1. Saas (.scss) (pré-processeur css)

Tour d'horizon des préprocesseurs css

Les préprocesseurs CSS les plus couramment employés sont aujourd'hui Sass, LESS , PostCSS et Stylus.

pré-processeur CSS	principales caractéristiques
LESS	préprocesseur codé en javascript (utilisation dynamique "just in time" possible mais moins rapide et rarement envisagée)
Sass	préprocesseur initialement codé en ruby , maintenant des implémentations en plein de langage (C , java, ...) . Un peu plus puissant que "less" .
PostCSS	préprocesseur modulaire à base de plugins , plus récent que Saas , moins connu que Saas
Stylus	basé sur nodeJs ,inspiré de saas, pour l'instant pas beaucoup utilisé

La suite de ce chapitre se focalisera sur le préprocesseur Saas (pour l'instant le plus utilisé) .

Présentation de Saas

Saas = **S**yntactically **A**wesome **S**tyle**s**heet .

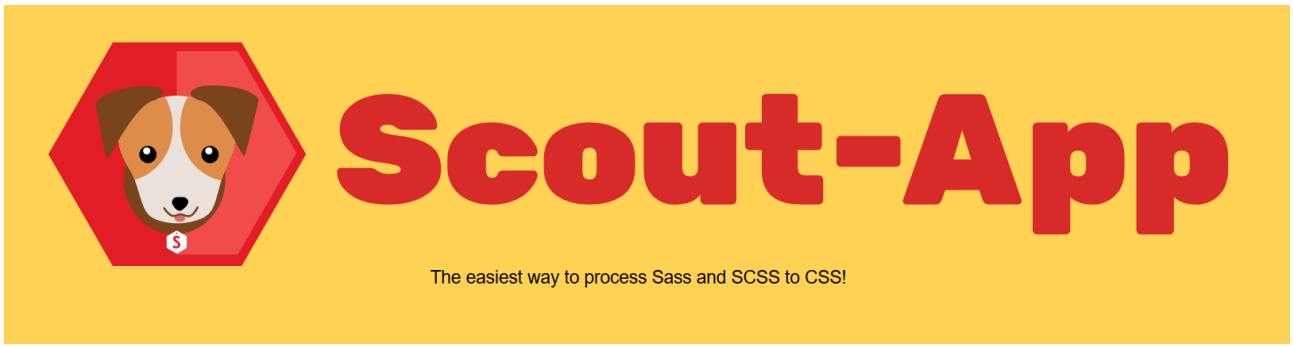
Les premières versions de Sass étaient très différentes de notre CSS; il n'y avait pas d'accolades, et les propriétés devaient être indentées avec un nombre précis d'espaces sous peine de recevoir un message d'erreur du compilateur.

La version 3.0 a introduit une nouvelle syntaxe, plus proche de CSS, appelée SCSS ("Sassy CSS" qu'on pourrait traduire par "CSS à la Sass" ou bien "CSS classieux"). SCSS peut maintenant être vu comme une extension de CSS, ce qui signifie que *tout ce qui est valide en CSS l'est aussi en SCSS*.

Principales fonctionnalités de sass/scss :

- utilisation de variables (couleurs , tailles, fontes, , ...)
- modularisation du "scss ==> css" (imbrication, héritage , mixin, ...)
- opérations simples pour agrandir, assombrir , éclaircir, ...
- boucles , évaluations de conditions, opérations personnalisées , ...

Installations possibles d'un processeur scss :



Télécharger et installer l'application "**Scout-app**" (ex: WIN_Scout-App_scss_2.18.16.zip) depuis le site https://scout-app.io/ .

ou bien , en mode global avec nodeJs :

```
npm install -g sass
```

NB : l'implémentation "nodeJs" (en javascript) n'est pas la plus rapide mais elle est portable (linux, windows, ...)

NB : Plein d'autres possibilités sont envisageables.

Bases de sccs

Fichiers avec extensions **.scss** . **Variables** préfixées par **$** (et par défaut globales).

Une variable peut éventuellement être définie et utilisée localement .

scss/***basic.scss***

```
$couleur1 : #ff00aa;
h2  { color: $couleur1 ; }
p {   $couleur1 : #eebb00;   color : $couleur1; }
```

```
sass  scss/basic.scss  out-css/basic.css
```

out-css/basic***.css***

```
h2 {  color: #ff00aa; }

p {  color: #eebb00; }
```

Il est possible de modifier la portée d'une variable en lui attribuant un suffixe !default ou !global .

import de sous fichiers "partial"

Un sous fichier "partial" a un nom commençant par un ***underscore*** "_" .

