Développement WEB

Partie : Frontend

Les préprocesseurs CSS

Prof. M. LAROUSSI

Qu'est ce qu'un préprocesseur CSS

Les feuilles de styles deviennent rapidement surchargées et illisibles et surtout difficilement maintenables.

Comment y remédier ?

La solution la plus utilisée est **le préprocesseur CSS**.

En informatique, un préprocesseur **est un programme qui procède à des transformations sur un code source**, avant l'étape de traduction proprement dite (compilation ou interprétation).

Un préprocesseur est un programme jouant le rôle d'une moulinette : on lui donne du code source et, en échange, il génère un code modifié.

Dans le cas d'un préprocesseur CSS comme **Sass**, on donne des fichiers écrits dans un langage spécifique à **Sass** et la moulinette génère des feuilles de style CSS qui pourront être comprises par un navigateur. **C'est la compilation**.

Qu'est ce qu'un préprocesseur CSS

Un préprocesseur CSS est un « programme » ou module sous Node.js (pour Less et Sass) qui interprète votre code source pour générer un code standard du web : le CSS.

Ce sont des générateurs dynamiques de feuilles de style CSS.

LESS, **Stylus** et **SASS** (**S**yntactically **A**wesome **S**tyle**s**heet) sont les préprocesseurs CSS les plus utilisés, ils permettent d'améliorer la syntaxe du langage en préservant :

- Ses fondamentaux,
- Ses possibilités,
- Sa conformité W3C.

- Sass est un préprocesseur CSS, c'est-à-dire qu'il génère du CSS à partir de ***fichiers.scss***
- Utiliser un préprocesseur CSS permet d'optimiser son code et de moins se répéter
- La syntaxe par défaut de Sass est le SCSS et tout code CSS est compatible avec cette syntaxe
- La commande sass --watch input:output indique à Sass qu'il doit recompiler automatiquement les fichiers .scss du dossier input à chaque modification et placer les fichiers CSS générés dans le dossier output.

Qu'est ce qu'un préprocesseur CSS

Pourquoi le SASS ?

- **Sass** ajoute aux CSS un ensemble de fonctionnalités qui permettent d'organiser de manière plus maintenable les feuilles de style.

- Améliore le découplage HTML CSS.

- Sa devise pourrait être "Don't Repeat Yourself" (ne vous répétez pas).

- Il ne s'agit donc pas de rajouter des propriétés, mais d'aider le développeur à y voir plus clair, à défricher son code (d'où la métaphore de la forêt vierge, CQFD).

Dans ce but, Sass permet de factoriser certains bouts de code, stocker les couleurs, polices, dimensions fréquemment utilisées en un unique endroit

rendre certaines règles plus facilement réutilisables d'un projet à l'autre.

Le SASS est disponible en deux syntaxes :

- **Des fichiers avec extension (.sass) → Syntaxe identité**

  Les fichiers ayant l'extension ***.sass*** permettent d'écrire du CSS sans se soucier des points-virgules et des accolades : l'indentation suffit à déterminer quelles règles s'appliquent à un sélecteur donné.

- **Des fichiers avec extension (.scss)**

  Les fichiers ayant l'extension .scss permettent d'écrire du CSS avec les règles habituelles de celui-ci : les déclarations sont séparées par un point-virgule et chaque set de règle est écrit entre accolades.

Différence entre SASS et SCSS?

- La principale différence réside dans la syntaxe. SCSS utilise une syntaxe proche de celle de CSS, tandis que Sass (lorsqu'il est utilisé seul) utilise une syntaxe avec indentation significative.
- La syntaxe SCSS est souvent préférée par les développeurs familiarisés avec CSS, car elle ressemble davantage à la syntaxe standard du langage.
- Les deux syntaxes, SCSS et Sass, peuvent coexister dans un même projet, permettant aux développeurs de choisir celle qui correspond le mieux à leurs préférences et à leur style de codage.

Les navigateurs ne comprennent pas le code Sass.

Pour utiliser Sass, nous allons devoir utiliser un compilateur dont le rôle va être de traduire les fichiers .sass ou .scss (fichiers Sass) en fichiers .css (fichiers CSS classiques) compréhensibles par le navigateur.

Pourquoi le SASS ?

***.sass**

```
.div
        margin: 10px
        font-size: 14px
        border-radius: 5px
        border: 1px solid black
```

***.scss**

```
.div {
        margin: 10px;
        font-size: 14px;
        border-radius: 5px;
        border: 1px solid black;
}
```

SASS, SCSS, CSS

SASS

```
.title
    padding:10px
    background-color:black
    h1
        @extend .title
        font-size: 1em
    p
        @extend .title
        color: red
```

CSS

```
.title {
    padding:10px;
    background-color:black;
}
.title h1 {
    font-size:1em;
}
.title p {
    color:red;
}
```

SCSS

```
.title {
    padding:10px;
    background-color:black;
    h1 {
        font-size:1em;
    }
    p {
        color:red;
    }
}
```

Installation Sass

- On peut l'installer en ligne de commande ou en mode application.
- Il est recommandé de l'installer en mode console (avec npm) pour deux raisons :
  - Meilleure flexibilité
  - Intégration automatique à l'IDE.

