Sass

Présentation

Sass - **Syntactically Awesome Style Sheets** - est un préprocesseur CSS, c'est à dire un compilateur permettant de générer des fichiers CSS à partir d'un fichier écrit dans une syntaxe différente du CSS pur

Les avantages d'utiliser un préprocesseur CSS

- plus grande maintenabilité et évolutivité du CSS
- possibilité d'utiliser des concepts liés aux langages de programmation (variables, boucles, conditions...)

Sass est le principal membre de la famille des préprocesseurs CSS

- LESS
- Stylus

Sass peut être utilisé avec deux syntaxes qui ne peuvent pas cohabiter dans le même fichier

- syntaxe avec extension **.sass**
    - ancienne syntaxe mais encore fonctionnelle
    - les tabulations remplacent les accolades
    - les retours à la ligne remplacent les points-virgules
- syntaxe avec extension **.scss**

    - syntaxe recommandée

SassMeister est une application en ligne permettant de

- tester les différentes syntaxes disponibles dans le menu **Options / Syntax**
- ajouter des extensions disponibles dans le menu **Options / Extensions**
- visualiser les différents formats de sortie du CSS disponibles dans le menu **Options / CSS Output**

Installation

Le compilateur Sass peut être installé via

- une extension d'un éditeur de texte
    - Visual Studio Code
    - Atom
- npm et Node.js

Lors de l'utilisation du compilateur, l'architecture du dossier de travail sera supposée être

```
+-- css
|       +-- main.min.css
+-- scss
|       +-- main.scss
+-- index.html
```

Utilisation de Visual Studio Code

Pour installer le compilateur Sass avec Visual Studio Code, il est conseillé d'installer l'extension Live Sass Compiler

Après l'installation

- aller dans le menu **Fichier / Préférences / Paramètres**
- ajouter la propriété

```
"liveSassCompile.settings.formats": [{
        "format": "compressed",
        "extensionName": ".min.css",
        "savePath": "~/../css/"
}]
```

Pour lancer le compilateur, ouvrir un fichier Sass, puis cliquer sur le lien **Watch my Sass** situé dans la barre de statut en bas du logiciel

Installation de npm et Node.js

Sur Windows

- télécharger l'installeur : https://nodejs.org/en/download/current/
- après l'installation : saisir **powershell** dans le champ de recherche du bouton **démarrer**
- vérifier l'installation

```
node -v ;; npm -v
```

Sur MacOS

- télécharger l'installeur : https://nodejs.org/en/download/current/
- après l'installation : utiliser le **Terminal**
- vérifier l'installation

```
node -v && npm -v
```

Sur Linux

- installation des paquets initiaux : https://nodejs.org/en/download/package-manager/

```
sudo apt-get update && sudo apt-get upgrade -y
sudo apt-get install curl -y
```

- installer Node.js

```
cd ~
curl -sL https://deb.nodesource.com/setup_<VERSION_PRINCIPALE>.x -o
nodesource_setup.sh
sudo bash nodesource_setup.sh
sudo apt-get install nodejs -y && sudo apt-get install npm -y
```

> Veiller à modifier la version à télécharger

- vérifier l'installation

```
node -v && npm -v
```

Installation du module node-sass

Le module **Node.js** node-sass permet d'installer un compilateur Sass

```
npm install -g node-sass@latest
```

Vérifier l'installation

```
node-sass -v
```

Si des droits sont nécessaires pour installer le module

- installer le module en outrepassant les permissions

```
sudo npm install -g --unsafe-perm node-sass@latest
```

Si des problèmes persistent

- se placer sur le bureau

```
cd ~/Desktop
```

- installer le compilateur sur le bureau

```
npm install node-sass@latest
```

- vérifier l'installation

```
~/Desktop/node_modules/node-sass/bin/node-sass -v
```

