eBook Gratuit

APPRENEZ sass

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#sass

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: sass

It is an unofficial and free sass ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official sass.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Démarrer avec sass

Remarques

Cette section fournit une vue d'ensemble de ce qu'est sass et de la raison pour laquelle un développeur peut vouloir l'utiliser.

Il devrait également mentionner tous les grands sujets dans sass, et établir un lien avec les sujets connexes. La documentation de sass étant nouvelle, vous devrez peut-être créer des versions initiales de ces rubriques connexes.

Pourquoi SASS?

- Fonction d'héritage
- Nous pouvons utiliser des instructions conditionnelles
- Plus fonctionnel que le `CSS` traditionnel
- Une manière efficace et claire d'écrire des `CSS`

Versions

Version	Date de sortie
3.4.22 (actuel)	2016-03-28
3.4.0	2014-08-18
3.3.0	2014-03-07
3.2.0	2012-08-10

Examples

Installer

En ce qui concerne l'utilisation de SASS, il existe plusieurs manières de configurer votre espace de travail. Certaines personnes préfèrent utiliser des outils de ligne de commande (probablement des utilisateurs Linux) et d'autres préfèrent utiliser des applications graphiques. Je vais couvrir les deux.

Outils de ligne de commande

La page 'Install SASS' de `sass-lang.com` couvre très bien. Vous pouvez utiliser SASS avec Ruby (qui peut être installé à partir d'un gestionnaire de paquets Linux ou vous pouvez télécharger le programme d'installation sous Windows). MacOS est livré avec Ruby préinstallé.

Une fois que vous avez installé Ruby, vous devez installer SASS (dans certains cas, `sudo` peut ne pas être nécessaire):

```
sudo gem install sass
```

Enfin, vous pouvez vérifier que vous avez installé SASS avec `sass -v` .

Applications GUI

Bien que vous puissiez utiliser un certain nombre d'applications GUI, je vous recommande Scout-App . Il construit et compresse automatiquement vos fichiers CSS pour vous, en sauvegarde de fichiers et supporte macOS, Windows et Linux.

Les variables

Si vous utilisez souvent une valeur, vous pouvez la stocker dans une variable. Vous pouvez par exemple l'utiliser pour définir des schémas de couleurs. Il vous suffira de définir votre schéma une seule fois et vous pourrez ensuite l'utiliser dans toutes vos feuilles de style.

Pour définir une variable, vous devez préfixer son nom avec le symbole $. (Comme vous le feriez en PHP.)

Vous pouvez stocker toute valeur de propriété CSS valide dans une variable. Comme les couleurs, les polices ou les URL.

Exemple 1:

```
$foreground: #FAFAFA;
$background: rgb(0, 0, 0);

body {
    color: $foreground;
    background-color: $background;
}

p {
    color: rgb(25, 25, 20);
    background-color: $background;
}
```

Importer

Supposons le scénario suivant: *Vous avez deux feuilles de style:* `_variables.scss` *et* `layout.scss` *. Logiquement, vous conservez toutes vos variables dans votre feuille de style variable, mais vous souhaitez y accéder depuis votre feuille de style de mise en page.*

> **Remarque:** vous pouvez remarquer que la feuille de style de variables a un trait de soulignement («_») avant son nom. C'est parce que c'est partiel - ce qui signifie qu'il va être importé.

> **sass-lang.com** dit **ceci** à propos des partiels: *Vous pouvez créer des fichiers Sass partiels contenant de petits extraits de CSS que vous pouvez inclure dans d'autres fichiers Sass. C'est un excellent moyen de modulariser votre CSS et de faciliter la maintenance. [...] Le trait de soulignement permet à Sass de savoir que le fichier n'est qu'un fichier partiel et qu'il ne doit pas être généré dans un fichier CSS. Les partiels Sass sont utilisés avec la directive @import.*

Les variables SCSS conviennent parfaitement à ce scénario. Supposons que votre `_variables.scss` ressemble à ceci:

```
$primary-color: #333;
```

Vous pouvez l'importer avec `@import` puis le nom de la feuille de style entre guillemets. Votre feuille de style de mise en page peut maintenant ressembler à ceci (notez qu'il n'ya pas de soulignement ni d'extension de fichier dans l'importation):

```
@import 'variables';
body {
  color: $primary-color;
}
```

Cela produirait quelque chose comme ceci:

```
body {
  color: #333;
}
```

Nidification

layout.scss

```
nav {
    ul {
        margin: 0;
        padding: 0;
        list-style: none;
        li {
            margin: 0 5px;
           }
       }
}
```

sortie

```
nav ul {
    margin: 0;
    padding: 0;
    list-style: none;
}
nav ul li {
    margin: 0 5px;
```

```
}
```

commentaires

SASS prend en charge deux types de commentaires:

- Commentaires en ligne - Ils ne couvrent qu'une seule ligne et sont généralement utilisés pour décrire une variable ou un bloc. La syntaxe est la suivante: `// Your comment here` (vous l'ajoutez avec une double barre oblique (`//`) et le reste de la ligne est ignoré par l'analyseur).
- Commentaires multilignes - Ils couvrent plusieurs lignes et sont généralement utilisés pour afficher un copyright ou une licence en haut d'un document. Vous pouvez ouvrir un bloc de commentaires multiligne avec `/*` et fermer un bloc de commentaires multiligne avec `*/` . Voici un exemple:

```
/*
   This is a comment
   It's a multiline comment
   Also a hiaku
*/
```

Lire Démarrer avec sass en ligne: https://riptutorial.com/fr/sass/topic/2045/demarrer-avec-sass

Chapitre 2: Boucles et conditons

Examples

En boucle

La directive `@while` boucle sur un bloc de code jusqu'à ce que la condition spécifiée devienne fausse. Dans l'exemple suivant, cette boucle s'exécutera jusqu'à `$font-size <= 18` tout en incrémentant la valeur de `$font-size` de 2.

```
$font-size: 12;

@while $font-size <= 18 {
    .font-size-#{$font-size} {
        font-size: ($font-size * 1px);
    }

    $font-size: $font-size + 2;
}
```

Sortie du code ci-dessus

```
.font-size-12 {
  font-size: 12px;
}

.font-size-14 {
  font-size: 14px;
}

.font-size-16 {
  font-size: 16px;
}

.font-size-18 {
  font-size: 18px;
}
```

pour la boucle

La directive `@for` vous permet de parcourir un code pour une quantité définie d'itérations et a deux formes:

- `@for <var> from <start> through <end> {}`
- `@for <var> from <start> to <end> {}`

La différence entre les deux formes est le *travers* et le *à* ; le mot - clé à *travers* comprendra la `<end>` dans la boucle où ne sera pas; using *through* équivaut à utiliser `>=` ou `<=` dans d'autres langages, tels que C ++, JavaScript ou PHP.

Remarques

- `<start>` et `<end>` doivent tous deux être des entiers ou des fonctions qui renvoient des entiers.
- Lorsque `<start>` est supérieur à `<end>` le compteur diminuera au lieu de l'incrément.

Exemple SCSS

```
@for $i from 1 through 3 {
  .foo-#{$i} { width: 10px * $i; }
}

// CSS output
.foo-1 { width: 10px; }
.foo-2 { width: 20px; }
.foo-3 { width: 30px; }
```

Directive conditionnelle (if)

La directive de contrôle `@if` évalue une expression donnée et si elle renvoie autre chose que `false` , elle traite son bloc de styles.

