sass/scss

- pré-processeur proposant des extensions à css pour :
  - améliorer le découplage html-css
  - programmer des scripts de génération de code css
- disponible en 2 syntaxes :
  - sass : syntaxe ruby-like
  - scss : syntaxe css-like
- source : http://sass-lang.com
- lecture recommandée : https://sass-lang.com/guide

installer sass

- https://sass-lang.com/install
- une installation en ligne de commande est fortement recommandée
  - plus flexible
  - permet une intégration dans votre IDE

- installer une application en plus peut être utile

principes de base scss

- scss : **sur-ensemble** de css
  - tout code css valide est aussi du scss valide

```
#navbar {
  width: 80%;
  height: 23px;
}
```

principes de base

- compilation : sass traite des fichiers sources `.sass` ou `.scss` pour les transformer en code css
  - chaque fichier source est traduit en un fichier css de même nom : `style.scss → style.css`
  - le compilateur peut traiter l'ensemble des fichiers d'un répertoire
  - le compilateur peut travailler en mode "watch" : il surveille un fichier ou un répertoire et refait la traduction automatiquement à chaque changement

```
$>sass scss/style.scss:css/style.css
$>sass –watch scss/style.scss:css/style.css
$>sass –watch scss:css
```

import de fichiers "partial"

- @import : permet d'importer un "partial" (fichier scss) dans un autre fichier
- intérêt : découper et réutiliser du côté du serveur
- 1 seul fichier produit et transféré au client

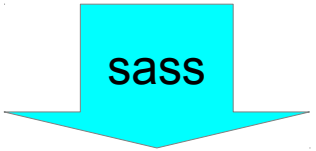
- **A noter** : un fichier dont le nom débute par `_` peut être importé mais aucun fichier résultat n'est produit pour lui
  - `_reset.scss` : peut être importé, mais le fichier `reset.css` n'est pas produit
  - on désigne le fichier importé par son chemin relatif sans extension ni _

```scss
html, body, div, span, ul {
  margin: 0;
  padding: 0;
  border: 0;
  font-size: 100%;
}
ol, ul {
  list-style: none;
}
```

_reset.scss

```scss
@import 'reset';

body {
  font: 100% Helvetica, sans-serif;
  background-color: #efefef;
}
```

main.scss




```css
html, body, div, span, ul {
  margin: 0;
  padding: 0;
  border: 0;
  font-size: 100%;
}

ol, ul {
  list-style: none;
}

body {
  font: 100% Helvetica, sans-serif;
  background-color: #efefef;
}
```

main.css

Variables

- **variables** : à utiliser sans modération

scss

```
$bcolor: #fafafa ;
$fcolor: #020202 ;
nav {
    background-color: $bcolor ;
    color: $fcolor ;
}
```

sass

css

```
nav {
    background-color:#fafafa;
    color: #020202 ;
}
```

- valeurs par défaut : affectation seulement si la variable n'a pas déjà une valeur définie par ailleurs

```
$bcolor: #fafafa !default;
$fcolor: #020202 !default;
```

expressions

- permet d'utiliser des expressions pour calculer des valeurs
- fournit une bibliothèque de fonctions prédéfinies

scss

```
$items: 5  ;
$bcolor: rgba(#f6f6f6,1);

article {
    float: left ;
    margin: 0 1% ;
    width: (100%/$items) - 2% ;
    background-color: $bcolor ;
    color: darken($bcolor,50) ;
}
```

css

```
article {
  float: left ;
  margin: 0 1% ;
  width: 18% ;
  background-color: #f6f6f6 ;
  color: #767676 ;
}
```

imbrication de sélecteurs

- imbrication des sélecteurs

scss

```
.tside {
   width: 23em  ;
  padding: 1em ;
   p {
     font-size:0.9em ; }
}
```

sass

css

```
.tside {
   width: 23em  ;
  padding: 1em ;
}
.tside p {
    font-size:0.9em ;
}
```

- référence au parent :

scss

```
.nav {
  display : flex ;
  &>li {
    color: red ;
    &:hover {
      color: maroon ;
     }
   }
}
```

sass

css

```
.nav {
   display: flex;
}

.nav>li {
    color: red ;
}
.nav>li:hover {
    color: maroon;
}
```

utilisation de l'imbrication

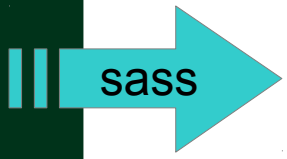
- **intérêt :** calquer les règles css sur l'arborescence html
- **attention** : ne pas abuser de l'imbrication car conduit à des sélecteurs avec des expressions de chemins très longues
  - problèmes de **performance** pour le navigateur
  - rend le code **css très dépendant de l'arbre html**
- à utiliser pour définir des "widgets" html/css réutilisables

