SASS

INSTALLATION

Étapes générales Pour installer Sass dans un projet HTML:

Sass nécessite Node.js pour fonctionner. Téléchargez et installez Node.js depuis le site officiel de Node.js.

Une fois Node.js installé, ouvrez votre terminal (ou invite de commandes) et exécutez la commande suivante pour installer Sass via npm : `npm install -g sass`

Créez un fichier style.scss dans votre projet et écrivez votre code Sass à l'intérieur.

Le Sass n'est pas lisible sur le navigateur, il faut alors compiler le Sass en CSS : `sass style.scss style.css`

Cette commande compile style.scss en style.css.

Pour compiler automatiquement vous pouvez utiliser --watch .

Intégrer le CSS dans votre HTML : <link rel="stylesheet" href="style.css">

QU'EST-CE QUE SASS ?

Sass (Syntactically Awesome Stylesheets) est un langage de préprocesseur CSS, ce qui signifie qu'il s'agit d'une extension de CSS permettant d'ajouter des fonctionnalités supplémentaires et de rendre l'écriture des feuilles de style plus efficace. Sass introduit une syntaxe plus expressive et propose plusieurs fonctionnalités qui ne sont pas présentes dans le CSS standard.

FONCTIONNALITÉS CLÉS DE SASS

Imbrication (**Nesting**): Vous pouvez imbriquer des sélecteurs pour représenter la structure hiérarchique du code HTML. Cela peut rendre le code plus lisible et mieux organiser les styles.

Variables : Sass permet l'utilisation de variables, ce qui facilite la gestion des valeurs réutilisables dans votre feuille de style. Cela peut améliorer la maintenance du code en évitant la duplication.

Mixins : Les mixins permettent de définir des ensembles de propriétés CSS réutilisables, similaires aux fonctions. Cela facilite la répétition de styles dans votre code.

Extend : L'instruction @extend permet de partager un ensemble de propriétés entre plusieurs sélecteurs, évitant ainsi la duplication de code.

Opérations mathématiques : Sass permet d'effectuer des opérations mathématiques directement dans le code CSS, ce qui peut être utile pour le design responsif.

IMBRICATION (NESTING)

L'imbrication (ou nesting) est une fonctionnalité qui permet d'imbriquer des sélecteurs CSS les uns dans les autres pour refléter la structure hiérarchique des éléments HTML. Cela facilite la gestion des règles CSS associées à une structure spécifique du document.

Feuille css

```css
nav {
  background-color: #333;
}

nav ul {
  list-style-type: none;
}

nav ul li {
  display: inline-block;
}

nav ul li a {
  text-decoration: none;
  color: white;
}

nav ul li a:hover {
  color: yellow;
}
```

Feuille sass

```sass
nav {
  background-color: #333;
  ul {
    list-style-type: none;
    li {
      display: inline-block;
      a {
        text-decoration: none;
        color: white;
        &:hover {
          color: yellow;
        }
      }
    }
  }
}
```

IMBRICATION (NESTING)

Les avantages de l'imbrication en SASS sont les suivants :

1.Lisibilité : Le code SASS reflète directement la structure du HTML, ce qui rend le code plus lisible et intuitif.

2.Réduction de la redondance : En évitant de répéter le sélecteur parent à chaque niveau, le code SASS est plus concis, ce qui réduit la redondance.

3.Maintenance facilitée : Si la structure HTML change, les ajustements nécessaires dans le code SASS sont souvent moins complexes, car ils reflètent naturellement la structure du document.

Cependant, il est important de ne pas abuser de l'imbrication, car une trop grande profondeur peut rendre le code difficile à comprendre. Une règle générale est de ne pas dépasser trois niveaux d'imbrication pour éviter la complexité excessive.

