sass

Description

Sass est un métalangage qui vient se greffer par dessus les css pour en améliorer la lisibilité et faciliter la maintenance.

Sass permet :

- d'utiliser des variables,
- d'utiliser des mixins pour éviter les répétitions,
- de modulariser en fichiers les css puis de les importer au moment de la compilation (donc, sans impact sur la performance), etc.
- d'imbriquer les sélecteurs pour mieux représenter leur ascendance / descendance,
- de régler au besoin le niveau de compression et de lisibilité des fichiers css
- …

Sass s'appuie sur le langage Ruby, mais aucune connaissance de Ruby n'est nécessaire pour utiliser Sass.

Les fichiers Sass ont une extension .sass ou .scss.

Conformément aux pratiques les plus répandues actuellement, nous allons privilégier l'extension .scss.

Quand Ruby compile les fichiers .scss, il crée ou met à jour le ou les fichiers .css.

Installer Sass sur Mac Os X

Pour pouvoir installer Sass, il faut d'abord que Ruby soit présent. Sur mac, c'est déjà le cas mais sur Windows, il faudra d'abord installé Ruby (http://rubyinstaller.org/downloads/).

Pour une procédure complète et adaptée à votre situation consulter la page du site officiel :

http://sass-lang.com/install

Premiers tests

Sur les postes des laboratoires, l'installation a déjà été faite, pour le vérifier, faites afficher les numéros de version.

Ouvrir le Terminal et entrer une à une les lignes suivantes. Ne tapez pas le $, c'est juste pour indiquer le début d'une ligne de commande. À la fin de la ligne utilisez la touche Enter.

```
$ ruby -v

$ sass -v
```

Pour cette première expérimentation, glisser dans votre dossier **Sites** le dossier materiel/**sandBox**.

Si le dossier Sites n'existe pas, créez-le :

```
$ mkdir ~/Sites
```

Assurez-vous que les permissions sur ce dossier sont correctes :

```
$ cd ~

$ ls -all
```

Dans Chrome, tapez **http://localhost/~user/** et cliquez sur le fichier sandBox/index.html.

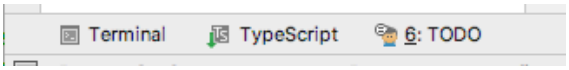
D'une part nous utiliserons :

- **Chrome** pour afficher le résultat de la page index.html.
- et un **éditeur de code** pour écrire nos styles en sass.
  D'autre part nous avons besoin d'un **terminal** pour passer les commandes de compilation à sass.

Pour simplifier, nous utiliserons PHPStorm qui possède un terminal intégré.

Glisser le dossier **SandBox** sur l'icône de **PHPStorm** qui est dans le DOCK. C'est important que ce soit le dossier SandBox qui soit glissé et non un dossier de niveau supérieur comme par exemples le dossier materiel ou un dossier crs04-sass.

Cliquer sur le bouton Terminal au bas, à gauche de l'éditeur :




```
$ cd Sites/sandBox/
```

Taper dans le terminal :

```
$ sass --watch scss:css
```

Ouvrir app.scss dans l'éditeur de code et taper :

```
/* app.scss */
h1 {
   color :red ;
}
```

Observer :

- la réponse de sass dans le terminal

```
>>> Sass is watching for changes. Press Ctrl-C to stop.
  directory css
      write css/app.css
      write css/app.css.map
```

- l'apparition du dossier css et à l'intérieur, du fichier app.css
- le fichier css dans l'éditeur de code
- l'apparition d'un dossier .sass-cache

La syntaxe de cette ligne de commande était :

```
sass --watch input-dir:output-dir
```

Pour procéder de fichier à fichier la syntaxe est similaire.

Exemple :

```
$ sass --watch styles.scss:styles.css
```

Stop watch !

Utiliser Ctrl-C dans le terminal pour arrêter le "watchage".

Pour ne pas créer des fichiers temporaires dans une cache (.sass-cache), ajouter le paramètre `no-cache`.

```
sass --watch path/input-dir:path/output-dir --no-cache
```

Modifier la lisibilité/compression du fichier css compilé

L'ajout du paramètre `style` à la commande `watch,` va permettre de modifier le degré de lisibilité/compression du fichier css compilé.

Quatre options : `nested` (par défaut), `expanded`, `compact`, `compressed`

Le mode **compressed** est à privilégier pour la compilation en situation de production puisqu'il permet de produire des fichiers plus légers en supprimant les tabulations et les sauts de ligne inutiles.

```
$ sass --watch scss:css --no-cache --style compressed
```

Le mode **expanded** est à privilégier pendant le développement car il offre une lisibilité maximale.