Via la console on lance la commande suivante :

Installation avec Node.JS

```
npm install -g sass
```

Installation sous Windows

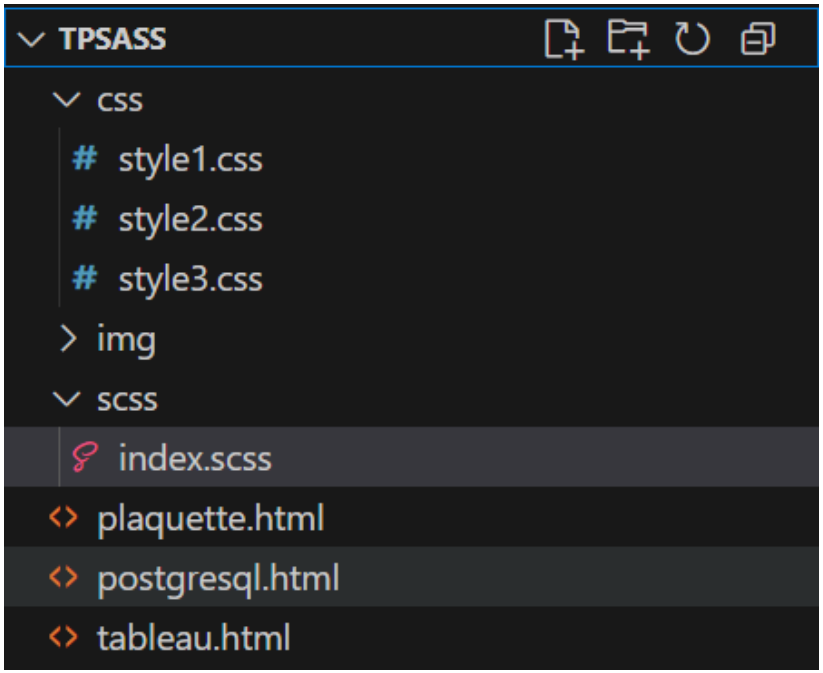
```
choco install sass
```

Installation sous MAC/Linux

```
brew install sass/sass/sass
```

Compilation Sass

- La compilation consiste **à transformer les scss en fichier css** et de l'appeler dans vos pages HTML.
- Pour cela, on créera un projet avec la structure suivante :

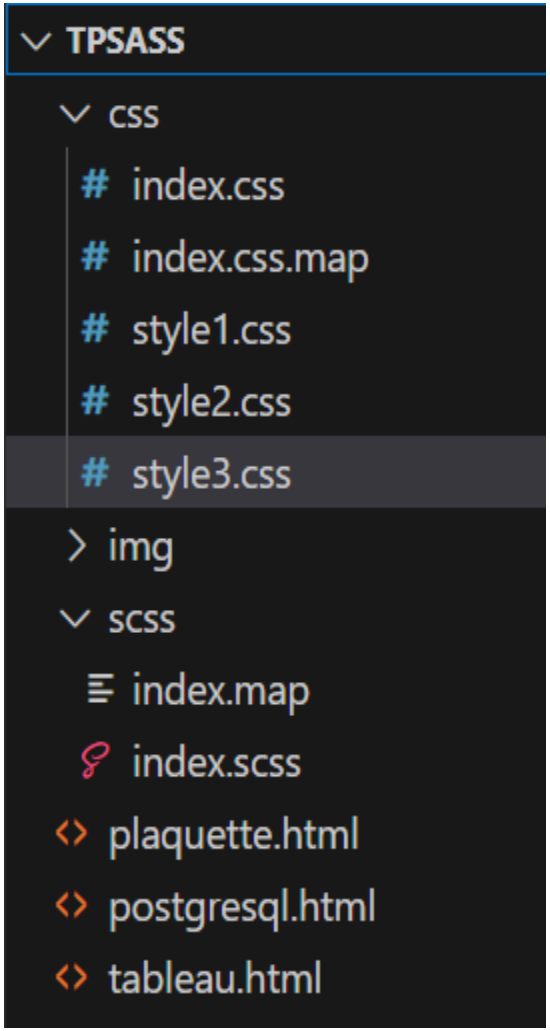


- On lance la commande :

```
PS D:\2025-2026\MMI\PROMOS\MMI2\SEMESTRE_3\SEANCE_2\TPSASS\scss> sass --watch .\index.scss index
[2025-09-23 23:13] Compiled index.scss to index.
Sass is watching for changes. Press Ctrl-C to stop.
```

