Sass

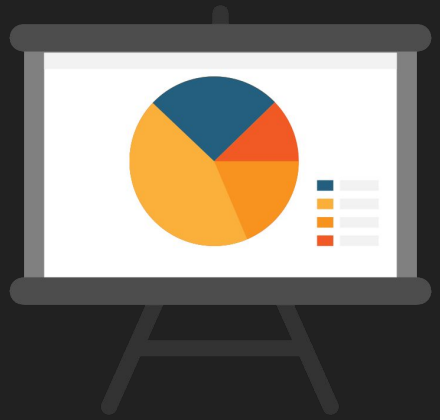
Syntactically Awesome Style Sheets

Rachid EDJEKOUANE (edjek@hotmail.fr)

Prérequis

- Avoir les bases en CSS
- Connaître les principales propriétés CSS

Introduction



Un préprocesseur CSS ?

Un préprocesseur CSS est un programme ou outil informatique permettant de générer dynamiquement des fichiers CSS.

- L'objectif est d'améliorer l'écriture de ces fichiers, en apportant plus de flexibilité au développeur web.
- La plupart des préprocesseurs CSS ajoutent des fonctionnalités qui n'existent pas en CSS pur, telles que : variable, mixins, sélecteur d'imbrication, etc…
- Ces fonctionnalités rendent la structure CSS plus lisible et plus facile à maintenir.

Inconvénients

Tout le monde ne plaide pas forcément en faveur de l'utilisation des préprocesseurs CSS :

- "Ils complexifient CSS", la syntaxe des préprocesseurs étant plus complexe que le CSS de base.
- "Ils n'ajoutent pas de fonctionnalités CSS aux CSS", les préprocesseurs ne pouvant générer que du CSS standard.
- "Ils ne font pas (toujours) gagner de temps", leur utilisation ne se justifiant pas pour des projets de taille modeste.
- "Ils peuvent être dangereux pour le standard CSS", leurs fonctionnalités risquant de faire stagner l'évolution du CSS.

Préprocesseurs CSS

Quelques exemples de préprocesseurs CSS.
(Sass, Less, Stylus, PostCss)

Un peu d’histoire : Sass

Le projet Sass a été créé en 2006.

Sass (Syntactically awesome stylesheets) est un langage de script préprocesseur qui est compilé ou interprété en CSS.

Sass est disponible en deux syntaxes.

- La syntaxe originale, appelée "syntaxe indentée" qui utilise l'indentation pour séparer les blocs de code et les sauts de ligne pour les séparer les directives.
- La nouvelle syntaxe, "SCSS", utilise les mêmes séparateurs de blocs que CSS.

Sass ou Scss

2 syntaxes, 1 seul préprocesseur :

.scss

```scss
.button {
    background: cornflowerblue;
    border-radius: 5px;
    padding: 10px 20px;

    &:hover {
        cursor: pointer;
    }

    &:disabled {
        cursor: default;
        background: grey;
        pointer-events: none;
    }
}
```

.sass

```sass
.button
    background: cornflowerblue
    border-radius: 5px
    padding: 10px 20px

    &:hover
        cursor: pointer

    &:disabled
        cursor: default
        background: grey
        pointer-events: none
```

Pourquoi SCSS d'abord

Nous allons utiliser SCSS pour plusieurs raisons :

- lisibilité : la syntaxe est très proche du CSS
- courbe d'apprentissage : il n'ajoute que quelques fonctionnalités supplémentaires en plus du CSS
- compatibilité; un fichier CSS est un fichier SCSS valide
- ressources : de nombreux articles en ligne à lire et des bibliothèques open source à utiliser
- évolutivité : il est facile de passer de SCSS à Sass

DRY : Don’t Repeat Yourself

DRY est une philosophie en programmation informatique consistant à éviter les répétitions de code au sein d’une application afin de faciliter la maintenance, le test, le débogage et les évolutions de cette dernière.

Sass : les fonctionnalités

Ces fonctionnalités nous aideront à mieux gérer nos styles, à éviter de se répéter, à simplifier la mise en place d'un style global, facilement modifiable et thématique.

On disposera d'un outil qui, une fois un fichier .scss créé, le compilera en un fichier CSS classique utilisable pour un site web.

Que nous offre Sass?

Sass permet d'utiliser des fonctionnalités qui n'existent pas en CSS

- Partials : Un partial est un fichier Sass nommé avec un underscore. Il permet à Sass de savoir que le fichier spécifique est partiel et qu'il ne sera pas généré dans un fichier CSS.
- Variables : On peut stocker des couleurs, des tailles, etc …
- Nesting : Une façon d'imbriquer des éléments à styliser
- Mixins : Écrire des morceaux de code réutilisables
- Placeholders : Étendre des propriétés d'un élément
- Fonctions et Conditions : "if", "for", "each", "while" sont disponibles

Découverte

Sass : comment ça marche ?

