Sass

**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 **compiler** différents fichiers,   
  ce qui permet de **modulariser** les CSS,
* d’**imbriquer** les sélecteurs pour représenter leur hiérarchie,
* de régler le niveau de **compression** des fichiers css,
* etc…

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

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

# Installer Sass

Télécharger la version autonome de dart-sass qui corresponds à votre système d’opérations

<https://github.com/sass/dart-sass/releases/>

Placer le dossier dart-sass à la racine de votre projet Web

Tester dans le terminal :

|  |
| --- |
| # Affiche la version de Sass $ ./dart-sass/sass --version ruby 2.4.2p198 (2017-09-14 revision 59899) [x86\_64-darwin16] |

Ne tapez pas le signe **$**; celui-ci ne sert qu’à indiquer le début de la commande.   
Après avoir tapé la commande, appuyez sur **Enter** pour exécuter la commande.

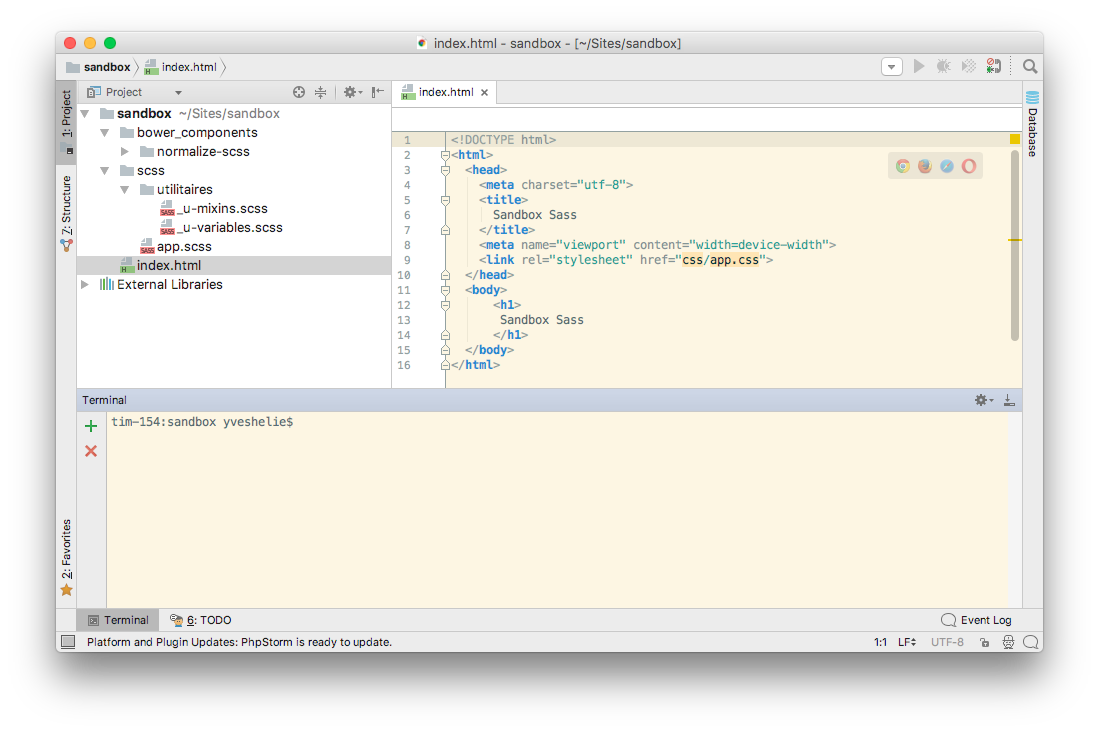
Pour développer nos feuilles de styles, nous utilisons principalement :

* un **éditeur de code** pour écrire nos styles en **Sass**;
* le **Terminal** pour passer les commandes de compilation à Sass;
* Chrome (**un navigateur**) pour afficher le résultat de la page.

