Fonctionalités avancées

SCSS est un langage reprenant beaucoup de fonctionnalités des autres langages de programmation. Si ce que nous avons présenté jusqu'à maintenant peut suffire pour créer des feuilles de style capables de décrire n'importe quelle UI, ses propriétés permettent des choses encore plus puissantes.

Le texte comme variable

Il est possible, en SCSS, de remplacer n'importe quelle propriété zone de texte CSS avec une variable, un peu comme le ferait un moteur de templating. Pour ce faire, on utilise la syntaxe suivante:

```
$scss_var: "some_string";

.my_class {
    #{$scss_var}: css_value
}
```

Vous pouvez utiliser cette syntaxe n'importe où dans un fichier SCSS, tant que la syntaxe du document est conforme à la syntaxe générale d'un document CSS. Plus concrètement, cela veut dire que vous avez le droit de faire quelque chose comme cela :

```
$flex: "flex";
$display: "display";

#{$flex}{
    #{$display}: #{$flex};
}
```

Ce morceau de code SCSS vous offre une classe utilitaire que vous pouvez ajouter sur n'importe quel élément HTML, et rendra le contenu CSS suivant:

```
.flex{
    display: flex;
}
```

Par contre, vous ne pouvez pas faire quelque chose comme ceci :

```
$flex: "flex";
$display: "display : flex;";

.#{$flex}{
```

```
    #{$display}
}
```

Cela ne compilera pas.

Pour l'instant cela ne semble pas forcément très utils, mais combiné à d'autres fonctionalités, cela vous permettra de générer par exemple un grand nombre de classes utilitaires à la volée.

Reprenons notre index.html. Après la mise en place du reset CSS, nous pouvons voir que notre Hello World n'a plus aucun style par défaut, et nous voudrions pouvoir générer des classes utilitaires pour les éléments de texte, allant de xs à xxl.

Voilà comment nous pourrions nous débrouiller :

```
@use 'sass:list';

$heading-sizes: xs, s, m, l, xl, xxl;

@for $i from 1 to 7 {
    .h-#{list.nth($heading-sizes, $i)}{
        $remSize: 0.5+calc($i / 4);
        font-size: #{$remSize}rem;
    }
}
```

Ici, cela nous produit le résultat suivant:

```
.h-xs {
  font-size: 0.75rem;
}

.h-s {
  font-size: 1rem;
}

.h-m {
  font-size: 1.25rem;
}

.h-l {
  font-size: 1.5rem;
}

.h-xl {
  font-size: 1.75rem;
}

.h-xxl {
```

```
  font-size: 2rem;
}
```

Il y a ici plusieurs choses que nous n'avons pas vu : l'utilisation de types complexes pour les variables (ici une liste, mais il existe aussi des maps, null, etc.)

Les listes

Pour déclarer une liste en scss, on utilise la syntaxe suivante :

```
$my_list: el_1, ..., el_n;
```

En important sass:list dans un module scss, il devient possible d'utiliser certaines propriétés sur les listes

- `list.nth(my_list, n)` : permet de récupérer le nième élément de la liste. Attention : les listes commencent à l'index 1 en SCSS.
- `append(my_list, element)` : permet d'ajouter un élément à une liste.
- `list.length(my_list)` : permet de récupérer la taille de la liste.
- `list.index(my_list, element)` : permet de récupérer l'index de l'élément dans une liste.

Les Maps

Il est possible d'utiliser une structure de données permettant de lier des clés à des valeurs en SCSS. Cela se fait de la façon suivante :

```
$my_list: (key_1 : value_1, ..., key_n : val_n);
```

Comme pour les listes, il faut ajouter une directive @use pour pouvoir utiliser des méthodes sur nos maps. Cela se fait de la façon suivante:

```
@use "sass:map";
```

L'itération

Il est possible d'itérer sur chaque élément d'une liste ou chaque clé et valeur d'une map avec la directive @each ou la directive for

Cela peut se faire des deux façons suivantes:

```
@use "sass:list";
```

```
$heading_sizes: xs, s, m, l, xl, xxl;

@for $i from 1 to list.length($heading_sizes) {
  ...
}
```

```
$heading-sizes: xs, s, m, l, xl, xxl;

@each $size in $heading_sizes {
  ...
}
```

Travailler avec une librairie SCSS

Si vous utilisez Bootstrap, sachez que cette librairie de composants est construite à l'aide de SCSS. Ainsi, si vous souhaitez la personnaliser, il va falloir connaitre les façons dont on peut récupérer des variables pour les changer.

En SCSS, les variables sont immutables, à une condition : si l'on ne les a pas déclaré avec le mot clé `!default`

Voyons un exemple :

```
$primary: red !default;
$secondary: blue !default;
```

Dans ce cas, en important ce module dans un autre module, lors de l'import, après le mot clé `@use`, on pourra utiliser le mot clé `with` pour changer les variables, comme ceci :

```
@use 'token' with (
  $primary: green,
  $secondary: aquamrine,
)
```

Conclusion

Il existe encore quelques autres règles que nous n'avons pas vu, mais à ce stade, vous devriez avoir une idée générale des possibilités de SASS, et notre but n'est pas de simplement faire un résumé de toute la documentation. Sachez tout de même que SCSS dispose d'une boucle while, que l'on peut exporter des mixins avec la directive @forward, etc.

Rappelez-vous avant tout que le but de votre CSS est d'être facile à utiliser et maintenable, n'abusez pas de la logique et essayez de ne pas rendre complexe le fait de trouver le nom de la classe que l'on est censé utiliser pour styliser un élément.