Sass Exemple

```
$test-variable: true !default

=test-mixin
  @if $test-variable
    display: block
  @else
    display: none

.test-selector
  +test-mixin
```

Exemple SCSS

```
$test-variable: true !default

@mixin test-mixin() {
  @if $test-variable {
    display: block;
  } @else {
    display: none;
  }
}

.test-selector {
  @include test-mixin();
}
```

Les exemples ci-dessus produisent les CSS suivantes:

```
.test-selector {
  display: block;
}
```

Chaque boucle

La directive @each vous permet de parcourir toute liste ou carte. Il se présente sous la forme de `@each $var or <list or map> {}` où `$var` peut être n'importe quel nom de variable et `<list or map>` peut être tout ce qui renvoie une liste ou une carte.

Dans l'exemple suivant, la boucle va parcourir la liste `$authors` attribuant chaque élément à `$author` , traiter son bloc de styles en utilisant cette valeur de `$author` et passer à l'élément suivant de la liste.

Exemple SCSS

```
$authors: "adam", "steve", "john";
@each $author in $authors {
    .photo-#{$author} {
      background: image-url("avatars/#{$author}.png") no-repeat
    }
}
```

Sortie CSS

```
.photo-adam {
  background: image-url("avatars/adam.png") no-repeat;
}
.photo-steve {
  background: image-url("avatars/steve.png") no-repeat;
}
.photo-john {
  background: image-url("avatars/john.png") no-repeat;
}
```

Affectation multiple

Les affectations multiples vous permettent d'accéder facilement à toutes les variables en déclarant plusieurs variables dans la directive `@each` .

Listes imbriquées

Pour accéder facilement à tous les éléments imbriqués, vous pouvez déclarer des variables distinctes correspondant à chaque élément imbriqué. Assurez-vous de disposer du nombre correct de variables et d'éléments imbriqués. Dans l'exemple suivant, chaque boucle effectue une itération dans une liste de trois éléments contenant chacun trois éléments imbriqués. Avoir la mauvaise quantité de variables déclarées entraînera une erreur de compilation.

```
@each $animal, $color, $cursor in (puma, black, default),
                                  (sea-slug, blue, pointer),
                                  (egret, white, move) {
  .#{$animal}-icon {
    background-image: url('/images/#{$animal}.png');
    border: 2px solid $color;
    cursor: $cursor;
```

```
  }
}
```

Plans

L'attribution multiple fonctionne également pour Maps mais se limite à deux variables, une variable pour accéder à la clé et une variable pour accéder à la valeur. Les noms `$key` et `$value` sont arbitraires dans l'exemple suivant:

```
@each $key, $value in ('first': 1, 'second': 2, 'third': 3) {
  .order-#{$key} {
    order: $value;
  }
}
```

Chaque boucle avec des cartes / valeurs de liste

Dans l'exemple ci-dessous, la valeur dans map `$color-array` est traitée comme une liste de paires.

Entrée SCSS

```
$color-array:(
  black: #4e4e4e,
  blue: #0099cc,
  green: #2ebc78
);
@each $color-name, $color-value in $color-array {
  .bg-#{$color-name} {
    background: $color-value;
  }
}
```

Sortie CSS

```
.bg-black {
  background: #4e4e4e;
}

.bg-blue {
  background: #0099cc;
}

.bg-green {
  background: #2ebc78;
}
```

Lire Boucles et conditons en ligne: https://riptutorial.com/fr/sass/topic/2671/boucles-et-conditons

Chapitre 3: Compas CSS3 Mixins

Introduction

Guide de mise en route à l'aide de l'existant Sass Compass. Compass est très utile avec CSS3 car il fournit des mixins pour écrire une ligne afin de prendre en charge tous les navigateurs utilisant les fonctionnalités CSS3. Il est également intéressant d'inclure des images de sprite.

Examples

Mettre en place l'environnement

Ouvrez votre ligne de commande

Installation à l'aide de Ruby

```
gem update --system

gem install compass
```

Créer un projet

```
compass create <myproject>
```

Cela initialisera un projet de compas. Il va ajouter un dossier appelé. Le dossier ressemblera à la structure suivante:

Dossier de fichiers	la description
toupet/	Vous placez des fichiers sass / scss dans ce dossier
feuilles de style /	Dans ce dossier, votre css compilé sera stocké
config.rb	Configurer la boussole - par exemple, chemin du dossier, compilation sass

---

Utiliser boussole

```
compass watch
```

Cela compilera vos fichiers sass à chaque fois que vous les modifierez. Le chemin du dossier sass peut être modifié dans le fichier config.rb

Utiliser CSS3 avec compas

Vous pouvez trouver une référence complète sur les composants CSS3 pris en charge sur cette page.

Pour utiliser CSS3 dans votre projet, Compass fournit des mixins pour prendre en charge les fonctionnalités CSS3 dans chaque navigateur. En plus de votre fichier Sass / Scss, vous devez spécifier que vous voulez utiliser la boussole

```
@import "compass/css3";
```

Rayon frontière

Incluez border-radius avec compas dans votre fichier sass:

```
div {
    @include border-radius(4px);
}
```

Sortie CSS

```
div {
  -moz-border-radius: 4px;
  -webkit-border-radius: 4px;
  border-radius: 4px;
}
```

Comme vous pouvez le voir, vous pouvez utiliser le nom CSS normal. Placez juste @include devant et utilisez () pour définir votre valeur.

Exemple de Flexbox

```
.row {
  @include display-flex;
  @include flex-direction(row);
}
```

Sortie CSS

```
.row {
  display: -webkit-flex;
  display: flex;
  -webkit-flex-direction: row;
  flex-direction: row;
```

```
}
```

Conclusion

Ce ne sont que deux exemples. Compass fournit beaucoup plus de mixins CSS3. Il est très pratique d'utiliser Compass et vous n'avez pas à craindre d'avoir oublié de définir un composant CSS3 pour un navigateur donné. Si le navigateur prend en charge la fonctionnalité CSS3, le compas le définira pour vous.

Lire Compas CSS3 Mixins en ligne: https://riptutorial.com/fr/sass/topic/10600/compas-css3-mixins

Chapitre 4: Convertir des unités

Examples

Convertir px en (r) em

Pour convertir px en em ou rem, vous pouvez utiliser la fonction suivante:

```
@function rem-calc($size, $font-size : $font-size) {
    $font-size: $font-size + 0px;
    $remSize: $size / $font-size;
    @return #{$remSize}rem;
}

@function em-calc($size, $font-size : $font-size) {
    $font-size: $font-size + 0px;
    $remSize: $size / $font-size;
    @return #{$remSize}em;
}
```

La `$font-size` est la `$font-size` la police d'origine.

Par exemple:

```
$font-size: 14;

body {
  font-size: #{$font-size}px;
  font-size: rem-calc(14px); // returns 1rem
  // font-size: rem-calc(28); // returns 2rem
}
```

Lire Convertir des unités en ligne: https://riptutorial.com/fr/sass/topic/6661/convertir-des-unites

Chapitre 5: Étendre / Hériter

Syntaxe

- `@extend .<className>`
- `@extend .<className>, .<className>`
- `@extend .<className> !optional`
- `@extend .<className>, .<className> !optional`

Paramètres

Paramètre	Détails
nom du cours	La classe que vous souhaitez étendre.