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

**Glissez** le dossier **sandbox** sur l’icône de **PHPStorm** qui est dans le “dock”.

|  |
| --- |
| Il est important que bien glisser le dossier **sandbox** et non pas un dossier de niveau supérieur (ex. c03i\_sass). |

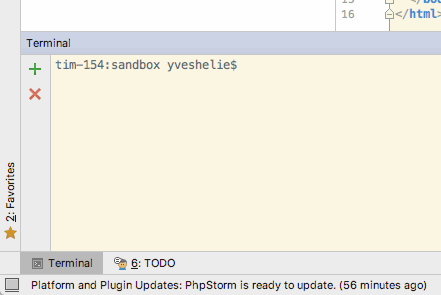


Éditeur de code

Arborescence

de fichiers

Terminal

Cliquez sur le bouton **Terminal** au bas, à gauche de l’éditeur :  
  


Par défaut, vous vous retrouverez dans le dossier **sandbox**. Si ce n’est pas le cas, naviguez vers celui-ci :

|  |
| --- |
| # Naviguer vers le dossier du “sandbox”  $ cd Sites/sandbox/ |

L’interface offerte dans **PhpStorm** est l’équivalent de l’application **Terminal** offerte par MacOS.

# Compiler les styles Sass

Tapez dans le terminal :

|  |
| --- |
| $ sass --watch scss:css |

Ouvrez **app.scss** dans l’éditeur de code et tapez :

|  |
| --- |
| h1 {  color : red; } |

Observez :

* la réponse de Sass dans le **terminal**;
* l’apparition du dossier **css** et à l’intérieur, du fichier **app.css**;
* le fichier **app.css** dans l’éditeur de code;
* l’apparition d’un dossier **.sass-cache**.

Plusieurs paramètres peuvent être utilisés lors de l’utilisation de la commande sass.

La syntaxe complète est :

|  |
| --- |
| sass --watch input:output --no-cache --style [style] |

|  |  |
| --- | --- |
| sass | Nom de la commande. |
| --watch | Indique à la commande qu’elle doit s’exécuter en mode “surveillance”. Le paramètre est facultatif; s’il est absent, la commande ne s’exécutera qu’une seule fois. |
| input:output | Détermine la provenance des fichiers Sass et la destination des fichiers CSS. Il est possible de cibler tant les dossiers que les fichiers. (scss:css *ou* scss/app.scss:css/app.css). |
| --no-cache | Pour ne pas créer de fichiers temporaires (.sass-cache).  Équivalent du paramètre -C. |
| --style [style] | Modifie la lisibilité et/ou la compression du fichier de sortie.  Équivalent du paramètre -t. Quatre options sont possibles : nested (par défaut), expanded, compact ou compressed. |

Pour arrêter la surveillance des fichiers, utilisez ctrl-c dans le terminal.

Pour en savoir plus sur les commandes Sass, affichez l’aide grâce au paramètre –h (help).

# Compression du fichier de sortie

Pendant le développement, on utilise le mode expanded car il offre une lisibilité maximale.

Lors de la mise en production, compressed est à privilégier puisqu’il permet de produire des fichiers plus légers en supprimant les tabulations et les sauts de ligne inutiles.

Au niveau des **commentaires**, deux types peuvent être utilisés :

* Les commentaires **bloc** sont des commentaires “généraux”
* Les commentaires en **ligne** s’adressent aux développeurs

|  |  |  |
| --- | --- | --- |
| Sass original | En mode expanded | En mode compressed |
| /\*\* Commentaire bloc \*\*/ // Commentaire en ligne  article {  h1 {  color: red;  } } | /\*\* Commentaire bloc \*\*/ article h1 {  color: red; } | article h1{color:red} |

