Webdesigner

Une formation de Bruxelles Formation CEPEGRA

Sass

Syntactically Awesome Style Sheets
Préprocesseur CSS

Bruxelles Formation | CEPEGRA | Olivier Céréssia | 2014

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 différents : LESS, SASS, Google Closure Stylesheets, Stylus, …

Mais c'est sur SASS que nous nous arrêterons car il est considéré comme le plus complet sur le marché.

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 ?

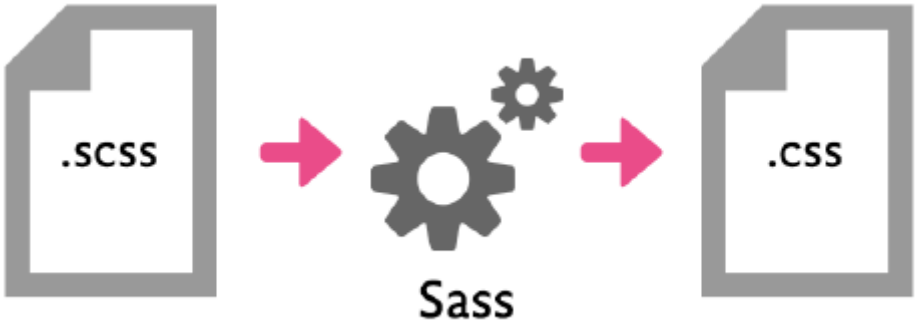
C'est simple. Dans nos feuilles de style, à l'heure actuelle, nous utilisons souvent les mêmes combinaisons de propriétés ensemble, nous répétons souvent des portions de code, nous appelons X fois la même couleur sur différents éléments.

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 !

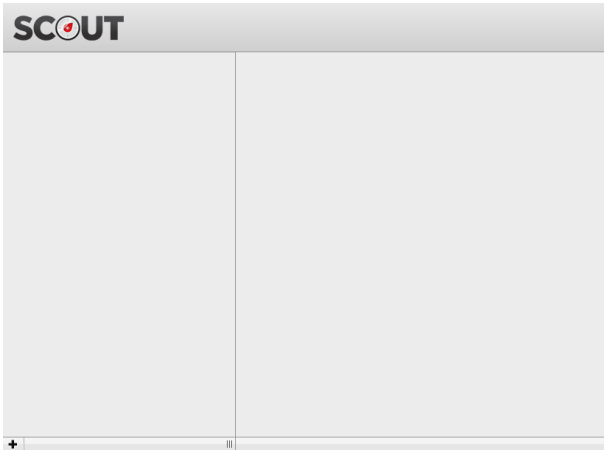
Bon, par contre, la conversion de l'un vers l'autre ne se fait pas de manière magique. Nous allons devoir faire des choses nouvelles qui vont peut-être vous faire un peu peur !

Installation de Sass

Sass a été écrit en Ruby et donc pour qu'il fonctionne, il va falloir installer un petit logiciel qui va nous permettre de bien traduire nos fichiers .scss vers .css. Ou alors, il existe une solution alternative qui a lieu à travers l'invite de commande et c'est cette solution-là qui est employée dans le milieu professionnel. Je vous laisse néanmoins les démarches ci-dessous pour installer un logiciel qui ne fait que la conversion Scss vers Css.

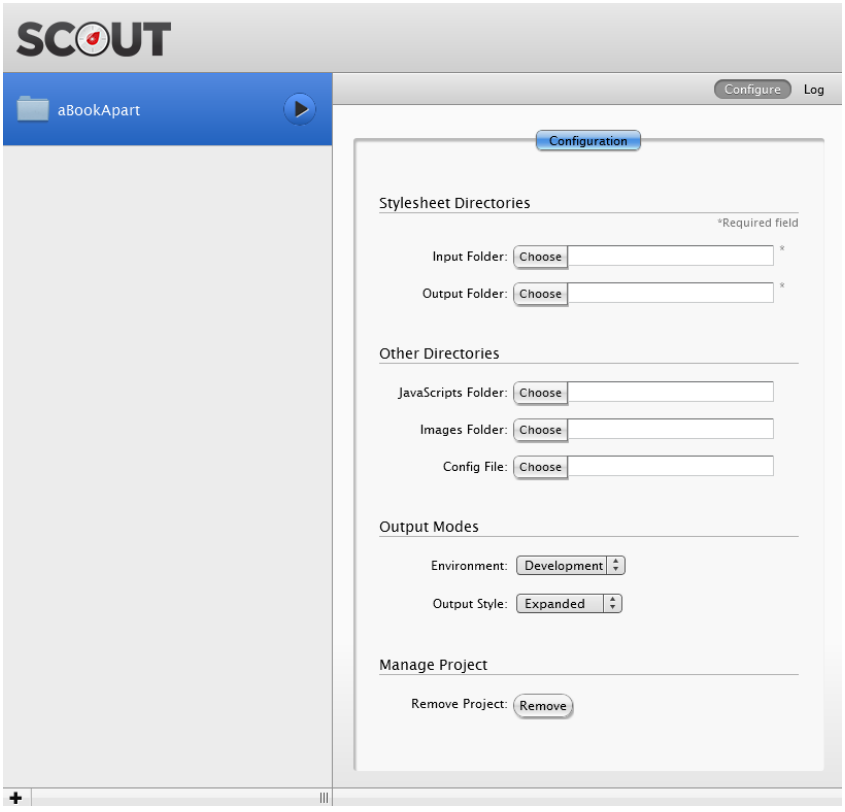
Ce petit installeur, vous allez le trouver ici : http://mhs.github.io/scout-app/ . Il est disponible pour Mac et pour Windows (yahoo).