Toute modification du **scss**, affectera automatiquement votre fichier **css**

Après, il suffit d'affecter le nouveau css à votre fichier html.




```
<!DOCTYPE
<html lan
    <head
        <
        <title>R312 - TPSASS</title>
        <link href="css/index.css" type="text/css" rel="stylesheet" />
    </head>
    <body>
        <div><table>
            <tr><td><img src="./img/logort.png" alt="RT"/></td>
                <td class="iut"><img src="./img/logoiut.png" alt="IUT" /></td>
                <td> </td></tr>
        </table></div>
        <h1>Département<br/>Réseaux et Télécoms</h1>
        <div class="couv"><img src="img/iut.png" alt="IUT" /></div>
        <h2 style="width: 50%;">Sommaire</h2>
        <div class="sommaire"><ul>
            <li><a href="#chap1">Réseaux et Télécoms</a></li>
```

context.

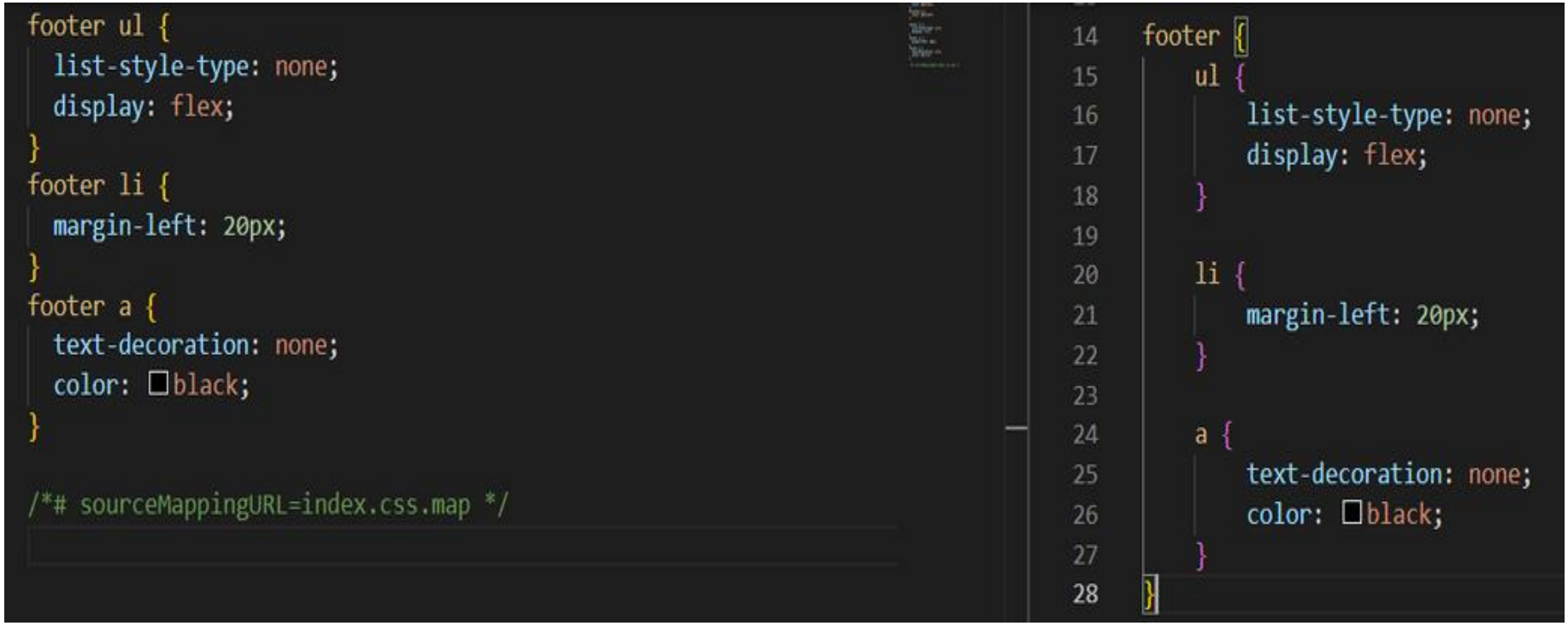
Widely available across major browsers (Baseline since 2015)

MDN Reference

Scss - Imbrication

Cette fonctionnalité des scss s'inscrit dans le cadre du principe DRY (**D**on't **R**epeat **Y**ourself).