# Le nichage des règles avec Sass

## Nichage des sélecteurs

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

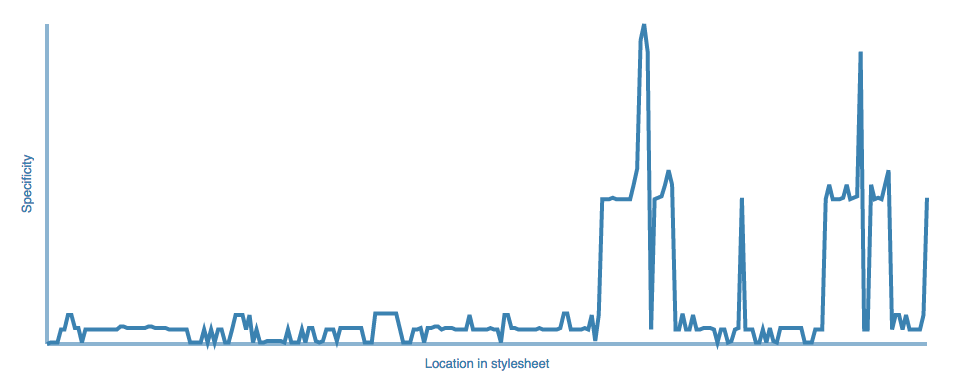
|  |  |
| --- | --- |
| CSS | Sass |
| nav {  width: 80%; }  nav ul {  list-style-type: none; }  nav li {  float: left; }  nav li a {  font-weight: bold; } | nav {   width: 80%;   ul {  list-style-type: none;  }   li {  float: left;   a {  font-weight: bold;  }   } } |

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 les plus indépendants possibles du DOM (de l’arbre HTML). Il faut donc éviter autant que possible les sélecteurs contextuels.

De plus, les sélecteurs contextuels complexifient la maintenance, introduisent des problèmes de spécificité et alourdissent les fichiers CSS.

Pour évaluez votre usage de la spécificité, testez votre CSS avec ce générateur de courbe : <https://jonassebastianohlsson.com/specificity-graph/>



**En théorie**, il n’y a pas de limites au nombre de niveaux de nichage qu’on peut effectuer.

**En pratique**, une limite (maximum!) de **quatre niveaux** de contextualisation est suggérée par l’équipe de Sass; viser **trois niveaux** serait encore mieux!

|  |  |
| --- | --- |
| À éviter | À privilégier |
| main {  nav {  ul {  li {  a {  [...]  }  }  }  } } | main {  nav {  ul {  [...]  }  li {  a {  [...]  }  }  } } |

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

## Nichage des propriétés

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

|  |  |
| --- | --- |
| Sass | CSS |
| .fakeshadow {   border: {   style: solid;   left: {  width: 4px;   color: #888;   }   right: {   width: 2px;   color: #ccc;  }   } } | .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 telles que :hover en utilisant l'esperluette (&). Le & fait référence au sélecteur parent.

|  |  |
| --- | --- |
| Sass | CSS |
| a {  color: #ce4dd6;  &:hover {  color: #ffb3ff;  }  &:visited {  color: #c458cb;  } } | a {  color: #ce4dd6; }  a:hover {  color: #ffb3ff; }  a:visited {  color: #c458cb; } |

## Nichage de la syntaxe BEM

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

|  |  |
| --- | --- |
| Sass | CSS |
| .block {  background-color: #2fd1af;  height:10vh;  &\_\_element {  background-color: #d17d71;  &--modificateur {  margin:15px;  }  } } | .block {  background-color: #2fd1af;  height: 10vh; }  .block\_\_element {  background-color: #d17d71; }  .block\_\_element--modificateur {  margin: 15px; } |

## Nichage des requêtes médias

Au lieu de séparer de leur contexte les modifications à apporter à un module ou à un élément, Sass nous permet de définir chaque changement à même le contexte de l’élément.

Le code sera donc beaucoup plus facile à comprendre et à maintenir.

|  |  |
| --- | --- |
| Sass | CSS |
| h1{  font-size: 32px;  @media(min-width: 640px){  font-size: 48px;  } } | h1 {  font-size: 32px; } @media (min-width: 640px) {  h1 {  font-size: 48px;  } } |

Il est vrai que cette approche multiplie les requêtes media dans la compilation CSS, mais il a été démontré que les impacts sur la performance étaient insignifiants.