extension

- @extend : réutilisation et extension d'une règle existante

```scss
.message {
  padding: 10px ;
  border: 2px solid #ccc ;
  color: #333 ;
}

.mess-success {
  @extend .message;
  border-color: green ;
}

.mess-error {
  @extend .message;
  border-color: red ;
}
```

scss




```css
.message,
.mess-success,
.mess-error {
  padding: 10px ;
  border: 2px solid #cccccc
 ;
  color: #333333  ;
}


.mess-success {
  border-color: green ;
}

.mess-error {
  border-color: red ;
}
```

css

- **Attention** : non utilisable à l'intérieur d'un @media

extension avec *placeholder*

- un ***placeholder*** : comme une classe, mais n'apparaitra pas dans le css final, et donc destiné à être uniquement utilisé avec @extend

```
%message {
  padding: 10px ;
  border: 2px solid #ccc ;
  color: #333 ;
}

.mess-success {
  @extend %message;
  border-color: green ;
}

.mess-error {
  @extend %message;
  border-color: red ;
}
```

sass

```
.mess-success,.mess-error{
  padding: 10px ;
  border: 2px solid #ccc;
  color: #333333  ;
}
.mess-success {
  border-color: green ;
}
.mess-error {
  border-color: red ;
}
```

mixins

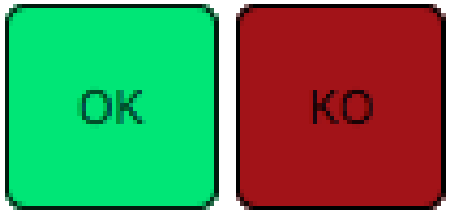
- construction sass permettant de définir des ensembles réutilisables et nommés de règles `propriétés:valeurs` css dépendant de un ou plusieurs arguments,
  - `@mixin` *`nom`* `( paramètres ) { <règles> }` : permet de définir un mixin en associant un nom à des règles paramétrées
  - `@include nom( args )` : utilisation d'un mixin, le compilateur remplace le nom du mixin par les règles le définissant

```scss
@mixin boite($width, $height :$width) {
  display: inline-block;
  width: $width;
  height: $height;
  background-color: magenta;
}

p {
  @include boite(25em,10em);
  color: blue;
}
.carre {
  @include boite(30em);
  color: goldenrod;
}
```

sass

```css
p {
 display: inline-block;
 width: 25em;
 height: 10em;
 background-color: magenta;
 color: blue;
}

.carre {
 display: inline-block;
 width: 30em;
 height: 30em;
 background-color: magenta;
 color: goldenrod;
}
```

exemple : boutons paramétrés




```
<body>

<button class="ok">
OK
</button>

<button class="ko">
KO
</button>


</body>
```

html propre et sémantique

```
@import "button";

button.ok {
  @include mybutton( #00E676);

}

button.ko {
  @include mybutton(#a21318);
}
```

style scss concis, flexible et réutilisable

```scss
/* _button.scss */

@mixin mybutton( $color) {

  display: inline-block;
  text-align: center;
  padding: 1rem;
  background-color: $color;
  color: darken($color, 30%);
  border: 1px solid darken($color, 40%);
  border-radius: 10%;
  &:hover {
    background-color: lighten($color, 30%);
  }

}
```

mixins ou @extends ?

- **@extends** : classes ou placeholder
  - pas de paramètres et non utilisables dans 1 media-query
  - code ***factorisé***
- **@mixin + @include**
  - paramètres
  - code ***duppliqué***
- **plutôt des placeholder, sauf si :**
  - la classe étendue existe déjà et est utilisée dans le code html : utiliser une classe
  - on doit avoir un code réutilisable et paramétré : mixins
  - on l'utilise dans un @media : mixins

sassScript

- sass propose un langage de script
  - structures de contrôle : @if, @for, @while, @each
  - fonctions : @function, @return
  - variables

- utilisation :
  - programmation de mixins complexes
  - génération de code css

variables

- les variables peuvent être utilisées :
- dans une expression
- comme valeur d'une propriété

```
$fcolor: #00b7f5;
$items: 5;
article {
  color: $fcolor ;
  width: 100% / $items;
}
```

- dans un sélecteur ou dans le nom d'une propriété avec la syntaxe #{ … }

```
$name: foo;
$attr: border;

p.#{$name} {
  #{$attr}-color: blue;
}
```

structures de contrôle

- @if : dans un sélecteur
- classes générées avec for :

```
p {
  @if $n > 15 {
    witdh: $n * 10%;
  } @else {
    width: 30%;
  }
}
```

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

- avec each :

```
@each $animal in chat, chien, lapin, salamandre {
  .#{$animal}-icon {
    background-image: url('/images/#{$animal}.png');
  }
}
```