Remarques

La règle `@extend` Sass vous permet de partager les propriétés CSS entre plusieurs classes, en gardant le code DRY et plus facile à lire.

Examples

Étendre une classe

```
.message
  color: white

.message-important
  @extend .message
  background-color: red
```

Cela prendra tous les styles de `.message` et les ajoutera à `.message-important` . Il génère le CSS suivant:

```
.message, .message-important {
  color: white;
}

.message-important {
  background-color: red;
}
```

Étendre de plusieurs classes

```
.message
  color: white
```

```
.important
  background-color: red

.message-important
  @extend .message, .important
```

Dans le code ci-dessus, `@extend` est utilisé dans une ligne pour ajouter le code de plusieurs classes à `.message-important` , cependant, il est possible d'utiliser une extension par ligne comme ceci:

```
.message-important
    @extend .message
    @extend .important
```

L'une ou l'autre de ces méthodes génère les CSS suivantes:

```
.message, .message-important {
  color: white;
}

.important, .message-important {
  background-color: red;
}
```

Chaînage Étend

```
.message
  color: white
  background: black

.message-important
  @extend .message
  font-weight: bold

.message-error
  @extend .message-important
  font-style: italic
```

Ce code provoque l' `.message-error` à partir de `.message-important` , ce qui signifie qu'il contiendra du code provenant de `.message-important` et `.message` , puisque `.method-important` s'étend de `.message` . Cela se traduit par le CSS suivant:

```
.message, .message-important, .message-error {
  color: white;
  background: black;
}

.message-important, .message-error {
  font-weight: bold;
}

.message-error {
  font-style: italic;
```

```
}
```

Clause de non-responsabilité: Assurez-vous que la ou les classes que vous utilisez ne se produisent qu'une *fois* dans le code, sinon Sass peut générer des CSS désordonnés et compliqués.

Extensions facultatives

Parfois, vous pouvez souhaiter qu'un `@extend` soit facultatif et ne nécessite pas que la classe spécifiée existe dans votre code.

```
.message-important
  @extend .message !optional
  background: red
```

Cela se traduira par le CSS suivant:

```
.message-important {
  background: red;
}
```

Déni de responsabilité: Ceci est utile pendant le développement lorsque vous n'avez pas encore tout votre code écrit et que vous ne voulez pas d'erreurs, mais il devrait probablement être supprimé en production car cela pourrait entraîner des résultats inattendus.

Placeholders

Parfois, vous créerez des classes qui ne seront pas utilisées à part entière, mais uniquement à l'intérieur d'autres jeux de règles. Cela signifie que le fichier CSS compilé sera plus volumineux que nécessaire. Les sélecteurs d'espaces réservés résolvent ce problème.

Les sélecteurs d'espaces réservés sont similaires aux sélecteurs de classe, mais ils utilisent le caractère de pourcentage (%) au lieu du (.) Utilisé pour les classes. Ils n'apparaîtront pas dans le CSS compilé.

```
%button {
    border: 5px solid black;
    border-radius: 5px;
    margin: 0;
}

.error-button {
    @extend %button;
    background-color: #FF0000;
}

.success-button {
    @extend %button;
    background-color: #00FF00;
}
```

Cela compilera vers le CSS suivant:

```
.error-button, .success-button {
    border: 5px solid black;
    border-radius: 5px;
    margin: 0;
}

.error-button {
    background-color: #FF0000;
}

.success-button {
    background-color: #00FF00;
}
```

Extension du parent

Essayant généralement d'étendre le parent comme ça:

```
.parent {
  style: value;

  @extend &;
}
```

Entraînera une erreur, indiquant que le parent ne peut pas être étendu. Cela a du sens, mais il y a une solution de contournement. Stockez simplement le sélecteur parent dans une variable.

```
.parent {
  $parent: &;
  style: value;
  @extend #{&};
}
```

Cela ne sert à rien de le faire dans l'exemple ci-dessus, mais cela vous permet d'envelopper les styles parents à partir d'un mixin inclus.

Lire Étendre / Hériter en ligne: https://riptutorial.com/fr/sass/topic/2894/etendre---heriter

Chapitre 6: Installation

Remarques

Cela ne concerne que Ruby, qui est le principal compilateur SASS pour de nombreux systèmes, mais d'autres options existent. Un nœud très courant pour tout développeur de nœuds serait node-sass, ce qui pourrait être plus facile et beaucoup plus rapide pour de nombreux utilisateurs.

Examples

Mac

Ruby est préinstallé sur un ordinateur Mac.

Suivez les instructions ci-dessous pour installer Sass:

1. Ouvrir CMD
2. Exécuter `gem install sass`
3. Si vous obtenez un message d'erreur, essayez `sudo gem install sass`
4. Vérifiez qu'il fonctionne avec `sass -v`

Linux

Ruby devra être installé avant l'installation. Vous pouvez installer Ruby via le gestionnaire de paquets apt, rbenv ou rvm.

Puis courir

```
sudo su -c "gem install sass"
```

les fenêtres

Le moyen le plus rapide d'obtenir Ruby sur votre ordinateur Windows consiste à utiliser Ruby Installer . Il s'agit d'un programme d'installation en un seul clic qui vous permet de tout configurer rapidement. Après avoir installé Ruby, suivez les instructions ci-dessous pour installer Sass:

1. Ouvrir CMD
2. Exécuter `gem install sass`
3. Si vous obtenez un message d'erreur, essayez `sudo gem install sass`
4. Vérifiez qu'il fonctionne avec `sass -v`

Lire Installation en ligne: https://riptutorial.com/fr/sass/topic/2052/installation

Chapitre 7: Les fonctions

Syntaxe

- @function nom-fonction (paramètre) {/ * Corps de la fonction * /}

Examples

Les fonctions de base

Une fonction est similaire à un mixin mais elle n'ajoute aucun style, elle ne renvoie qu'une valeur. Les fonctions doivent être utilisées pour empêcher la logique répétée dans vos styles.

Sass possède des fonctions intégrées appelées à l'aide de la syntaxe de la fonction CSS standard.

```
h1 {
    background: hsl(0, 25%, 50%);
}
```

Les fonctions sont déclarées en utilisant la syntaxe ci-dessous,

```
@function multiply(x, y) {
    @return x * y;
}

// example use below
h1 {
    margin-top: multiply(10px, 2);
}
```

Dans le code ci-dessus, `@function` déclare une fonction et `@return` signifie la valeur `@return` .

Lire Les fonctions en ligne: https://riptutorial.com/fr/sass/topic/4782/les-fonctions

Chapitre 8: Les opérateurs

Examples

Opérateur d'assignation

Sass utilise les deux points (: opérateur d'attribuer des valeurs à des variables).

Exemple

```
$foreColor: red;

p {
    color: $foreColor;
}
```

Opérateurs arithmétiques

Sass prend en charge les opérateurs arithmétiques standard suivants:

Opérateur	La description
+	Une addition
-	Soustraction
*	Multiplication
/	Division
%	Reste

Exemples

```
p {
    font-size: 16px + 4px; // 20px
}
```

```
h3 {
    width: 2px * 5 + 12px; // 22px
}
```

```
h2 {
    width: 8px + (12px / 2) * 3; // 26px
}
```

L'ordre normal des opérations s'applique comme d'habitude.

Opérateurs de comparaison

Sass prend en charge tous les opérateurs de comparaison habituels: `<` , `>` , `==` `!=` , `<=` , `>=` .