# Définir des variables

Sass nous permet l’utilisation du symbole $ pour créer une variable.

|  |
| --- |
| $couleurDominante: navy; // couleur nommée $couleurSecondaire: #000080; // couleur hexadécimale $chaine: " avec chaine ajoutee"; // chaîne $baseFontSize: 10px; // valeur numérique $bordureMince: 1px solid $couleurDominante; // valeurs multiples $paddingNormal: 15px 10px 20px 10px; // valeurs multiples  h1, h2 {  color: $couleurDominante; } #paragraphe {  font-size: $baseFontSize;  border: $bordureMince;  padding: $paddingNormal;  &:after {  content: $chaine;   } } |

# Les mixins

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

Lorsqu’un petit groupe de règles CSS est régulièrement utilisé, il est sage d’en faire un mixin. Ainsi, s'il faut apporter un changement, il ne suffit que de le faire à un seul endroit!

## Déclaration d’un mixin

Un mixin est déclaré avec le symbole @ suivi du mot-clé mixin puis du nom subjectivement choisi pour décrire le groupe de règles.

|  |
| --- |
| @mixin traitsCommuns {  border-radius: 10px;  border: 1px solid green;  padding: 10px; } |

## Utilisation d’un mixin

Pour l’utiliser, il suffit d’utiliser le symbole @, suivi du mot-clé include, puis du nom du mixin.

|  |  |
| --- | --- |
| Sass | CSS |
| @mixin traitsCommuns {  border-radius: 10px;  border: 1px solid green;  padding: 10px; }  header {  color: #274D87;  @include traitsCommuns; }   footer {  color: #3264AF;  @include traitsCommuns; } | header {  color: #274D87;  border-radius: 10px;  border: 1px solid green;  padding: 10px; }   footer {  color: #3264AF;  border-radius: 10px;  border: 1px solid green;  padding: 10px; } |

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

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

Pour plus d’exemples, voir le fichier \_utilitaires.scss.

Des bibliothèques de mixins Sass sont également créées par des développeurs afin de faciliter le travail d’intégration.

* Compass : [http://compass-style.org](http://compass-style.org/)
* Bourbon : [http://bourbon.io](http://bourbon.io/)
* Susy : [http://susy.oddbird.net](http://susy.oddbird.net/) 🡨 obsolète maintenant!

# Les fonctions

En plus de permettre des opérations mathématiques, Sass offre des fonctions mathématiques et de coloration de base.

Plus de détails sur les fonctions de Sass : <http://sass-lang.com/documentation/Sass/Script/Functions.html>

## Fonctions mathématiques

|  |  |
| --- | --- |
| Sass | CSS |
| $largeurMaximale: 1000px;  .item1 {  width: $largeurMaximale/2;  }  .item2 {  width: percentage(4/12);  } | .item1 {  width: 500px; }  .item2 {  width: 33.3333333333%; } |

## Fonctions de coloration

Les fonctions de coloration facilitent la tâche des intégrateurs en permettant de décliner facilement une palette de couleurs.

Elles peuvent servir, par exemple, à créer facilement des effets de survol :

|  |  |
| --- | --- |
| Sass | CSS |
| $couleurPrincipale : red;  a {  color: $couleurPrincipale;  &:hover {  // Rendre la couleur 10% plus foncée  color: darken($couleurPrincipale, 10%);  } } | a {  color: red; } a:hover {  color: #cc0000; } |

Quelques exemples de fonctions de coloration : lighten, darken, grayscale, saturate, desaturate, fadein, fadeout, fade, spin, mix…

Pour en savoir plus sur les fonctions de coloration, consultez ce guide : <http://jackiebalzer.com/color>

## Fonctions personnalisées

Sass permet également la déclaration de nos propres fonctions utilitaires. Jumelées à la création de nos mixins, le tout devient un outil assez puissant afin de nous faciliter la tâche d’intégration!