Lorsqu'un tel fichier sera traité par le préprocesseur sass , sont contenu sera recopié / inclus dans un fichier ou apparaîtra l'instruction @import sans reproduire le "@import dans le css généré .

Autrement dit un fichier _partielXy.scss n'est pas transformé en ~~_partielXy.css~~ .

Exemple :

_variables.scss

```
$couleur1 : #ff00aa;
$couleur2 : #eebb00;
```

basic.scss

```
@import "variables";
h2  { color: $couleur1 ; }
p {    color : $couleur2; }
```

--> ***basic.css***

```
h2 {  color: #ff00aa;}

p {  color: #eebb00; }
```

Imbrications de règles et pseudo sélecteur de parent &

Pour rendre le paramétrage des styles plus modulaires (avec moins de copier/coller) , on pourra imbriquer des règles les une dans les autres et au sein d'une règle imbriquée , le pseudo sélecteur & fera référence au sélecteur de la règle parente/englobante .

Exemple :

```
@import "variables";


h2  {
   color: $couleur1 ;
   &:hover {
            color: $couleur3 ;
        }
        }
p {
```

```
  color : $couleur2;
  &:hover {
            color: $couleur4 ;
        }
}
```

==>

```
h2 {
  color: #ff00aa;
}
h2:hover {
  color: #ff0000;
}

p {
  color: #eebb00;
}
p:hover {
  color: #00bb00;
}
```

Fonctions intégrées/prédéfinies

basic.scss

```
h3 { color: lighten($couleur1, 20%);  }
h4 { color: darken($couleur1, 20%); }
h5 { background-color: transparentize($couleur1, 0.5);}
```

==>

basic.css

```
h3 {
  color: #ff66cc;
}

h4 {
  color: #990066;
}

h5 {
  background-color: rgba(255, 0, 170, 0.5);
}
```

La liste des fonctions prédéfinies est accessible au bout de l'url suivante :

https://sass-lang.com/documentation/Sass/Script/Functions.html

principales fonctions intégrées :

fonctions intégrées	finalités/caractéristiques
lighten(baseColor, coeff)	éclaircir une couleur
darken(baseColor, coeff)	assombrir une couleur

@extend et %placeholder

L'instruction **@extend** (sans s) permet de récupérer (par pseudo-héritage) tous les attributs de mise en forme d'une autre règle considérée comme un simple "snippet" (morceau à réutiliser).

Lorsqu'un bloc d'attributs scss n'est utile que pour effectuer ultérieurement des héritages/copies on peut utiliser la syntaxe spéciale **%placeholderXyz** { ... } et ... { **@exend %placeholderXyz ; ... }**

Exemple :

basic.scss

```
...
%abstract-bold-underline{
  text-decoration: underline;
  font-weight : bold;
}
h1 { @extend %abstract-bold-underline; text-transform : uppercase;  }
h3 { @extend h1; color: $couleur1 ; }
h4 { @extend h1; color: $couleur2 ; }
h5 { @extend h1; color: $couleur3;}
```

==>

basic.css

```
h1, h5, h4, h3 {
  text-decoration: underline;
  font-weight: bold;
}

h1, h5, h4, h3 {
  text-transform: uppercase;
}

h3 {  color: #ff66cc;}
h4 {  color: #990066;}
...
```

Bien que techniquement utilisable entre éléments quelconques , le mot clef @extend devrait idéalement être utilisé entre éléments de sémantiques proches de manière à que la notion d'héritage soit intelligible (compréhensive , intuitive, ...) .

@mixin et @include

@mixin permet la définition d'un *snippet* réutilisable via **@include** .

un ***@mixin*** est assimilable à une fonction réutilisable avec la possibilité de passer des paramètres , d'effectuer des boucles , d'évaluer des conditions, ...

Exemple :

basic.scss

```
@mixin with-border($color){
 border: 2px solid $color;
}
```

==>

basic.css

```
h3 { @include with-border(blue); color: red;  }
```

interpolations , évaluations sophistiquées de variables

La syntaxe #{$variable} permet des juxtapositions et autres manipulations sophistiquées de variables scss .