Exemples

```
(10px == 10) // Returns true

("3" == 3) // Returns false
```

```
$padding: 10px;
$padding <= 8px; // Returns false
```

Lire Les opérateurs en ligne: https://riptutorial.com/fr/sass/topic/3047/les-operateurs

Chapitre 9: Les variables

Syntaxe

- $ *nom_variable* : *valeur* ;

Examples

Toupet

Les variables sont utilisées pour stocker une valeur une fois qui sera utilisée plusieurs fois dans un document Sass.

Ils sont principalement utilisés pour contrôler des choses telles que les polices et les couleurs, mais peuvent être utilisés pour toute valeur de propriété.

Sass utilise le symbole `$` pour faire quelque chose comme variable.

```
$font-stack: Helvetica, sans-serif
$primary-color: #000000

body
  font-family: $font-stack
  color: $primary-color
```

SCSS

Tout comme dans Sass, les variables SCSS sont utilisées pour stocker une valeur qui sera utilisée plusieurs fois dans un document SCSS.

Les variables sont principalement utilisées pour stocker les valeurs de propriété fréquemment utilisées (telles que les polices et les couleurs), mais peuvent être utilisées pour toute valeur de toute propriété.

SCSS utilise le symbole `$` pour déclarer une variable.

```
$font-stack: Helvetica, sans-serif;
$primary-color: #000000;

body {
  font-family: $font-stack;
  color: $primary-color;
}
```

Vous pouvez utiliser `!default` lors de la déclaration d'une variable si vous souhaitez affecter une nouvelle valeur à cette variable uniquement si elle n'a pas encore été affectée:

```
$primary-color: blue;
```

```
$primary-color: red !default; // $primary-color is still "blue"
$primary-color: green;        // And now it's green.
```

Portée variable

Les variables existent dans une portée spécifique, un peu comme dans JavaScript.

Si vous déclarez une variable en dehors d'un bloc, vous pouvez l'utiliser dans toute la feuille.

```
$blue: dodgerblue;

.main {
    background: $blue;

    p {
        background: #ffffff;
        color: $blue;
    }
}

.header {
    color: $blue;
}
```

Si vous déclarez une variable dans un bloc, elle ne peut être utilisée que dans ce bloc.

```
.main {
    $blue: dodgerblue;

    background: $blue;

    p {
        background: #ffffff;
        color: $blue;
    }
}

.header {
    color: $blue; // throws a variable not defined error in SASS compiler
}
```

Les variables déclarées au niveau de la feuille (en dehors d'un bloc) peuvent également être utilisées dans d'autres feuilles si elles sont importées .

Localize Variables avec directive @ at-root

La directive @ at-root peut être utilisée pour localiser les variables.

```
$color: blue;

@at-root {
  $color: red;

  .a {
```

```
    color: $color;
  }
  .b {
    color: $color;
  }
}

.c {
  color: $color;
}
```

est compilé pour:

```
.a {
  color: red;
}

.b {
  color: red;
}

.c {
  color: blue;
}
```

Interpolation

Les variables peuvent être utilisées en interpolation de chaîne. Cela vous permet de générer dynamiquement des sélecteurs, des propriétés et des valeurs. Et la syntaxe pour faire ainsi une variable est `#{$variable}` .

```
$className: widget;
$content: 'a widget';
$prop: content;

.#{$className}-class {
  #{content}: 'This is #{$content}';
}
// Compiles to

.widget-class {
  content: "This is a widget";
}
```

Vous ne pouvez cependant pas l'utiliser pour générer dynamiquement des noms de mixins ou de fonctions.

Variables dans SCSS

Dans SCSS, les variables commencent par `$` sign et sont définies comme des propriétés CSS.

```
$label-color: #eee;
```

Ils ne sont disponibles que dans les sélecteurs imbriqués où ils sont définis.

```
#menu {
    $basic-color: #eee;
    color: $basic-color;
}
```

S'ils sont définis en dehors des sélecteurs imbriqués, ils peuvent être utilisés partout.

```
$width: 5em;

#menu {
  width: $width;
}

#sidebar {
  width: $width;
}
```

Ils peuvent également être définis avec le drapeau `!global` , auquel cas ils sont également disponibles partout.

```
#menu {
  $width: 5em !global;
  width: $width;
}

#sidebar {
  width: $width;
}
```

Il est important de noter que les noms de variables peuvent utiliser des tirets et des traits de soulignement de manière interchangeable. Par exemple, si vous définissez une variable appelée `$label-width` , vous pouvez y accéder sous la forme `$label_width` et vice versa.

Lire Les variables en ligne: https://riptutorial.com/fr/sass/topic/2180/les-variables

Chapitre 10: Mettre à jour la version Sass

Introduction

Mettez à jour votre version de Sass en utilisant gem / ruby

Examples

les fenêtres

Vous pouvez vérifier la version de Sass en utilisant `sass -v`

Mettre à jour toutes les mises à `gem update` ruby gems

Mettre à jour uniquement Sass `gem update sass`

Linux

Vous pouvez vérifier la version de Sass en utilisant `sass -v`

Mettre à jour toutes les mises à jour de ruby gems `sudo gem update`

Mettre à jour uniquement Sass `sudo gem update sass`

Lire Mettre à jour la version Sass en ligne: https://riptutorial.com/fr/sass/topic/10599/mettre-a-jour-la-version-sass

Chapitre 11: Mixins

Syntaxe

- `@mixin mixin-name ($argument1, $argument, ...){ ... }`

Examples

Créer et utiliser un mixin

Pour créer un mixin, utilisez la directive `@mixin` .

```
@mixin default-box ($color, $borderColor) {
    color: $color;
    border: 1px solid $borderColor;
    clear: both;
    display: block;
    margin: 5px 0;
    padding: 5px 10px;
}
```

Vous pouvez spécifier une liste d'arguments entre parenthèses après le nom de mixin. N'oubliez pas de démarrer vos variables avec `$` et séparez-les par des virgules.

Pour utiliser le mixin dans un autre sélecteur, utilisez la directive `@include` .

```
footer, header{ @include default-box (#ddd, #ccc); }
```

Les styles du mixin seront désormais utilisés dans le `footer` et l'en- `header` , avec la valeur `#ccc` pour la variable `$color` et `#ddd` pour la variable `$borderColor` .

Mixin avec argument variable

Il y a des cas dans les mixins où il peut y avoir un ou plusieurs arguments lors de son utilisation. Prenons un cas de `border-radius` où il peut y avoir un seul argument comme `border-radius:4px;` ou plusieurs arguments comme `border-radius:4px 3px 2px 1px;` .

Le mélange traditionnel avec les arguments de mot clé sera comme ci-dessous: -

```
@mixin radius($rad1, $rad2, $rad3, $rad4){
 -webkit-border-radius: $rad1 $rad2 $rad3 $rad4;
 -moz-border-radius: $rad1 $rad2 $rad3 $rad4;
 -ms-border-radius: $rad1 $rad2 $rad3 $rad4;
 -o-border-radius: $rad1 $rad2 $rad3 $rad4;
 border-radius: $rad1 $rad2 $rad3 $rad4;
}
```

Et utilisé comme

```
.foo{
    @include radius(2px, 3px, 5px, 6px)
}
```

L'exemple ci-dessus est complexe (à coder, lire et maintenir) et si vous ne pouvez pas transmettre une seule valeur ou deux valeurs, cela générera une erreur et pour utiliser **une, deux ou** trois valeurs, vous devrez définir trois autres mixins.