Pour en savoir plus sur les commandes sass, afficher l'aide –h (help) :

```
$ sass -h
```

Cet aide en ligne permet de découvrir des éléments de syntaxe supplémentaire comme par exemple le fait que le paramètre –style peut aussi s'écrire –t et le paramètre –no-cache peut aussi s'écrire –C, ainsi les lignes de commandes suivantes sont équivalentes :

```
$ sass --watch scss:css --no-cache --style compressed

$ sass --watch scss:css -C -t compressed
```

Commentaires

Deux types de commentaires peuvent être utilisés avec Sass

```
//  commentaires pour les développeurs, qui ne seront pas compilés
/* commentaires css classiques qui seront reproduits dans le .css */
```

Le nichage des règles avec Sass

Nichage des sélecteurs

On peut écrire des sélecteurs contextuels (ou descendants) en représentant l'arborescence par des accolades.

Par exemple, pour obtenir ceci:

```
// css
nav {
   width: 80%;
   height: 23px;
}
nav ul {
   list-style-type: none;
}
nav li {
   float: left;
}
nav li a {
   font-weight: bold;
}
```

```
// scss
nav {
   width: 80%;
   height: 23px;
   ul {
      list-style-type: none;
   }
   li {
      float: left;
      a { font-weight: bold; }
   }
}
```

Le nichage des sélecteurs est pratique, mais à utiliser avec beaucoup de modération !

Dans une approche BEM ou OOCSS, on cherche à rendre les aspects de présentation le plus indépendants possibles du DOM (de l'arbre HTML). Il faut donc éviter autant que possible les sélecteurs contextuels.

De plus, les sélecteurs contextuels complexifie la maintenance, introduise des problèmes de spécificité et alourdisse les fichiers css.

Une limite (GROS GROS MAX) à quatre niveaux de contextualisation est suggéré par l'équipe de sass.

Référence : *The Inception Rule* (http://thesassway.com/beginner/the-inception-rule)

```
~~main nav ul li a~~
main nav li a
```

Pour évaluez votre usage de la spécificité, testez votre CSS avec ce générateur de courbe :

https://jonassebastianohlsson.com/specificity-graph/

Nichage des propriétés

Vous pouvez aussi nicher les propriétés plutôt que de répéter les premiers mots des propriétés :

```
//scss
.fakeshadow {
   border: {
      style: solid;
      left: {
         width: 4px;
         color: #888;
      }
      right: {
         width: 2px;
         color: #ccc;
      }
   }
}
```

Sauvegarder et observer les changements dans le fichier app.css

```
//css
.fakeshadow {
   border-style: solid;
   border-left-width: 4px;
   border-left-color: #888;
   border-right-width: 2px;
   border-right-color: #ccc;
}
```

Nichage des pseudo-classes avec '&'

Il est possible de nicher des sélecteurs de pseudo-classes tels que :hover en utilisant l'**éperluette (&).**

Le **&** référence le sélecteur parent.

```
//scss
a {
   color: #ce4dd6;
   &:hover { color: #ffb3ff; }
   &:visited { color: #c458cb; }
}
```

```
//css
a {
  color: #ce4dd6;
}

a:hover {
  color: #ffb3ff;
}

a:visited {
  color: #c458cb;
}
```

Nichage de la syntaxe BEM

L'éperluette permet aussi de nicher les noms de classes formés avec la nomenclature BEM

```
// scss
.block {
  background-color: #2fd1af;
  height:10vh;
  &__element {
    background-color: #d17d71;
    &--modificateur {
      margin:15px;
      &-valeur{
        font-size:bigger;
      }
    }
  }
}
```

```
// css
.block {
  background-color: #2fd1af;
  height: 10vh;
}

.block__element {
  background-color: #d17d71;
}

.block__element--modificateur {
  margin: 15px;
}

.block__element--modificateur-valeur {
  font-size: bigger;
}
```

Nichage des requêtes médias

Au lieu de séparer de leur contexte les modifications à apporter à un module ou à un élément, Sass va nous permettre de définir chaque changement à même le contexte de l'élément. Cela donne un code beaucoup plus facile à comprendre et maintenir. Il est vrai que les requêtes medias vont se multiplier dans la compilation en CSS mais il a été démontré que cette multiplication a des effets insignifiants sur la performance.