Vous installez ce logiciel ensuite, vous l'ouvrez et vous arrivez sur cet écran :



Dans le coin inférieur gauche, vous avec un petit « + » qui va vous permettre d'ajouter un nouveau « projet » à Scout et de lui indiquer les informations de localisation dont il a besoin pour convertir vos fichiers .scss en .css.

1) Dans un premier temps, vous sélectionnez donc le répertoire dans lequel il va devoir travailler. (par exemple celui qui contient tout votre site web)
2) Ensuite vous arrivez sur cet écran de configuration :



3) Vous allez devoir indiquer le dossier dans lequel se trouvent vos feuilles de styles au format .scss (vous pourriez créer un dossier pour les séparer de vos fichiers .css)
4) Vous indiquez ensuite le dossier de « sortie » dans lequel vont être générées vos feuilles de styles classiques au format .css
5) Les options suivantes nous intéressent moins pour le moment (Javascript, images & config)
6) Dans « Output Modes » vous avez 2 options importantes qui ont un impact direct sur la forme que va revêtir votre fichier .css lors de sa création :

a. Environment : vous avec le choix entre dev et prod. Si vous choisissez dev, il va vous commenter chaque ligne générée avec Sass, c'est peu glamour, donc optez plutôt pour un environnement de production (mais retenez que l'option dev peut être intéressante dans les feuilles de styles kilométriques)
b. Output style : vous avez le choix entre nested (imbrication parent enfant avec tabulation), expanded (une propriété css par ligne), compact (toutes les propriétés css sur la même ligne) ou compressed (minification du fichier css – à faire à la fin d'un projet)

7) Réglez donc ces 2 options sur Production et expanded ou compact selon vos habitudes de travail.
8) Le dernier bouton « remove » ne sert qu'à la fin d'un projet, quand vous n'avez plus besoin de travailler dans votre dossier.

Et voilà, c'est tout, vous n'avez pas besoin de plus d'outils que cela pour pouvoir commencer à travailler avec Sass ! … Ah non je mens en fait ☺ Installez aussi dans Sublime Text l'extension qui va vous permettre d'écrire en .scss comme vous le faisiez en .css ☺

Maintenant, nous allons créer et tester notre premier fichier .scss !

Appuyez sur le bouton « play » dans scout, en vis-à-vis du projet que vous venez de créer, la machine se met en route … !

• Mon premier fichier .scss

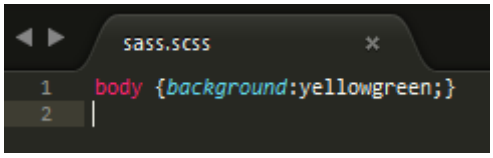
C'est un moment émotionnellement chargé que celui de créer votre premier fichier au format Sass. Pour cela, vous vous placez dans le dossier que vous avez indiqué comme étant le dossier source dans Scout et vous créez un fichier au format .scss avec le nom que vous voulez.

Vous l'enregistrez, mais, avant de faire CTRL+S, ouvrez Scout en parallèle et regardez ce qu'il va se passer quand vous allez faire CTRL+S sur votre fichier .scss :

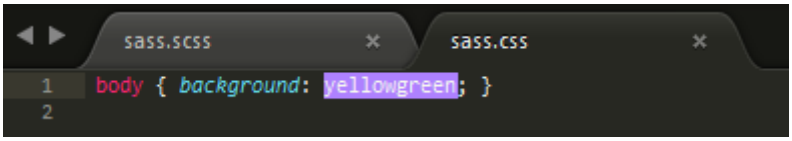
```
>>> Change detected at 14:52:58 to: sass.scss
create sass.css
```

Génial ! Il a repéré qu'il y a eu un changement dans le fichier sass.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 !

• 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%}
```

Magique !

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

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 ! ☺