En utilisant la **variable Argument,** vous n'avez pas à vous soucier du nombre d'arguments que vous pouvez transmettre. Les arguments variables peuvent être déclarés en définissant un nom de variable suivi de **trois points (...)** . Voici un exemple d'argument de variable.

```
@mixin radius($radius...)
{
    -webkit-border-radius: $radius;
    -moz-border-radius: $radius;
    -ms-border-radius: $radius;
    -o-border-radius: $radius;
    border-radius: $radius;
}
```

Et utilisé comme

```
.foo{
    @include radius(2px 3px 5px 6px)
}
.foo2{
    @include radius(2px 3px)
}
.foo3{
    @include radius(2px)
}
```

L'exemple ci-dessus est beaucoup plus simple (pour coder, maintenir et lire), vous n'avez pas à vous soucier du nombre d'arguments à venir - est-ce **un ou plusieurs** .

S'il y a plus d'un argument et que vous voulez accéder au second argument, vous pouvez le faire en écrivant *`propertyname : nth(variable_name, 2)`* .

Défauts sensibles

SASS vous donne la possibilité d'omettre n'importe quel paramètre, sauf ceux que vous souhaitez remplacer. Reprenons l'exemple de la `default-box` :

```
@mixin default-box ($color: red, $borderColor: blue) {
    color: $color;
    border: 1px solid $borderColor;
    clear: both;
    display: block;
    margin: 5px 0;
    padding: 5px 10px;
}
```

Ici, nous allons maintenant appeler le mixin en écrasant le deuxième paramètre

```
footer, header{ @include default-box ($borderColor: #ccc); }
```

la valeur de `$borderColor` est `#ccc` , tandis que `$color` reste `red`

Arguments optionnels

Les arguments optionnels de SASS vous permettent d'utiliser un paramètre uniquement si vous spécifiez sa valeur; sinon, il sera ignoré. Prenons un exemple du mixin suivant:

```
@mixin galerie-thumbnail ($img-height:14em, $img-width: null) {
    width: $img-width;
    height: $img-height;
    outline: 1px solid lightgray;
    outline-offset: 5px;
}
```

Donc, un appel à

```
.default {
  @include galerie-thumbnail;
}
.with-width {
  @include galerie-thumbnail($img-width: 12em);
}
.without-height {
  @include galerie-thumbnail($img-height: null);
}
```

va simplement sortir les éléments suivants dans le fichier CSS:

```
.default {
  height: 14em;
  outline: 1px solid lightgray;
  outline-offset: 5px;
}

.with-width {
  width: 12em;
  height: 14em;
  outline: 1px solid lightgray;
  outline-offset: 5px;
}

.without-height {
  outline: 1px solid lightgray;
  outline-offset: 5px;
}
```

SASS ne génère pas de propriétés dont la valeur est `null` , ce qui est très utile lorsque nous devons inclure un argument facultatif dans notre appel ou non.

Directive @content

Les mixins peuvent recevoir un bloc de code compatible SASS, qui devient alors disponible dans le mixin en tant `@content` directive `@content` .

```
@mixin small-screen {
  @media screen and (min-width: 800px;) {
    @content;
  }
}

@include small-screen {
  .container {
    width: 600px;
  }
}
```

Et cela produirait:

```
  @media screen and (min-width: 800px;) {
    .container {
      width: 600px;
    }
  }
```

Les mixins peuvent utiliser la directive `@content` et accepter toujours les paramètres.

```
@mixin small-screen($offset) {...
```

Lire Mixins en ligne: https://riptutorial.com/fr/sass/topic/2131/mixins

Chapitre 12: Nidification

Examples

Nidification de base

Chaque fois que vous déclarez une nouvelle règle *dans* une autre règle, elle est appelée imbrication. Avec l'imbrication de base, comme illustré ci-dessous, le sélecteur imbriqué sera compilé en tant que nouveau sélecteur CSS avec tous ses parents ajoutés, séparés par un espace.

```
// SCSS
.parent {
  margin: 1rem;

  .child {
    float: left;
  }
}

// CSS output
.parent {
  margin: 1rem;
}

.parent .child {
  float: left;
}
```

Profondeur de nidification

L'imbrication est une fonctionnalité très puissante, mais doit être utilisée avec prudence. Il peut arriver assez facilement et rapidement que vous commenciez à nidifier et à continuer d'inclure tous les enfants dans un nid, un nid ou un nid. Permettez-moi de démontrer:

```
// SCSS
header {
  // [css-rules]

  .holder {
    // [css-rules]

    .dropdown-list {
      // [css-rules]

      ul {
        // [css-rules]

        li {
          margin: 1rem 0 0 1rem;
        }
      }
```

```
    }
  }
}

// CSS output of the last rule
header .holder .dropdown-list ul li {
  margin: 1rem 0 0 1rem;
}
```

Problèmes

Spécificité

Le `li` de l'exemple ci-dessus a un jeu de `margin` . Disons que nous voulons remplacer cela plus tard dans une requête média.

```
@media (max-width: 480) {

  // will not work
  .dropdown-list ul li {
    margin: 0;
  }

  // will work
  header .holder .dropdown-list ul li {
    margin: 0;
  }
}
```

Ainsi, en imbriquant trop profondément, vous devrez vous nicher à nouveau chaque fois que vous voudrez écraser une certaine valeur. Pire encore, c'est souvent la règle `!important`

```
@media (max-width: 480) {

  // BIG NO-NO, don't do this
  .dropdown-list ul li {
    margin: 0 !important;
  }
```

Pourquoi la règle `!important` Important est-elle une mauvaise idée?

Vous devriez écrire votre SCSS de manière à ce que ces solutions ne soient même pas nécessaires. Utiliser `!important` sur un problème aussi mineur vous conduira déjà dans un trou de lapin!

Réutilisabilité

Ceci est assez similaire au problème de la spécificité, mais mérite d'être souligné séparément. Si vous donnez un style à un bouton ou même à un menu déroulant, vous pouvez réutiliser ces styles ailleurs sur votre page.

En imbriquant trop profondément, vos styles ne sont liés qu'aux éléments situés à l'intérieur du parent le plus externe (l'élément situé en haut de votre SCSS). Cela vous amène à copier des styles et à les coller ailleurs. Peut-être dans une autre règle imbriquée.

Vos feuilles de style deviendront de plus en plus grandes et difficiles à gérer.

À quelle profondeur devriez-vous nicher?

La plupart des guides de style définissent la profondeur maximale à 2. C'est un bon conseil en général, car il y a très peu d'occasions où vous souhaiteriez nicher plus profondément. La plupart du temps, 2 suffisent.

Imbrication avec @ at-root

L'imbrication est probablement le plus souvent utilisée pour créer des sélecteurs plus spécifiques, mais elle peut également être utilisée simplement pour l'organisation du code. En utilisant la directive `@at-root` , vous pouvez «sauter» de l'endroit où vous l'avez imbriqué dans votre Sass, ce qui vous ramène au niveau supérieur. Cela vous permet de garder les styles groupés sans créer plus de spécificité que nécessaire.