Exemple :

```
// scss
h1{
  @include fontSize(32px);

   @media(min-width: 640px){
    @include fontSize(48px);
  }
}
```

```
// css
h1 {
  font-size: 32px;
  font-size: 3.2rem;
}
@media (min-width: 640px) {
  h1 {
    font-size: 48px;
    font-size: 4.8rem;
  }
}
```

Définir des variables

Sass utilise le symbole $ pour transformer quelque chose en variable.

```
// Définir des variables
$couleurDominante: navy;                    // valeur de couleur nommée
$couleurSecondaire: #000080;                // valeur de couleur hexadécimale
$chaine: " avec chaine ajoutee";            // chaine
$baseFontSize: 10px;                        // valeur numérique
$bordureMince: 1px solid $couleurDominante; // variable à valeurs multiples
$paddingNormal: 15px 10px 20px 10px;        // variable à valeurs multiples

// Utiliser des variables
h1, h2 {
    color: $couleurDominante;
}
#paragraphe {
    font-size: $baseFontSize;
    border: $bordureMince;
    padding: $paddingNormal;
}
#paragraphe:after {
    content: $chaine;
}
```

Les mixin

Les mixins permettent de définir un groupe de règles qui sont communes à plusieurs sélecteurs.

Lorsque vous vous rendez compte que vous répétez souvent un petit groupe de règles css, faites en mixin, ainsi, si vous changez un petit détail, vous pourrez le changer uniquement à un endroit : dans le mixin !

On déclare un mixin avec le symbole @ suivi du mot-clé mixin puis du nom subjectivement choisi pour décrire le groupe de règles.

```
// Définir un mixin
@mixin traitsCommuns {
    border-radius: 10px;
    border: 1px solid green;
    padding: 10px;
}

// Utiliser un mixin

header {
    color: #274D87;
    @include traitsCommuns;
}

footer {
    color: #3264AF;
    @include traitsCommuns;
}
```

Les mixins permettent de stocker des fragments de code (*snippets*) réutilisables tels que le *clearfix* :

```
@mixin clearfix {
  &:after {
    content: " ";
    display: table;
    clear: both;
  }
}
```

Voir le fichier _utilitaires.scss.

Voir aussi des bibliothèques :

Compass http://compass-style.org
Bourbon http://bourbon.io
Susy http://susy.oddbird.net

Les fonctions

Sass offre des fonctions mathématiques et des fonctions de coloration.

Exemples de fonctions mathématiques:

Division : http://sass-lang.com/documentation/file.SASS_REFERENCE.html#Division_and__

Pourcentage : http://sass-lang.com/documentation/Sass/Script/Functions.html#percentage-instance_method

Exemples de fonctions de coloration :

`lighten, darken, grayscale, saturate, desaturate, fadein, fadeout, fade, spin, mix…`

Guide visuel des fonctions de coloration : http://jackiebalzer.com/color

Les fonctions de coloration permettent de créer efficacement des effets
comme par exemple pour créer des pseudo-classes d'hyperliens :

Source : http://www.blogduwebdesign.com/ressources-css3/css-sass-augmentez-votre-productivite-dans-vos-integrations/571

Fonctions et mixins particulièrement utiles

```
// Déclaration de la fonction de calcul rem

@function calculateRem($size) {
  $remSize: $size / 10px;
  @return #{$remSize}rem;
}


/** Déclaration d'un mixin
  * qui utilise la fonction de calcul rem
  * et qui génère la solution de repli en pixel suivi par la valeur calculée en rem
  */

@mixin fontSize($size) {
  font-size: $size;
  font-size: calculateRem($size);
}


// Exemple d'utilisation SCSS
h5{
  @include fontSize(15px);
}
```

```
// css
h5{
   font-size :15px ;
   font-size :1.5rem ;
}
```

Fonction de calcul de grille fluide

```
/**
  * Déclaration de la fonction Fluidize
  * exprime la fameuse équation du responsive
  * valeurCibleEnPixels divisée par valeurDuContexteEnPixels multiplié par 100
  * = valeurCibleEn%
  */


@function fluidize($target, $context){
   @return ($target / $context) * 100%;
}


// Exemple d'utilisation SCSS

.sidebar{
   width: fluidize(350px, 1000px);
}
```

```
// css
.sidebar{
   width:35%;
}
```

Modularisation des styles et importation par Sass

Un des grands avantages de sass est de faciliter la **modularisation** des css.