Utilisation du module node-sass

Ligne de commande la plus fréquemment utilisée

```
cd <WORKING_DIR>
node-sass --watch --recursive --output-style=compressed --source-map=true
scss/main.scss css/main.min.css
```

- l'option **--watch** ou **-w** permet d'indiquer que la compilation est exécutée dès qu'un fichier est modifié

  > il est impératif de laisser le terminal/powershell ouvert pour que cette option soit active

- l'option **--recursive** ou **-r** permet d'indiquer que la compilation doit également s'exécuter dès qu'un fichier situé dans un sous-dossier est modifié
- le format de sortie est défini sur **compressé**
- un fichier **source-map** est généré, permettant aux outils de développement de relier le fichier compilé aux fichiers sources

Si des droits ont été nécessaires pour installer le module et qu'il est présent sur le bureau

```
cd <WORKING_DIR>
~/Desktop/node_modules/node-sass/bin/node-sass --watch --recursive --output-
style=compressed --source-map=true scss/main.scss css/main.min.css
```

Fichiers partiels et importation

Si une méthode de structuration des CSS est utilisée, le nombre de fichiers CSS peut devenir conséquent

La multiplication des fichiers CSS est

- fastidieux lors de l'importation dans le HTML
- peu performant pour le chargement d'une page Web

L'importation de fichiers partiels - ou **partials** - permet de conserver différents SCSS mais de les importer et de les compiler dans un fichier CSS unique

L'architecture du dossier de travail recommandée

```
+-- scss
|       +-- assets
|       |       +-- normalize.css
|       +-- layout
|       |       +-- _default.scss
|       |       +-- …
|       +-- modules
|       |       +-- _button.scss
|       |       +-- …
|       +-- themes
|       |       +-- _default.scss
|       |       +-- _button.scss
|       |       +-- …
|       +-- _base.scss
|       +-- _reset.scss
|       +-- _states.scss
|       +-- main.scss
```

Le fichier `main.scss` sert de point d'entrée unique pour l'importation de tous les fichiers partiels

Pour importer un fichier

- un fichier partiel doit être préfixé d'un tiret bas
- utiliser la règle `@import` sans préciser l'extension, dans le fichier `main.scss`

```
@import "reset";
@import "assets/normalize";
@import "base";

@import "layout/default",
        "modules/button",
        "states",
        "themes/default"
;
```

- si un fichier CSS n'est pas préfixé d'un tiret bas et ne contient pas d'extension, il sera importé et compilé dans le fichier principal

```
@import "reset",
        "assets/normalize",
        …
;
```

- si un fichier CSS n'est pas préfixé d'un tiret bas et contient une extension, il sera importé dans le fichier principal mais non compilé; le fichier importé doit être alors présent dans le dossier **css**

```
@import "reset",
        "assets/normalize.css",
        …;
```

> L'ordre d'importation des fichiers partiels conserve son importance

Lors de la compilation de fichiers partiels, les commentaires sont supprimés

Pour conserver certains commentaires, utiliser un point d'exclamation après l'ouverture du commentaire

```
/*! comment */
```

Variables

Présentation

Comme dans tout langage de programmation, les variables

- permettent de stocker une information réutilisable dans le code
- possèdent un type
- possèdent une portée

Une variable est composée

- d'un nom précédé du caractère **$** et suivi du caractère **:**
- d'une valeur

```
$variable: value;
```

Il est recommandé de créer un fichier partiel stockant toutes les variables

Types simples de variables

Comme dans tout langage de programmation, les variables possèdent un type

- chaîne de caractères ou **string**
- nombre ou **number**
- booléen ou **boolean**
- vide ou **null**

La valeur d'une variable détermine son type

```
$color: tomato;
$string: "https://mywebsite.com/logo.svg";
$number: 5;
$value: 1rem;
$bool: true;
$empty: null;

body{
        background: $color url($string);
        font-size: $number + rem;
        padding: $value * $number;
        @if $bool {
                text-transform: uppercase;
        }
}
```