Par exemple, vous pourriez avoir quelque chose comme ceci:

```
.item {
    color: #333;

    @at-root {
        .item-wrapper {
            color: #666;

            img {
                width: 100%;
            }
        }
    }

    .item-child {
        background-color: #555;
    }
}
```

Cela compilerait à ceci:

```
.item {
  color: #333;
}
.item-wrapper {
  color: #666;
}
.item-wrapper img {
  width: 100%;
}
.item .item-child {
```

```
  background-color: #555;
}
```

En faisant cela, tous nos styles liés à la classe `.item` sont réunis dans le SCSS, mais nous n'avons pas nécessairement besoin de cette classe dans chaque sélecteur.

Contextes exclus

Par défaut, les déclarations à l'intérieur de `@at-root` apparaîtront dans n'importe quel contexte. Cela signifie que les règles à l'intérieur d'un bloc `@media` , par exemple, resteront là.

```
@media print {
  .item-wrapper {
    @at-root {
      .item {
        background: white;
      }
    }
  }
}

// Will compile to
@media print {
  .item {
    background: white;
  }
}
```

Ce comportement n’est pas toujours souhaitable, vous pouvez donc exclure le contexte du média en transmettant le `media` à l’option `without` directive de `@at-root` directive `@at-root` .

```
@at-root (without: media) {..
```

Pour plus d'informations, voir la **documentation officielle**

Le sélecteur parent (&)

L'imbrication est idéale pour conserver les sélecteurs associés afin que les futurs développeurs puissent mieux comprendre votre code. Le sélecteur parent, représenté par une esperluette ("&"), peut vous aider dans des situations plus complexes. Il y a plusieurs façons de l'utiliser.

Créez un nouveau sélecteur qui requiert à la fois le sélecteur parent et un autre sur le même élément en plaçant le nouveau sélecteur directement après un sélecteur parent.

```
// SCSS
.parent {

  &.skin {
    background: pink;
  }
}
```

```
// CSS output
.parent.skin {
  background: pink;
}
```

Faites apparaître le parent après un sélecteur imbriqué dans le CSS compilé en plaçant le sélecteur parent *après* le sélecteur imbriqué.

```
// SCSS
.parent {

  .wrapper & {
    border: 1px solid black;
  }
}
```

```
// CSS output
.wrapper .parent {
  border: 1px solid black;
}
```

Etats et pseudo-éléments

Outre l'utilisation de l'imbrication pour les classes et les enfants, l'imbrication avec le sélecteur parent est généralement utilisée pour combiner les états de `:active` `:hover` et `:focus` pour les liens.

```
// SCSS
a {
  color: blue;

  &:active,
  &:focus {
    color: red;
  }

  &:visited {
    color: purple;
  }
}
```

```
// CSS output
a {
  color: blue;
}

a:active,
a:focus {
  color: red;
}

a:visited {
  color: purple;
}
```

De même, vous pouvez styliser des pseudo-éléments en les imbriquant avec le sélecteur parent.

```
// SCSS
.parent {

  &::after {
    display: table;
    clear: both;
    content: '';
  }

  &::only-child {
    font-weight: bold;
  }
}
```

```
// CSS output
.parent::after {
  display: table;
  clear: both;
  content: '';
}

.parent::only-child {
  font-weight: bold;
}
```

Propriétés d'imbrication

Certaines propriétés CSS appartiennent à un espace de noms, par exemple `border-right` appartient au namespace `border` . Pour écrire moins, nous pouvons utiliser l'imbrication de propriété et ignorer ces préfixes, même sur plusieurs niveaux.

Si nous devions créer une bordure à droite et à gauche d'une classe nommée `.borders` nous pourrions écrire ceci:

```
 //SCSS
 .borders {
    border: 2px dashed blue;
    border: {
        left: 1px solid black;
        right: 1px solid red;
    }
  }

 // CSS output
 .borders {
    border: 2px dashed blue;
    border-left: 1px solid black;
    border-right: 1px solid red;
  }
```

Cela nous évite d'avoir à écrire `border-right` et `border-left` , mais nous écrivons toujours du code répétitif avec les lignes `1px solid black` `1px solid red` et `1px solid red` . Nous pouvons écrire des

CSS encore moins répétitifs avec:

```
// SCSS
.borders {
  border: 2px dashed blue {
    left: 1px solid black;
    right: {
      color: red;
    }
  }
}

// CSS output
.borders {
   border: 2px dashed blue;
   border-left: 1px solid black;
   border-right-color: red;
}
```

Lire Nidification en ligne: https://riptutorial.com/fr/sass/topic/2178/nidification

Chapitre 13: Partials et Import

Examples

Importer

L'utilisation de `@import` vous permet de diviser vos fichiers en plusieurs fichiers plus petits. Cela a du sens, car vous pouvez conserver une meilleure structure pour vos feuilles de style et éviter les fichiers très volumineux.

Exemple

Disons que vous avez quelques fichiers.

```
- application.scss
- header.scss
- content
  |-- article.scss
  '-- list.scss
- footer.scss
```

Votre feuille de style principale `application.scss` peut importer tous les fichiers ainsi que définir ses propres styles.

```
// application.scss
// Import files:
@import 'header.scss';
@import 'content/article.scss';
@import 'content/list.scss';
@import 'footer.scss';

// other styles in application.scss
body {
  margin: 0;
}
```

Notez que vous pouvez également omettre l'extension `.scss` pour pouvoir écrire `@import 'header';` au lieu de `@import 'header.scss'` .

Cela conduit à `application.scss` ayant tous les `.scss` importés inclus dans le fichier compilé que vous servez au client (navigateur). Dans ce cas, votre fichier compilé serait `application.css` que vous incluez dans votre code HTML.

```
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="/application.css?v=18c9ed25ea60">
  </head>
  <body>
    ...
  </body>
```

```
</html>
```

Bien que vous utilisiez plusieurs fichiers, vous ne fournissez qu'un seul fichier, ce qui élimine le besoin de plusieurs requêtes (une pour chaque fichier) et accélère le temps de chargement de vos visiteurs.

Principaux avantages

- Meilleure structure pour le développement en utilisant un dossier et plusieurs fichiers
- Servir un seul fichier au client (navigateur)

Partiels

Vous pouvez créer des fichiers partiels contenant des extraits plus petits de vos feuilles de style. Cela vous permet de modulariser votre CSS et permet une meilleure structure de vos feuilles de style. Un partial est un fichier Sass nommé avec un trait de soulignement principal, à savoir: `_partial.scss` . Le trait de soulignement permet à Sass de savoir que le fichier spécifique est partiel et qu'il ne sera pas généré dans un fichier CSS.

Les partiels Sass sont destinés à être utilisés avec la directive `@import` . Lorsque vous utilisez `@import` , vous pouvez omettre le trait de soulignement principal.

Exemple

En supposant que vous ayez une structure de fichiers avec des partiels comme celui-ci

```
- main.scss
- _variables.scss
- content
  |-- _buttons.scss
  '-- _otherelements.scss
```

Vous pouvez inclure ces partiels dans votre fichier `main.scss` comme suit (les `main.scss` soulignement et les extensions de fichier sont omis dans cet exemple):

```
// main.scss - Imports:
@import 'variables';
@import 'content/buttons';
@import 'content/otherelements';
```

Lire Partials et Import en ligne: https://riptutorial.com/fr/sass/topic/2893/partials-et-import

Chapitre 14: Scss mixins utiles

Examples

Pure css3 flèches de pointeur avec bordure de contour

!!! Le conteneur doit être placé relativement ou absolument

$ direction - haut, bas, gauche, droite

$ margin - marge par le bord dans la **direction $** . Pour le haut et le bas, c'est de gauche à droite. Pour la gauche et la droite, c'est de haut en bas.