Le fichier principal, **styles.css** ou **app.css**, devrait contenir une **table des matières** en début de document pour expliquer la structure de ses sections.

Chaque section est **chapeautée par son intitulé** qui doit correspondre à ce qui est annoncé dans la table des matières.

Il faut être rigoureux dans la création des entêtes de section car le but de la table des matières est de pouvoir naviguer rapidement à la section en faisant une sélection puis en utilisant la fonction *FIND*.

Chaque **section** contient un groupe de règles ou une instruction @import 'section'.

Exemple d'un fichier **app.scss** qui importera les fragments
`_normalize.scss, utilitaires/_u-variables.scss` et …

```
/**
  * ------------------
  * TABLE DES MATIÈRES
  * ------------------
  * LIB (Bibliothèques: dans un dossier bower_components)
  ** Normalize
  * Utilitaires
  ** variables   // L'ordre est important… les variables à importées avant les mixins !
  ** mixins
  * Sandbox
  */

/** Normalize **/
//@import 'chemin-vers-/normalize/import-now';

/** Sandbox **/
```

Au moment de la compilation, Sass inclus dans le fichier maître *app.css* les compilations de tous les fichiers .scss importés. Il créera aussi des fichiers sections.css inutiles si vous ne faites rien pour l'en empêcher.

Nommer les fragments avec un suffixe "_"

Pour ne pas créer ces fichiers inutiles, il faut utiliser une nomenclature prévue par Sass pour spécifier les fragments (partials), il faut ajouter un souligner devant le nom de fichier (section.scss devient _section.scss).

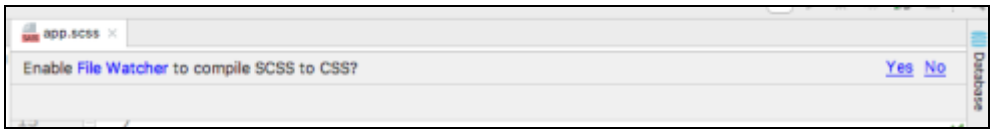
Syntaxe des importations sass

Notez que l'instruction @import utilisez par sass n'a pas besoin de contenir l'extension .scss ni les suffixes "_", de plus elle peut servir à l'importation de plus d'un fichier. Par exemple :

```
@import "utilitaires/u-variables", "utilitaires/u-mixins";
```

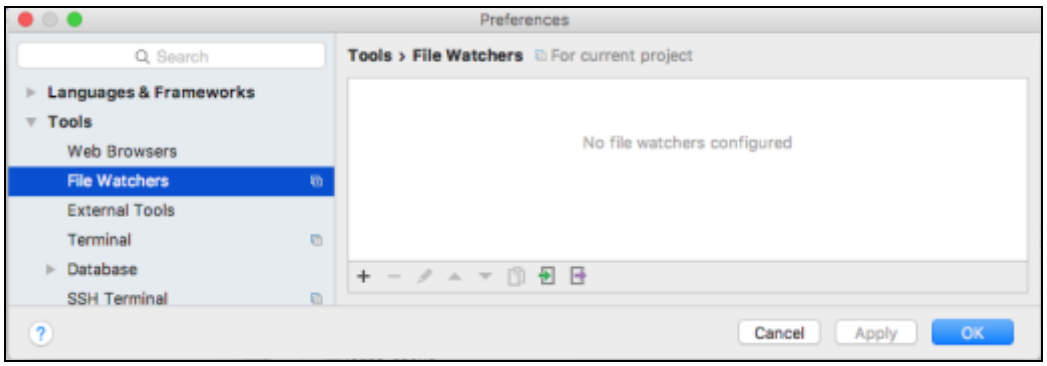
Annexe les Watchers de PHPStorm

Avez-vous remarqué cette petite question qui apparaît dans le haut des fichiers .scss dans PHPStorm ?



Si vous cliquez sur le lien File Watcher dans la question, cela ouvrira la fenêtre de

PHPStorm > préférences > Tools > File Watchers



Dans cette fenêtre, utilisez le bouton + pour ajouter un watcher vous permettra de vous passer du terminal puisque PHPStorm lancera lui-même la commande sass --watch.

Cependant, il est important de parmétrer le watcher autrement, les fichiers css vont se créer au même niveau que le fichier scss plutôt que dans le répertoire distinct des css.

Dans les champs de saisie **`Arguments`** et **`Output paths to refresh`**,

devant le paramètre **`$FileNameWithoutExtension`**, ajouter le lien relatif "../css/"