Elle est très largement utilisée, pour éviter la redondance des sélecteurs CSS.



```css
footer ul {
  list-style-type: none;
  display: flex;
}
footer li {
  margin-left: 20px;
}
footer a {
  text-decoration: none;
  color: black;
}

/*# sourceMappingURL=index.css.map */
```

```scss
footer {
    ul {
        list-style-type: none;
        display: flex;
    }

    li {
        margin-left: 20px;
    }

    a {
        text-decoration: none;
        color: black;
    }
}
```

Cette fonctionnalité comporte de nombreux avantages comme la rapidité à laquelle le code sera écrit, la lisibilité et donc la maintenabilité du code car grâce à cette visibilité il est plus simple d'intervenir sur du code déjà existant.

Scss - La référence au parent

Avec le sélecteur & (l'esperluette), on a indiqué à Sass où il devait insérer le sélecteur du bloc parent.

```scss
.alert {
  // La classe Prente

  &:hover {
    font-weight: bold;
  }

  [dir=rtl] & {
    margin-left: 0;
    margin-right: 10px;
  }

  :not(&) {
    opacity: 0.8;
  }
}
```

Scss – Les variables

Les variables représentent un moyen important et puissant utilisé dans les **scss**.

Quand certaines valeurs reviennent régulièrement, la notion de variables répond parfaitement à ce problème.

Dans le cas où la variable change de valeur, l'impact sur les sélecteurs qui l'utilisent sera automatique.

La syntaxe générale est :

```
// À insérer au-début de main.scss
$color: #ff0;
```

```
$base-color: #c6538c;
$border-dark: rgba($base-color, 0.88);

.alert {
  border: 1px solid $border-dark;
}
```

Les variables - Exemples

```
//Autre exemple bidon
$var: 15px;
p {
  font-size: $var + 5px; // = 20px
  width: $var * (5+5) - 50px; // = 100px
}
```

```
h1, h2 {
    font-family: $head-font;
}
```

Les variables - Interpolation

Dans le cas de l'interpolation de variable en Sass, il s'agit de forcer l'évaluation d'une variable :

- Une chaîne de caractère
- Un sélecteur
- Un nom de propriété.

```
$variable: value;

// Variable Interpolation in selectors
.#{$variable}-text {
    property: $variable;
}

// Variable Interpolation in properties
.button {
    background-color: #{$variable};
    border: 1px solid #{$variable};
}
```

Les variables - Interpolation

```
$theme:     'glossy';
$property: 'radius';
$value:      4px;

body.#{$theme} {
  button {
    border-#{property}: $value;
  }
}
```

La directive @import

- Durant le processus de développement, il est souvent intéressant de diviser son code en plusieurs fichiers, pour mieux s'organiser.
- L'inconvénient est que cela risque d'augmenter le nombre de requêtes effectuées vers le serveur de production, et ralentit donc le chargement de la page.
- Dans le cas de plusieurs fichiers, on doit utiliser la directive @import pour appeler les fichiers entre eux.

Dans le cas de l'arborescence suivante, les appels d'importation se feront dans le fichiers ***main.scss***

```
sass
├── sections.scss
├── button.scss
├── cards.scss
├── contact.scss
├── header_footer.scss
├── reset.scss
├── typo.scss
├── config.scss
└── main.scss
```

La directive @import

Placez dans chaque fichier le contenu de la section correspondante.

Votre **main.scss** aura le code suivant grâce à la directive **@import !**

```
// Fichier main.scss
@import 'reset'; // en premier, évidemment
@import 'config'; // important de le mettre ici, sinon les autres
                  // fichiers n'auront pas accès à nos variables
@import 'buttons';
@import 'typo';
@import 'cards';
@import 'header_footer';
@import 'sections';
@import 'contact';
```

Les mixins dans les scss

Comme il est possible de stocker des données simples dans variables, on peut stocker des blocs de code entier, grâce au mixins

```scss
// Elle se définit grâce à la directive @mixin puis le nom que vous souhaitez lui attribuer
@mixin inline-block {
    // code
}

// Pour inclure votre mixins, une ligne suffit
@include inline-block;
```

```scss
@mixin inline-block($align, $margin : 10px) { // Si aucune valeur n'est précisée alors $margin vaut 10px
    display: inline-block;
    vertical-align: $align;
    margin-left: $margin;
}

// Appel de la mixin
.bloc {
    @include inline-block(middle);
}
```

Les mixins dans les scss

```scss
@mixin button {
    font-family: $head-font;
    font-size: 1.25rem;
    text-decoration: none;
    text-align: center;
    padding: .5rem;
    width: 12.5rem;
    border: none;
    display: block;
    float: none;
    margin: .5rem auto 0;
    background-color: $background-color;
    color: $color;
    &:hover {
        background-color: $hover-color;
    }
    @media #{$large-screen} {
```

```scss
@mixin reset-list {
  margin: 0;
  padding: 0;
  list-style: none;
}

@mixin horizontal-list {
  @include reset-list;

  li {
    display: inline-block;
    margin: {
      left: -2px;
      right: 2em;
    }
  }
}

nav ul {
  @include horizontal-list;
}
```