$ colors - first est une couleur de bordure, second - est une couleur d'arrière-plan (peut-être vaut-il mieux hériter de la couleur d'arrière-plan d'un parent)

$ arrowSide - est une taille relative d'une flèche

$ isInset - la flèche est à l'intérieur (true) ou à l'extérieur de son conteneur

Voici un Plunker fonctionnel https://plnkr.co/edit/PRF9eLwmOg8OcUoGb22Y?p=preview

```
%pointer-core {
    content: " ";
    position: absolute;
    border: solid transparent;
    z-index: 9999;
}

@mixin pointer($direction, $margin: 10px, $colors: (#999, $gray), $arrowSide: 8px, $isInset:
false){

    $opposites: (
        top: bottom,
        bottom: top,
        left: right,
        right: left
    );

    $margin-direction: (
        top: left,
        bottom: left,
        left: top,
        right: top
    );

    &:before {
        @extend %pointer-core;
        border-width: $arrowSide;

        @if $isInset {
            border-#{$direction}-color: nth($colors, 1);
            #{$direction}: -1px;
        }
```

```
        @else
        {
            border-#{map-get($opposites, $direction)}-color: nth($colors, 1);
            #{map-get($opposites, $direction)}: 100%;
        }

        #{map-get($margin-direction, $direction)}: 0;

        margin-#{map-get($margin-direction, $direction)}: $margin - 1;
    }

    &:after {
        @extend %pointer-core;
        border-width: $arrowSide - 1;

        @if $isInset {
            border-#{$direction}-color: nth($colors, 2);
            #{$direction}: -1px;
        }
        @else
        {
            border-#{map-get($opposites, $direction)}-color: nth($colors, 2);
            #{map-get($opposites, $direction)}: 100%;
        }

        #{map-get($margin-direction, $direction)}: 0;

        margin-#{map-get($margin-direction, $direction)}: $margin;
    }
}
```

Exemple de pointeur d'info-bulle

```
$color-purple-bg: #AF6EC4;
$color-purple-border: #5D0C66;


$color-yellow-bg: #E8CB48;
$color-yellow-border: #757526;

.tooltip {
    position: relative;

    &--arrow-down {
        @include pointer('bottom', 30px, ($color-purple-border, $color-purple-bg), 15px);
    }

    &--arrow-right {
        @include pointer('right', 60px, ($color-yellow-border, $color-yellow-bg), 15px);
    }
}
```

Lire Scss mixins utiles en ligne: https://riptutorial.com/fr/sass/topic/6605/scss-mixins-utiles

Chapitre 15: SCSS vs Sass

Examples

Principales Différences

Bien que les gens disent souvent `Sass` comme le nom de ce CSS-préprocesseur, ils signifient souvent le `SCSS` -syntax. `Sass` utilise l'extension de fichier `.sass` , tandis que `SCSS` - `Sass` utilise l'extension `.scss` . Ils sont tous deux appelés "Sass".

D'une manière générale, le `SCSS` -syntax est plus couramment utilisé. `SCSS` ressemble à un CSS ordinaire avec plus de fonctionnalités, alors que `Sass` est très différent des CSS classiques. Les deux syntaxes ont les mêmes capacités.

Syntaxe

Les principales différences sont que `Sass` n'utilise pas de accolades ou de points-virgules, ce que fait `SCSS` . `Sass` est également sensible aux espaces, ce qui signifie que vous devez indenter correctement. Dans `SCSS` , vous pouvez formater et indenter vos règles à votre guise.

SCSS:

```
// nesting in SCSS
.parent {
  margin-top: 1rem;

  .child {
    float: left;
    background: blue;
  }
}
```

TOUPET:

```
// nesting in Sass
.parent
  margin-top: 1rem

  .child
    float: left
    background: blue
```

Après la compilation, les deux produiront le même CSS suivant:

```
.parent {
  margin-top: 1rem;
}
.parent .child {
  float: left;
  background: blue;
}
```

Mixins

`Sass` tendance à être la syntaxe la plus "paresseuse". Rien n'illustre mieux ce que vous définissez et incluez les mixins.

Définir un mixin

`=` Comment vous définissez un mixin dans `Sass` , `@mixin` dans `SCSS` .

```
// SCSS
@mixin size($x: 10rem, $y: 20rem) {
  width: $x;
  height: $y;
}

// Sass
=size($x: 10rem, $y: 20rem)
  width: $x
  height: $y
```

Y compris un mixin

`+` Comment vous inclure dans `Sass` , `@include` dans `SCSS` .

```
// SCSS
.element {
  @include size(20rem);
}

// Sass
.element
  +size(20rem)
```

Plans

En ce qui concerne les cartes, `SCSS` est généralement la syntaxe la plus simple. Comme `Sass` est basé sur un retrait, vos cartes doivent être enregistrées sur une seule ligne.

```
// in Sass maps are "unreadable"
$white: ( white-50: rgba(255, 255, 255, .1), white-100: rgba(255, 255, 255, .2), white-200:
rgba(255, 255, 255, .3), white-300: rgba(255, 255, 255, .4), white-400: rgba(255, 255, 255,
.5), white-500: rgba(255, 255, 255, .6), white-600: rgba(255, 255, 255, .7), white-700:
```

```
rgba(255, 255, 255, .8), white-800: rgba(255, 255, 255, .9), white-900: rgba(255, 255, 255, 1
)
```

Comme vous pouvez formater votre code sur plusieurs lignes avec `SCSS` , vous pouvez formater vos cartes pour les rendre plus lisibles.

```
// in SCSS maps are more readable
$white: (
  white-50: rgba(255, 255, 255, .1),
  white-100: rgba(255, 255, 255, .2),
  white-200: rgba(255, 255, 255, .3),
  white-300: rgba(255, 255, 255, .4),
  white-400: rgba(255, 255, 255, .5),
  white-500: rgba(255, 255, 255, .6),
  white-600: rgba(255, 255, 255, .7),
  white-700: rgba(255, 255, 255, .8),
  white-800: rgba(255, 255, 255, .9),
  white-900: rgba(255, 255, 255, 1)
);
```

commentaires

Les commentaires dans Sass vs Scss sont largement similaires, sauf en ce qui concerne les lignes multiples. `SASS` lignes multiples `SASS` sont sensibles à l'indentation, tandis que `SCSS` s'appuie sur des terminateurs de commentaires.

Commentaire sur une seule ligne

style.scss

```
// Just this line will be commented!
h1 { color: red; }
```

style.sass

```
// Exactly the same as the SCSS Syntax!
h1
  color: red
```

Commentaire multi-lignes

style.scss

Initiateur: `/*`

Terminator: `*/`

```
/* This comment takes up
 * two lines.
 */
h1 {
   color: red;
}
```

Cela va `h1` éléments `h1` avec la couleur **rouge** .

style.sass

Maintenant, `SASS` a *deux* initiateurs, mais pas de terminateurs respectifs. Les commentaires multilignes dans `SASS` sont sensibles aux **niveaux d'indentation** .

Initiateurs: `//` et `/*`

```
// This is starts a comment,
   and will persist until you
   return to the original indentaton level.
h1
  color: red
```

Cela va `h1` éléments `h1` avec la couleur **rouge** .