Selon le type de la variable, différentes opérations sont possibles

- avec une chaîne de caractères : concaténation à l'aide du caractère **+**
- avec un chiffre : opérations arithmétiques
- avec un booléen : opérations booléennes **and**, **or** et **not**

Débogage

Le terminal/powershell permet de deboguer à l'aide de différentes instructions

- la règle `@debug` permet d'afficher un message
- la règle `@warn` permet d'afficher un message d'avertissement
- la règle `@error` permet d'afficher un message d'erreur

```
$variable: 1;

@debug $variable;
@warn $variable;
@error $variable;
```

Interpolation de variables

La valeur d'une variable peut être interpolée à l'aide d'un dièse et d'accolades

```
$color: tomato;
$food: salmon;
$website: "https://mywebsite.com/";

/* I love #{$food} */
body{
        background: $color url(#{$website}logo.svg);
        background: $color url("#{$website}logo.svg");
        background: $color url($website + "logo.svg");
}
```

Portée des variables

Une variable est **globale** si elle est définie hors d'un sélecteur CSS

- la variable est accessible par tous les sélecteurs CSS

Une variable est **locale** si elle est définie dans un sélecteur CSS

- la variable est uniquement accessible dans le sélecteur CSS où elle a été définie
- en cas d'homonymie, la variable locale est prioritaire sur la variable globale
- l'ordre de déclaration est important

```
$color: tomato;

body{
        $color: olive;
        border: 1rem solid $color;
}
```

Le mot clé `!default` permet de conserver la valeur précédente d'une variable tant que sa valeur n'a pas été modifiée ou redéfinie avec `null`

```
$variable: 1;
$variable: 2 !default;
@debug $variable;

$variable: null;
$variable: 2;
@debug $variable;
```

Le mot clé `!global` permet d'attribuer la valeur d'une variable locale à une variable globale

```
$color: tomato;

body{
        $color: olive !global;
}

@debug $color;
```

Types complexes de variables

Comme dans tout langage de programmation, une variable peut possèder plusieurs valeurs

- type **list** : les valeurs sont accessibles par un indice numérique débutant à 1
- type **map** : les valeurs sont accessibles par une clé

Dans les langages de programmation

- le type **list** équivaut à un tableau ou **array**
- le type **map** équivaut à un objet ou **object**

List

Pour créer une `list`

```
```

```
$list: (36, 24, 21, 18, 16, 14);
$list: 36 24 21 18 16 14;
```

Pour parcourir une `list`

```
@for $i from 1 through length($list) {
        h#{$i} {
                font-size: nth($list, $i) +px;
        }
}

@for $i from 1 to length($list) + 1 {
        h#{$i} {
                font-size: #{nth($list, $i) / 16}rem;
        }
}
```

- la boucle `for` permet d'accéder à chaque élément de la `list`
- la variable `i` représente l'indice numérique
- le mot clé `through` équivaut à **inférieur ou égal**
- le mot clé `to` équivaut à **inférieur**
- la fonction `length` permet de récupérer le nombre d'éléments présents dans la `list`
- la fonction `nth` permet de récupérer un indice particulier dans la `list`

Map

Pour créer une `map`

```
$map: (
        h1: 36,
        h2: 24,
        h3: 21,
        h4: 18,
        h5: 16,
        h6: 14
);
```

Pour parcourir une `map`

```
@each $key, $value in $map {
        #{$key}{
                font-size: #{$value}px;
```

```
        }
}

p{
        font-size: #{map-get($map, h6) / 16}rem;
}
```

- la boucle `each` permet d'accéder à la clé et à la valeur de chaque élément de la `map`
- la variable `key` représente la clé
- la variable `value` représente la valeur
- la fonction `map-get` permet de récupérer une clé particulière dans la `map`

