélecteurs qui est finalement beaucoup plus logique.

Jusqu'à présent, vous écriviez ceci :

```
nav ul { margin: 0;}
nav li { display: inline-block; }
```

Maintenant, avec Sass, on écrit plutôt de cette façon-ci :

```
nav {
  ul { margin: 0; }
  li { display: inline-block; }
}
```

Mais le résultat restera le même que précédemment, on gagne juste un peu en écriture et c'est un peu plus logique d'imbriquer ul et li à l'intérieur de la nav puisqu'ils sont ses enfants. Rien de fou, mais ça va changer un peu votre façon d'écrire vos sélecteurs, c'est sûr.

En gros, cette façon d'écrire est plus logique, puisqu'elle s'inspire directement de la structure de notre document html, pour autant que vous ayez bien marqué votre indentation.

Encore mieux, vous pouvez aussi imbriquer des propriétés css qui commencent toutes par le même préfixe comme par exemple font-(size/family/weight/style/etc.) de cette façon :

```
header {
  font : {
          size: 54px;
          family: Georgia, serif;
          weight: bold;
          }
}
```

Pratique, non ? ☺ Et ça fonctionne aussi avec une propriété comme text-(transform / decoration / align).

Toujours plus fort, il est aussi possible de cibler le parent direct à l'intérieur d'une propriété css de cette façon :

```
a {
  color : red;
    &:hover {color : blue;}
}
```

Ce qui génère le code css suivant :

```
a { color : red;}
a:hover { color : blue;}
```

• Les mixins

Encore une nouvelle fonctionnalité super-intéressante qui va vous faire gagner de précieuses minutes de dev.

Prenons un exemple assez éloquent qui nous touche assez souvent :

```
nav ul { transition: .3s all ease-in-out;}
wrapper a { transition: .5s all ease-in-out;}
```

Il n'est pas rare d'avoir la même transition (à un détail près (la durée)) un peu partout sur notre site Web. Et normalement, pour que ces transitions fonctionnent partout, nous devons les faire précéder d'un préfixe propre à chaque parser, nous devrions donc écrire :

```
nav ul { -webkit-transition: .3s all ease-in-out;
         -moz-transition: .3s all ease-in-out;
         -ms-transition: .3s all ease-in-out;
         transition: .3s all ease-in-out;}
wrapper a { -webkit-transition: .3s all ease-in-out;
            -moz-transition: .3s all ease-in-out;
            -ms-transition: .3s all ease-in-out;
            transition: .3s all ease-in-out;}
```

C'est un peu fastidieux, non ?

Heureusement, Sass a pensé à nous et voici la solution miracle : les mixins !

Voyons voir comment ça fonctionne :

```
@mixin transition {
         -webkit-transition: .3s all ease-in-out;
         -moz-transition: .3s all ease-in-out;
```

```
        -ms-transition: .3s all ease-in-out;
        transition: .3s all ease-in-out;
}
nav ul {@include transition;}
wrapper a {@include transition; color : red}
```

On peut aussi placer des arguments dans un mixin pour faire varier certaines parties de nos propriétés :

```
@mixin transition($secondes) {
        -webkit-transition: $secondes all ease-in-out;
        -moz-transition: $secondes all ease-in-out;
        -ms-transition: $secondes all ease-in-out;
        transition: $secondes all ease-in-out;
}
nav ul {@include transition(.3s);}
wrapper a {@include transition(.5s); color : red}
```

Si vous désirez placer plusieurs arguments, voilà comment l'écrire :

```
@mixin transition($secondes, $effet) {
        -webkit-transition: $secondes all $effet;
        -moz-transition: $secondes all $effet;
        -ms-transition: $secondes all $effet;
        transition: $secondes all $effet;
}
nav ul {@include transition(.3s, ease);}
wrapper a {@include transition(.5s, ease-in-out);}
```

Et si, pour certains arguments, vous désirez avoir une valeur par défaut :

```
@mixin transition($secondes : .3s, $effet : ease-in-out) {
        -webkit-transition: $secondes all $effet;
        -moz-transition: $secondes all $effet;
        -ms-transition: $secondes all $effet;
```

```
        transition: $secondes all $effet;
}
nav ul {@include transition(.3s, ease);}
wrapper a {@include transition(.5s);}
```

Dans ce cas-là, si vous n'indiquez pas de valeur pour votre 2e argument, il prendra la valeur par défaut. Et si vous désirez ne changer que la valeur du 2e argument, alors il faut ré-indiquer le nom de la variable qui l'introduit : `nav ul {@include transition($effet : ease);}`

Les mixins sont particulièrement intéressants avec CSS3 simplement parce que nous sommes obligés de répéter les préfixes devant chaque valeur CSS3 non-validées par nos navigateurs préférés.

• L'imbrication dans les media queries

Une petite subtilité ici aussi. Lorsqu'on l'on fait usage des mixins pour nos media-queries, nous pouvons directement les mettre à l'intérieur de chaque règle que l'on va contredire.

L'avantage ? Vous n'avez pas à répéter le sélecteur puisque vous allez imbriquer la déclaration dans le sélecteur même.