Exemple :

```
$className: c1;
$attr: border;
p.$className {
 $attr-color: blue;
}
```

```
$className: c1;
$attr: border;
p.#{$className} {
  #{$attr}-color: blue;
}
```

==>

```
p.c1 {
  border-color: blue;
}
```

Boucles scss

```
@for $i from 1 through 3 {
  .spacer-#{$i} {
    margin: $i * 1rem;
  }
}
```

==>

```
.spacer-1 {
  margin: 1rem;
}

.spacer-2 {
  margin: 2rem;
}

.spacer-3 {
  margin: 3rem;
}
```

Evaluation de condition scss

```
@mixin with-rotation($rotation) {
  @if $rotation == 0 {
  }
  @else {
    transform: rotate($rotation);
  }
}


.avecBordureEtRotation {
  width:300px;
  @include with-rotation(25deg);
  @include with-border(blue);
}
```

==>

```
.avecBordureEtRotation {
  width: 300px;
  transform: rotate(25deg);
  ...
}
```

Bonnes pratiques et organisations "scss"

Pour projets simples/ordinaires :

_base.scss *ou bien* **_variables.scss** + **_mixins.scss** (variables , mixins , reset or normalize)

_layout.scss (container , grid , layout , ...)

_component.scss (navbar , card , ...)

main.scss *ou bien* **styles.scss** (@import , ...)

Quelques idées (en vrac) :

- **_reset.scss** ou **_normalize.scss** (valeurs par défaut , moins de différences entre les navigateurs, ...)
- utiliser (ou bien s'inspirer de) certains "framework css" tels que "**blueprint** css" ou "**compass** css" .
- Utiliser une des librairies de fichiers .scss prédéfinis

bourbon (https://www.bourbon.io/)	très complet
https://github.com/matthieua/Sass-css3-mixins	simple
https://cssowl.owl-stars.com/	
https://davidtheclark.github.io/scut/color-swap.html	
http://breakpoint-sass.com/	
http://colindresj.github.io/saffron/	
https://sassline.com/	
http://gillesbertaux.com/andy/	

- ...

Responsive scss

_bases.scss

```
$my-media-breakpoints-bootstrap-css: (
   "sm": 576px,
   "md": 768px,
   "lg": 992px,
   "xl": 1200px
);
$my-media-breakpoints : $my-media-breakpoints-bootstrap-css;
//coeffs for font-size of text :
$my-media-coeffs: (   "sm": 100%,   "md": 110%,   "lg": 120%,    "xl": 130% );

@mixin media-min($_key) {
  @if map-has-key($my-media-breakpoints, $_key) {
                @media screen and (min-width: map-get($my-media-breakpoints, $_key)) {
                        &{ @content; }
                }
        } @else {
   // Log a warning.
   @warn 'Invalid breakpoint key: #{$_key}.';
  }
}

@mixin media-max($_key) {
   @media screen and (max-width: map-get($my-media-breakpoints, $_key) - 1px) {
      &{ @content; }
   }
}

@mixin media-between($_keymin, $_keymax) {
   @media screen and (min-width: map-get($my-media-breakpoints, $_keymin)) and
      (max-width: map-get($my-media-breakpoints, $_keymax) - 1px) {
      &{ @content; }
   }
}
```

```
/* alternative names for media-query mixins :
media-min or respond-below or media-breakpoint-down
, media-max or respond-above ,  media-between or respond-between
*/

%my-responsive-text {
   font-size:80%;
   //boucle possible sur my-media-breakpoints :
   @each $_key, $_value in $my-media-breakpoints {
      @include media-min($_key) {
        font-size: map-get($my-media-coeffs, $_key);
      }
   }
}
```

main-styles.scss

```
@import "_bases";

#contentXx {    width: 100%;
   @include media-min('md') {
      font-size:100%; color:blue;
   }
   @include media-min('lg') {
      font-size:110%; color:red;
   }
}

#contentYy {    width: 100%;
   @include media-max('md') {
      font-size:100%; color:blue;
   }
   @include media-between('lg','xl') {
      font-size:110%; color:red;
   }
}
#contentZz {  @extend %my-responsive-text; }
```

==>

```
out-css/main-styles.css
#contentZz {  font-size: 80%;}
@media screen and (min-width: 576px) {
  #contentZz {    font-size: 100%;  }
}
@media screen and (min-width: 768px) {
  #contentZz {    font-size: 110%;  }
}
@media screen and (min-width: 992px) {
  #contentZz {    font-size: 120%;  }
}
@media screen and (min-width: 1200px) {
  #contentZz {    font-size: 130%;  }
}

#contentXx {  width: 100%;}
@media screen and (min-width: 768px) {
  #contentXx {    font-size: 100%;    color: blue;  }
}

#contentYy {  width: 100%;}
@media screen and (max-width: 767px) {
  #contentYy {    font-size: 100%;    color: blue;  }
}
@media screen and (min-width: 992px) and (max-width: 1199px) {
  #contentYy {    font-size: 110%;    color: red;  }
}
```