Imbrication

Imbrication de sélecteurs CSS

L'imbrication permet de factoriser l'écriture des sélecteurs CSS

Si l'élément enfant est un sélecteur CSS, les élément seront chaînés

```
 #parent{
        background: gold;
        .child{
                background: teal;
        }
}
```

Le sélecteur d'enfant direct peut également être utilisé

```
 #parent{
        background: gold;
        &gt; .child{
                background: teal;
        }
}
```

Si l'élément enfant n'est pas un sélecteur CSS, les élément seront concacténés à l'aide du caractère **&**

```
.module{
        background: lavender;
        &-title{
                background: chocolate;
        }
}
```

Si le caractère **&** est utilisé après un sélecteur CSS, il fera référence au sélecteur parent

```
 #parent{
        background: gold;
        &:hover .child, &:focus{
                background: teal;
        }
}
```

> L'imbrication est à utiliser avec parcimonie, il est fortement recommandé de ne pas dépasser 2/3 niveaux d'imbrication pour réduire la priorité des sélecteurs

Imbrication de propriétés CSS

Tout comme les sélecteurs, les propriétés CSS peuvent être imbriquées

```
body{
        background: {
                image: url(https://mywebsite.com/);
                repeat: no-repeat;
                size: cover;
                position: center top;
        }
}
```

Imbrication de requêtes de média

Les requêtes de média peuvent être imbriquées dans leurs sélecteurs CSS

```
body{
        background: gold;
        @media only screen and (max-width: 60rem){
                background: indianred;
        }
}
```

Externaliser une imbrication à la compilation

Il est parfois obligatoire d'écrire du code CSS en dehors de tout sélecteur; principalement lorsque des règles CSS sont utilisées

La règle `@at-root` permet de regrouper des propriétés propres à un sélecteur lors de l'écriture du SCSS,mais ces propriétés seront externalisées vers la racine du code CSS, lors de la compilation

```
body{
        background: olive;
        animation: anim 1s infinite alternate;

        @at-root @keyframes anim{
                0%{ background: olive; }
                100%{ background: chocolate; }
        }
}
```

Regroupement

Présentation

Le regroupement de propriétés CSS - ou **mixins** - permet de créer un ensemble de propriétés CSS réutilisables

Les regroupements permettent de factoriser l'écriture des sélecteurs CSS en réutilisant des groupes de propriétés CSS récurrentes et de les inclure au besoin dans tout autre sélecteur CSS

Ces regroupements se rapprochent des fonctions présentes dans tout langage de programmation car il est possible d'utiliser des paramètres

Il est recommandé de créer un fichier partiel stockant tous les regroupements

Utilisation sans paramètres

Pour créer un regroupement

```
@mixin color{
        background: gold;
        color: pink;
}

body{
        @include color;
}
```

- la règle `@mixin` permet de créer le regroupement et de le nommer
- la règle `@include` permet d'inclure un regroupement à l'aide de son nom

Les regroupements peuvent s'imbriquer

```
@mixin color{
        background: gold;
        color: pink;
        @include font;
```

```
}

@mixin font{
        font-family: arial;
}
```

Pour ajouter des propriétés CSS à un regroupement imbriqué

```
@mixin color{
        background: gold;
        color: pink;
        @include font{
                font-weight: bold;
                font-style: italic;
        };
}

@mixin font{
        font-family: arial;
        @content;
}
```

- la règle `@content` permet de retourner l'ensemble du contenu du regroupement; à utiliser uniquement dans un regroupement

Utilisation avec paramètres

Pour créer un paramètre ne possèdant pas une valeur par défaut

```
@mixin color($theme){
        @if $theme == "summer"{
                background: gold;
        } @else {
                background: steelblue;
        }
}

body{
        @include color("summer");
}
```

Pour créer un paramètre possèdant une valeur par défaut

```
@mixin color($theme: "winter"){
        @if $theme == "summer"{
                background: gold;
        } @else {
```

```
                background: steelblue;
        }
}

body{
        @include color;
}
```

Les variables globales sont accessibles par un regroupement

```
$theme: "summer";

@mixin color{
        @if $theme == "summer"{
                background: gold;
        } @else {
                background: steelblue;
        }
}

body{
        @include color;
}
```

Fonctions prédéfinies et personnalisées

Fonctions prédéfinies

Fonctions prédéfinies

Comme dans tout langage de programmation, Sass proposent des fonctions prédéfinies

Ces fonctions sont regroupées en catégories : couleur, nombre, chaîne de caractères, `list`, `map`...