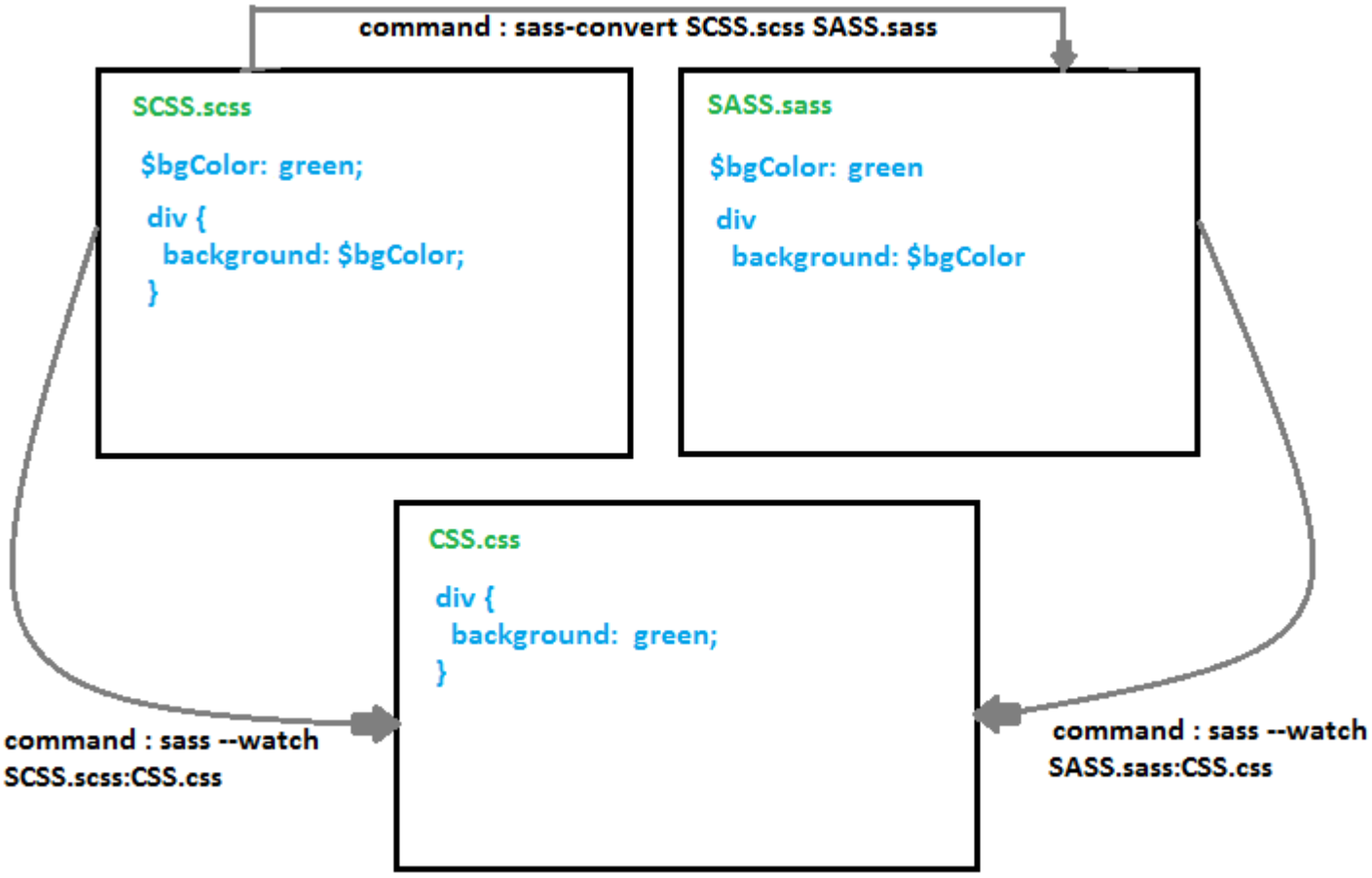
La même chose peut être faite avec l'initiateur `/*` :

```
/* This is starts a comment,
   and will persist until you
   return to the original indentaton level.
h1
  color: red
```

Alors voilà! Les principales différences entre les commentaires dans `SCSS` et `SASS` !

Comparaison entre SCSS et SASS

- `SCSS` syntaxe `SCSS` ressemble plus à une syntaxe `CSS` mais la syntaxe `SASS` est un peu différente de `SCSS` mais les deux produisent exactement le même code `CSS` .
- Dans `SASS` nous ne `SASS` pas les propriétés de style avec un point-virgule (`;`) mais dans SCSS, nous finissons les propriétés de style avec (`;`).
- Dans `SCSS` nous avons utilisé le paranthesis `{}` pour fermer les propriétés de style, mais dans `SASS` nous n'utilisons pas de `paranthesis` .
- `Indentation` est très importante dans `SASS` . Il définira les propriétés imbriquées dans la `class` ou l' `id` de l'élément.
- En `scss` nous pouvons définir plusieurs variables sur une seule ligne mais dans `SASS` nous ne pouvons pas le faire.



pour la syntaxe de la boucle

Avec la sortie de sass 3.3 et plus la version, la syntaxe des conditions @if et else est la même. nous pouvons maintenant utiliser des expressions avec non seulement **scss** mais aussi **sass** .

syntaxe sass

```
@for $i from 1 through 3 {
  .item-#{$i} { width: 2em * $i; }
}
```

Compilé pour

```
.item-1 {
  width: 2em;
}
.item-2 {
  width: 4em;
}
.item-3 {
  width: 6em;
}
```

la syntaxe scss

```
@for $i from 1 through 3 {
  .item-#{$i} { width: 2em * $i; }
}
```

compilé pour

```
.item-1 {
  width: 2em;
}
.item-2 {
  width: 4em;
}
.item-3 {
  width: 6em;
}
```

Lire SCSS vs Sass en ligne: https://riptutorial.com/fr/sass/topic/2428/scss-vs-sass

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec sass	Angelos Chalaris, Benolot, Christopher, Community, Kartik Prasad, Rohit Jindal, SamJakob, Stewartside
2	Boucles et conditons	Akash Kodesia, allejo, Angelos Chalaris, GMchris, MMachinegun, ScottL
3	Compas CSS3 Mixins	Schlumpf
4	Convertir des unités	SuperDJ
5	Étendre / Hériter	Euan Williams, GMchris, user2367593
6	Installation	Angelos Chalaris, Pyloid, Stewartside
7	Les fonctions	Euan Williams, GMchris, Hudson Taylor, Pyloid
8	Les opérateurs	Angelos Chalaris, Hudson Taylor, Pyloid
9	Les variables	Daniyal Basit Khan, evuez, GMchris, Hudson Taylor, jaredsk, Pyloid, Stewartside, yassh
10	Mettre à jour la version Sass	Schlumpf
11	Mixins	Akash Kodesia, Angelos Chalaris, GMchris, Hudson Taylor, Ninda, Roxy Walsh
12	Nidification	aisflat439, alexbea, Amy, Christopher, Devid Farinelli, GMchris, Hudson Taylor, John Slegers, MMachinegun
13	Partials et Import	Angelos Chalaris, Hudson Taylor, MMachinegun
14	Scss mixins utiles	Kindzoku
15	SCSS vs Sass	75th Trombone, Everettss, Jared Hooper, MMachinegun, Muzamil301, Pyloid, Robotnicka, Rohit Jindal