### Fonction de calcul rem

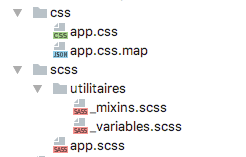
|  |  |
| --- | --- |
| Sass | CSS |
| @function calculateRem($size) {  $remSize: $size / 10px;  @return #{$remSize}rem; }  @mixin fontSize($size) {  font-size: $size;  font-size: calculateRem($size); }  h5{  @include fontSize(15px); } | h5 {  font-size: 15px;  font-size: 1.5rem; }  /\* Résultat compilé :  \* - Solution de repli  \* - Valeur en rem  \*/ |

### Fonction de calcul de grille fluide

|  |  |
| --- | --- |
| Sass | CSS |
| /\*\*  \* Déclaration de la fonction fluidize  \* valeurCible / valeurContexte \* 100 = valeurCible en %  \*/ @function fluidize($target, $context){   @return ($target / $context) \* 100%; } .sidebar{  width: fluidize(350px, 1000px); } | .sidebar {  width: 35%; } |

## Modularisation des styles et importation par Sass

Un des grands avantages de Sass est de faciliter la modularisation des CSS, car il nous permet d’importer différents fichiers Sass et de compiler le tout dans un seul fichier de sortie. Cette fonctionnalité nous permettra donc de morceler nos règles CSS en différents fichiers.



Fichiers source

Fichier compilé

Le fichier principal, **styles.css** ou **app.css**, contiendra 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 contiendra un groupe de règles ou une instruction @import ‘section’.

Exemple d’un fichier app.scss qui importera les fragments :

|  |
| --- |
| /\*\*  \* @author Prénom Nom - courriel  \* ------------------------------------  \* Table des matières  \* ------------------------------------  \* Bibliothèques (dans le dossier bower\_components)  \*\* Normalize  \* Utilitaires  \*\* Variables (attention à l’ordre d’importation!)  \*\* Mixins  \* Sandbox  \*/  /\* Bibliothèques \*/ /\*\* Normalize \*\*/ @import '../bower\_components/normalize-css/normalize';  /\* Utilitaires \*/ /\*\* Variables \*\*/ @import 'utilitaires/\_variables'; /\*\* Mixins \*\*/ @import 'utilitaires/\_mixins';  /\*\* Sandbox \*\*/  [...] |

## Processus de compilation des fichiers Sass

Au moment de la compilation, Sass compilera automatiquement dans le fichier maître **app.css** les fichiers .scss importés. Il créera également des fichiers .css pour chacun des fichiers trouvés dans le dossier **scss**.

### Importation de fichiers Sass

Pour importer un fichier dans Sass, utilisez l’instruction @import.

|  |
| --- |
| @import "utilitaires/\_variables.scss"; @import "utilitaires/\_mixins"; // L’extension .scss est facultative |

Notez que l’instruction @import utilisée par Sass n’a pas besoin de contenir l’extension .scss, et peut servir à l’importation de plus d’un fichier.

|  |
| --- |
| @import "utilitaires/\_variables", "utilitaires/\_mixins"; |

### 

### Fichiers fragments (partials)

Pour éviter de créer des fichiers .css inutiles pour chacun des fragments (partials), il faut utiliser une nomenclature prévue par Sass. Il suffit d’ajouter un souligné (*underscore*) devant le nom de fichier. Par exemple, **\_variables.scss** ou **\_mixins.scss**.

|  |  |
| --- | --- |
| Résultat sans soulignés | Résultat avec soulignés |
|  |  |

Lors de l’importation, il n’est pas nécessaire de mettre le souligné devant le nom du fichier.

|  |
| --- |
| @import "utilitaires/variables", "utilitaires/mixins"; |

## Annexes

### 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 paramètres : 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 paramétrer le *watcher* autrement, car par défaut les fichiers CSS vont se créer au même niveau que le fichier .scss plutôt que dans le répertoire distinct /css.

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

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