VARIABLES

En SASS, une variable est déclarée avec le signe dollar $ suivi du nom de la variable et de la valeur à lui attribuer.

```
$primary-color: #3498db;
$font-stack: 'Arial', sans-serif;
$base-padding: 10px;
```

Par convention ces variables sont à déclarer au début de votre feuille de styles SASS

Exemple d'utilisation d'une variable:

```
body {
    background-color: $primary-color;
}
```

VARIABLES

Déclarer des variables permet de :

- **Réutiliser du code** pour éviter la duplication
- **Faciliter la mise à jour** lorsqu'elle est utilisée à plusieurs endroits de votre feuille SASS
- **Améliorer la visibilité** en donnant un nom spécifique à vos variables
- **Personnaliser rapidement** en ajustant les valeurs à vos variables

Déclarer des variables en SASS offre une approche plus efficace et maintenable pour gérer les valeurs réutilisables dans les feuilles de style, ce qui contribue à un développement CSS plus propre et plus organisé.

LES MIXINS

Les **mixins** Sass nous permettent de créer des groupes de déclarations CSS en vue de les réutiliser plus tard dans notre code.

Permet de gagner beaucoup de temps en créant des groupes de règles.

Pour créer une **mixin** On va créer utiliser le préfixe **@mixin** suivi du nom de votre mixin.

Pour utiliser la mixin, il faut ajouter @include suivi du nom de la **mixin**

```scss
// Définition d'un mixin
@mixin bouton-style {
  padding: 10px 20px;
  font-size: 16px;
  border: 1px solid #ccc;
  border-radius: 5px;
}
```

```scss
// Utilisation du mixin
.button {
  @include bouton-style;
  background-color: #3498db;
  color: white;
}
```

LES MIXINS

Une **mixin** peut avoir des paramètres : le nom de votre **mixin** est suivi de parenthèses.
Le paramètre s'écrit comme les variables avec un $ devant

```scss
// Définition d'un mixin avec un argument
@mixin bouton-personnalise($couleur) {
  padding: 10px 20px;
  font-size: 16px;
  border: 1px solid $couleur;
  border-radius: 5px;
  background-color: $couleur;
  color: white;
}
```

```scss
// Utilisation du mixin avec un argument
.button {
  @include bouton-personnalise(#3498db);
}
```

l'utilisation de mixins en Sass offre une approche modulaire et flexible pour créer des styles CSS, améliorant ainsi la réutilisabilité et la maintenabilité du code.

EXTEND

L'instruction @extend en Sass est utilisée pour partager les propriétés d'une sélection CSS avec une autre sélection. Cela permet de réutiliser des styles d'une manière plus sémantique que l'utilisation de mixins. L'idée est de créer une classe CSS qui contient un ensemble de styles, puis d'étendre cette classe à d'autres sélecteurs où vous souhaitez appliquer ces styles.

```
// Définition d'une classe de style réutilisable
.bouton-commun {
  padding: 10px 20px;
  font-size: 16px;
  border: 1px solid #ccc;
  border-radius: 5px;
}
```

```
// Utilisation de extend pour appliquer les styles
.bouton {
  @extend .bouton-commun;
  background-color: #3498db;
  color: white;
}

.primary-bouton {
  @extend .bouton-commun;
  background-color: #e74c3c;
}
```

OPÉRATEURS

Nous pouvons utiliser des opérateurs en Sass:

- Opérateurs de concaténation
- Opérateurs arithmétiques
- Opérateurs de comparaisons
- Opérateurs logiques

OPÉRATEURS DE CONCATÉNATION

Sass supporte trois opérateurs de concaténation ou opérateurs de chaines de caractères différent :

Opérateur	Description
+	Retourne une chaine qui contient les 2 expressions de départ concaténées
-	Retourne une chaine qui contient les 2 expressions de départ concaténées Séparées par "-"
/	Retourne une chaine qui contient les 2 expressions de départ concaténées Séparées par "/"

OPÉRATEURS DE ARITHMÉTIQUES

Les opérateurs arithmétiques vont nous permettre d'effectuer des opérations arithmétiques (des calculs) entre des nombres.