La documentation officielle se situe à l'adresse : http://sass-lang.com/documentation/Sass/Script/Functions.html

Fonctions liées aux couleurs

Un nombre conséquent de fonctions permettent de manipuler les couleurs

- `lighten($color, $percent)` permet d'éclaircir une couleur
- `darken($color, $percent)` permet d'obscurcir une couleur
- `saturate($color, $percent)` permet de saturer une couleur
- `desaturate($color, $percent)` permet de désaturer une couleur
- `mix($color1, $color2)` permet de mélanger deux couleurs
- `grayscale($color)` permet de convertir une couleur en niveaux de gris

- `complement($color)` permet d'obtenir la complémentaire d'une couleur
- ...

```
$color: tomato;

p{
        color: $color;
        &:hover{
                color: darken($color, 30%);
        }
}
```

Un service en ligne permet de visualiser le rendu des fonctions à partir d'une couleur : http://jackiebalzer.com/color

Fonctions liées aux nombres

Certaines fonctions permettent de manipuler les nombres

- `abs($number)` permet d'obtenir la valeur absolue d'un chiffre
- `ceil($number)` permet d'obtenir la valeur par excès d'un chiffre
- `floor($number)` permet d'obtenir la valeur par défaut d'un chiffre
- `round($number)` permet d'obtenir l'arrondi d'un chiffre
- `random($limit)` permet d'obtenir une nombre aléatoire
    - **décimal**, entre 0 et 1, si un chiffre n'est pas défini en paramètre
    - **entier** si un chiffre est défini en paramètre
- ...

Fonctions liées aux chaînes de caractères

Certaines fonctions permettent de manipuler les chaînes de caractères

- `unquote($string)` permet de supprimer les guillemets d'une chaîne de caractères
- `quote($string)` permet d'ajouter des guillemets à une chaîne de caractères
- `str-length($string)` permet de récupérer la longueur d'une chaîne de caractères
- ...

Fonctions liées aux list et map

Certaines fonctions permettent de manipuler les `list`

- `length($list)` permet de récupérer la longueur d'une `list`
- `nth($list, $index)` permet de récupérer un indice particulier dans une `list`
- `max($list)` permet d'obtenir la valeur maximale contenue dans une `list`
- `min($list)` permet d'obtenir la valeur minimale contenue dans une `list`
- `append($list, $value)` permet d'ajouter une entrée dans une `list`
- ...

Certaines fonctions permettent de manipuler les `map`

- `map-get($map, $key)` permet de récupérer une clé particulière dans une `map`
- `map-keys($map)` permet de récupérer toutes les clés d'une `map`
- `map-values($map)` permet de récupérer toutes les valeurs d'une `map`
- ...

Fonctions personnalisées

Une fonction personnalisée

- se crée avec la règle `@function`
- ne contient pas de propriétés CSS comme un regroupement; sert uniquement à **retourner une valeur**
- doit utiliser la règle `@return` pour retourner une valeur
- accepte des arguments
- accède aux variables globales

```
$hexa: (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f);

@function random-hexa(){
        $i: 0;
        $result: "#";
        @while $i < 6 {
                $random: round(random(length($hexa)));
                $result: $result + nth($hexa, $random);
                $i: $i + 1;
        }
        @return unquote($result);
}

body{
        background: random-hexa();
}
```