Plusieurs possibilités pour utiliser Sass mais nous utiliserons une extension de VSCode :

Live Sass Compiler

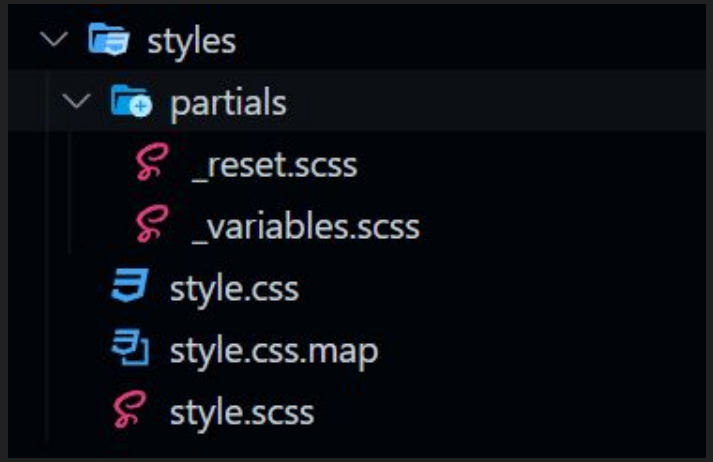
En pratique

Sass : les partials

Afin de modulariser votre CSS, et de le maintenir plus facilement, on peut utiliser les partials :

- Les fichiers partials sont des fichiers Sass, dont le nom est préfixé par un underscore "_".
- L'underscore permet à Sass de savoir que le fichier n'est qu'un fichier partiel, qu'il ne faudra pas compiler en un fichier CSS à part entière.

Sass : les partials




```scss
@import 'partials/reset';
@import 'partials/variables';

body {
    font-size: 18px;
}

h1 {
    font-weight: normal;
}
```

Attention : l'usage d'un @import fait que les styles importés se propage (pollution du namespace), préférer le @use

Sass : les variables

Une variable permet de stocker la valeur attribuée à une propriété CSS.

Elle pourra être utilisée à différents endroits du code.

Nous pouvons créer un fichier spécifique aux variables, et les importer dans notre fichier principal: style.scss.

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

```scss
@import 'partials/reset';
@import 'partials/variables';

body {
    font-size: 18px;
    color: $primary-color;
}

h1 {
    font-weight: normal;
}
```

Sass : le nesting

Sass permet d'imbriquer vos sélecteurs CSS en conservant la structure présente dans l'HTML.

Ceci est appelé nesting.

```html
<section>
    <h2>Exemple d'utilisation SASS :</h2>
    <p>
        Basique exemple de code en HTML et CSS,
        afin de tester les fonctionnalités de SASS
    </p>
</section>
<div>
    <button>Simple boutton</button>
</div>
```

```scss
section {
    background-color: $primary-color;
    padding: 20px;
    h2 {
        color: $secondary-color;
    }
    p {
        color: $secondary-color;
        padding: 10px;
    }
}

div {
    display: flex;
    button {
        background-color: $primary-color;
    }
}
```

Sass : les mixins

Une mixin vous permet de créer des groupes de déclarations CSS réutilisables.

Vous pouvez même passer des valeurs pour rendre votre mixin plus flexible.

Vous les définissez en utilisant @mixin et vous les intégrez en utilisant @include

Sass : les mixins

```scss
@mixin layout($padding, $margin) {
    padding: $padding;
    max-width: 1080px;
    margin: $margin auto;
    border-radius: 5px;
}
section {
    @include layout(50px, 20px);
    background-color: $primary-color;
    padding: 20px;
    h2 {
        color: $secondary-color;
    }
    p {
        color: $secondary-color;
        padding: 10px;
    }
}
```

Sass : placeholder selector

Un "placeholder selector", une sorte de sélecteur abstrait qui n'est là que pour être étendu avec @extend.

On définit un tel sélecteur comme on définirait un sélecteur de classe, en remplaçant le "." par un "%".

```scss
%font-size {
    font-size: 30px;
}
section {
    @include layout(50px, 20px);
    background-color: $primary-color;
    padding: 20px;
    h2 {
        @extend %font-size;
        color: $secondary-color;
    }
    p {
        @extend %font-size;
        color: $secondary-color;
        padding: 10px;
    }
}
```

Sass : les fonctions prédéfinies

Sass possède plusieurs fonctions prédéfinies.

Pour utiliser une fonction prédéfinie, il suffit de l'appeler en utilisant son nom et de lui passer éventuellement les données dont elle a besoin en argument.

La syntaxe générale pour appeler une fonction est la suivante : nom_de_fonction(argument-1, argument-2).

Sass : les fonctions prédéfinies

Dans cet exemple à chaque compilation, la couleur du bouton changera aléatoirement :

```
div {
    @include layout(30px, 20px);
    display: flex;
    button {
        margin: 20px auto;
        padding: 10px 30px;
        border-radius: 5px;
        background-color: rgb(random(255), random(255), random(255));
    }
    :hover {
        background-attachment: $btn;
    }
}
```

Sass : les fonctions

Nous allons également pouvoir créer nos propres fonctions Sass.

Pour cela, nous allons devoir utiliser la règle @function suivi de la définition de notre fonction, c'est-à-dire de son nom, de la liste des arguments dont elle a besoin pour fonctionner et des instructions qu'elle doit exécuter.

Sass : les fonctions

```scss
@function multiplication($nombre, $fact) {
    @return $nombre * $fact;
}

@function whatColor($condition) {
    @if ($condition== 'green') {
        @return green;
    } @else {
        @return blue;
    }
}

@while ($margin < $width/2) {
    $margin: $margin + 1;
}
```

Sass : Lectures complémentaires

Consultez certains des liens pour mieux comprendre le fonctionnement de Sass:

Sass : Site officiel

Sass guidelines : Un guide de style engagé pour du code Sass sain, maintenable et extensible.

Les fonctions prédéfinies : Rappel de qq fonctions prédéfinies

Le préprocesseur Sass : Tuto vidéo de Grafikart

```
@import 'merci';
```