Opérateur	Description
+	Addition
-	Soustraction
*	Multiplication
/	Division
%	Modulo

```scss
$largeur: 100px;
$marge: 10px;

.element {
  width: $largeur + $marge;
  height: $largeur - $marge;
  margin: $marge * 2;
  padding: $largeur / 2;
}
```

OPÉRATEURS DE COMPARAISON

Les opérateurs de comparaison nous permettent de comparer différentes valeurs entre elles. Lorsqu'on utilise un opérateur de comparaison, on demande à Sass d'effectuer la comparaison indiquée par l'opérateur. A l'issue de la comparaison, une valeur booléenne est automatiquement renvoyée : soit la valeur true si la comparaison est vérifiée par Sass, soit false si elle est jugée fausse.

Opérateur	Description
==	Permet de tester l'égalité sur les valeurs (renvoie true si les valeurs sont égales)
!=	Permet de tester la différence des valeurs (renvoie true si les valeurs sont différentes)
<	Permet de tester si une valeur est strictement inférieure à une autre
>	Permet de tester si une valeur est strictement supérieure à une autre
<=	Permet de tester si une valeur est inférieure ou égale à une autre

OPÉRATEURS DE LOGIQUES

Les opérateurs logiques sont des opérateurs qui vont principalement être utilisés au sein de conditions. Ils vont nous permettre de créer des tests plus robustes.

Sass supporte trois opérateurs logiques : **and**, **or** et **not**.

L'opérateur logique **and** nous permet d'effectuer plusieurs comparaisons. On va utiliser cet opérateur pour créer des codes comme cela : "si telle comparaison est vérifiée ET si telle autre comparaison l'est aussi…".

L'opérateur **and** renvoie la valeur true uniquement si chaque comparaison renvoie true.

L'opérateur logique **or** va lui renvoyer true dès qu'une comparaison est évaluée à true.

Enfin, l'opérateur logique **not** inverse le résultat logique d'une comparaison. Si une comparaison est évaluée à true de base et qu'on utilise l'opérateur not, par exemple, le résultat final sera false.

CONDITION IF… ELSE

La règle **@if** permet de créer une structure conditionnelle if. Cette structure conditionnelle permet d'exécuter un code si une expression (la condition dans la condition) est évaluée à true.

La règle **@if** est souvent accompagnée d'une règle **@else** qui permet d'exécuter un autre bloc de code dans le cas où l'expression du if est évaluée à false.

```
// Définition d'une variable
$couleur: #3498db;

// Utilisation de l'opérateur or dans la directive conditionnelle
.element {
  @if $couleur == #3498db or $couleur == #e74c3c {
    background-color: $couleur;
    color: white;
  } @else {
    background-color: #ccc;
    color: black;
  }
}
```

BOUCLE WHILE

Pour créer une boucle while en Sass , il faut utiliser la règle **@while**.
Le principe de cette boucle est de répéter un code en boucle tant qu'une condition de sortie de boucle n'est pas vérifiée.

Il faudra donc toujours bien faire attention à créer un code dans la boucle qui fait qu'on atteint la condition de sortie à un moment ou à un autre car dans le cas contraire on va créer une boucle infinie qui va faire planter notre code.

```scss
// Variable initiale
$compteur: 1;

// Boucle @while permettant de créer 5 classes element
@while $compteur <= 5 {
  .element-#{$compteur} {
    width: 50px * $compteur;
    height: 50px;
    background-color: #3498db;
    margin-bottom: 10px;
  }

  // Incrémentation du compteur
  $compteur: $compteur + 1;
}
```

IMPORT

En Sass, l'instruction **@import** est utilisée pour inclure le contenu d'un fichier Sass dans un autre. Cela permet de diviser votre code en plusieurs fichiers pour une organisation plus modulaire et une meilleure gestion.

```scss
// variables.scss
$couleur-principale: #3498db;
$taille-police: 16px;
```

```scss
// styles.scss
@import 'variables';

body {
  font-size: $taille-police;
  background-color: $couleur-principale;
}

.titre {
  color: $couleur-principale;
}
```