L'inconvénient ? Vos media queries vont être éparpillées un peu partout dans votre feuille de styles.

Voici comment ça fonctionne :

```
Section.main {
            float : left;
            @media screen and (max-width :800px) {
            float : none;
            }
}
```

• Les opérations mathématiques

Fini de devoir sortir la calculatrice pour savoir quel pourcentage représente un bloc par rapport à un autre ! Sass s'en charge pour vous de cette façon :

```
.wrapper {max-width : 960px}
.left-col { float : left ; width : 600px / 960px * 100%}
.right-col { float : right ; width : 300px / 960px * 100%}
```

Et le résultat dans votre feuille de style après la conversion :

```
.wrapper {max-width : 960px}
.left-col { float : left ; width : 62.5%}
.right-col { float : right ; width : 31.25%}
```

Et vous pouvez utiliser tous les opérateurs mathématiques classiques : +, -, * ou / ainsi que %.

• @Extend

Sass permet également d'étendre l'utilisation de certaines règles au sein d'autres sélecteurs. Par exemple, vous créez une classe qui fait ceci :

```
.message {
  border: 1px solid #ccc;
  padding: 10px;
  color: #333;
}
```

Mais vous avez des messages de différentes sortes : des messages d'erreur et des message de confirmation. Les deux types de messages ont la même base (ci-dessus) mais diffèrent évidemment au niveau de la couleur (par exemple de leur bordure), voici comment gagner du temps avec Sass :

```
.success {
  @extend .message;
  border-color: green;
}
.error {
  @extend .message;
  border-color: red;
}
```

Attention, @extend dysfonctionne quand il est utilisé au sein d'un @media.

• @import

Une autre fonction intéressante de SASS est la possibilité de « scinder » son travail, pour plus facilement s'y retrouver lors de l'édition par après.

Communément on essaye vraiment de scinder les choses qui sont groupables.

On pourrait de cette façon créer des fichiers pour :

- Le reset
- Tous les appels des fonts
- Les mixins
- Les variables
- Le header de votre site
- Le footer de votre site
- Tout ce qui concerne vos formulaires
- Une colonne ou un encart qui est toujours pareil partout
- Et finalement votre fichier principal qui sera commun à toutes les pages de votre site

Ceci n'est qu'une façon de découper les choses, il y a des tas de façons de faire et c'est à vous de trouver celle qui vous convient le mieux et, si vous bossez en équipe, de vous adapter à ce que l'équipe aura mis en place.

En découpant les choses en plein de petits éléments, il sera probablement plus facile de réutiliser certaines parties pour vos futurs projets, pensez-y !

Des tas de sites parlent de leur façon de faire :

Sitepoint : http://www.sitepoint.com/architecture-sass-project/

TheSassWay : http://thesassway.com/beginner/how-to-structure-a-sass-project

WiseHeartDesign : http://wiseheartdesign.com/articles/2010/01/22/structuring-a-sass-project/

Vous retrouverez souvent le même genre de dossiers (modules, partials, vendor …), ce sont presque devenu des conventions mais pas tout à fait alors surveillez cette façon de procéder, ça pourrait être intéressant pour votre activité professionnelle future.

Au niveau syntaxique, c'est simple :

```
@import "reset.scss";
@import "variables.scss";
@import "mixins.scss";
```

Il est important de comprendre comment fonctionne cet « import ». Il s'agit en fait de scinder vos règles et vos propriétés Css en plein de fichiers ponctuels pour rendre l'ensemble plus facile à éditer pour vous ou vos collègues et de les ramener tous en un seul fichier lors de l'export final.

• Fonctions liées aux couleurs

Sass embarque avec lui toute une série de fonctions liées aux couleurs qui vont vous permettre de, facilement, modifier :

- La luminosité d'une couleur,
- Sa saturation,
- Sa teinte,
- Sa transparence,
- Etc.

Voyons quelques cas pratiques.

Imaginons une couleur que vous avez déjà *variabilisée* et nommée `$base-color` et vous aimeriez, pour cette couleur, obtenir, à partir de cette couleur, une couleur plus claire ou plus foncée (pour réaliser un effet de survol par exemple). Il vous suffit d'écrire :

```
darken( $base-color, 10% ) // Pour la foncer de 10%

lighten( $base-color, 10% ) // Pour l'éclaircir de 10%
```

De la même façon vous pouvez désaturer cette couleur :

```
saturate( $base-color, 20% )

desaturate( $base-color, 20% )
```

Plus d'informations ici : https://robots.thoughtbot.com/controlling-color-with-sass-color-functions

Conclusion

Il y a encore moyen d'aller plus loin avec Sass quand on l'associe à un framework css (Compass en l'occurrence) mais nous n'irons pas jusque-là.

Il est juste important que vous vous rendiez compte du gain en productivité que vous pourriez obtenir en vous familiarisant avec cette nouvelle façon d'écrire votre code css.

Dans le milieu de l'entreprise, Sass est majoritairement utilisé donc l'adopter, c'est ajouter une corde utile à votre arc.

Bonne chance avec Sass ! ☺