
Sass

Syntactically Awesome Style Sheets

SASS est un **préprocesseur CSS** agissant comme une extension de CSS et qui permet d'écrire notre code de manière plus efficace et plus pratique.

Avec SASS, on bénéficie de fonctionnalités supplémentaires qui ne sont pas présentes dans le CSS traditionnel. Par exemple, SASS nous permet d'utiliser des fonctions, des boucles, des variables et des conditions pour générer dynamiquement des styles CSS, ce qui rend le code plus flexible et plus puissant.

Une fois qu'on a écrit notre code SASS, il faudra le compiler en CSS3 pour qu'il puisse être interprété par les navigateurs web

En résumé, SASS rend notre code plus modulaire, plus réutilisable et plus facile à maintenir.

SASS

SCSS

SASS est le préprocesseur principal qui introduit toutes les fonctionnalités supplémentaires par rapport au CSS traditionnel.

SASS était la toute première version de Sass (Le préprocesseur). Cependant, en raison de sa syntaxe indentée, de nombreux développeurs ont préférés une syntexe plus familières, semblable à celle du CSS.

Et c'est à ce moment que SCSS (Sassy CSS) fut créé. Celui-ci fonctionne exactement de la même manière que SASS mais utilise une syntaxe similaire au CSS avec des accolades et des points virgules.

SASS

```
$couleur: #FF0000

.section
    background-color: $couleur
    color: #FFFFFF

    h1
        font-size: 24px
        font-weight: bold

    p
        font-size: 16px
```

SCSS

```
$couleur: #FF0000;

.section {
    background-color: $couleur;
    color: #FFFFFF;

    h1 {
        font-size: 24px;
        font-weight: bold;
    }

    p {
        font-size: 16px;
    }
}
```

INSTALLATION


node
JS


node
JS


npm

Installation

npm init
npm install sass
Créer le script dans le fichier package.json pour générer la compilation en CSS
npm run [nom du script] (Ex : npm run sass)

Installation

npm init
npm install sass
Créer le script dans le fichier package.json pour générer la compilation en CSS
npm run [nom du script] (Ex : npm run sass)

package.json

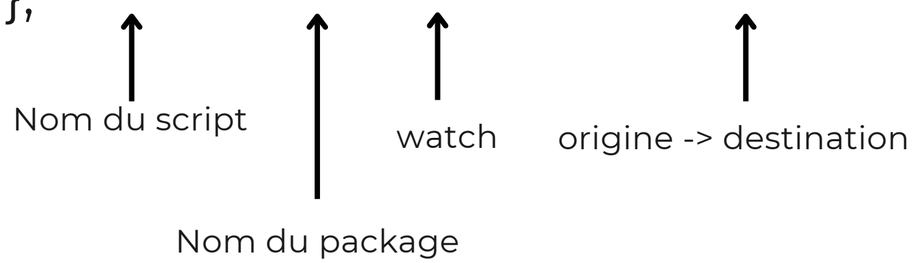
```
"scripts": {
    "sass": "sass -w assets/scss:assets/css"
 },
```

Installation

npm init
npm install sass
Créer le script dans le fichier package.json pour générer la compilation en CSS
npm run [nom du script] (Ex : npm run sass)

package.json

```
"scripts": {
    "sass": "sass -w assets/scss:assets/css"
},
```

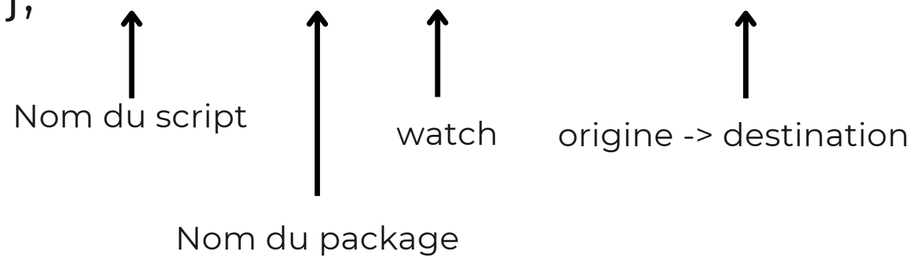


Installation

npm init
npm install sass
Créer le script dans le fichier package.json pour générer la compilation en CSS
npm run [nom du script] (Ex : npm run sass)

package.json

```
"scripts": {
    "sass": "sass -w assets/scss:assets/css"
},
```



LES VARIABLES

Une variable est un conteneur qui stocke une valeur spécifique et lui attribue un nom. En programmation, les variables servent à stocker des données pour une utilisation ultérieure dans le code.

L'utilisation de variables permet d'effectuer des modifications rapides et cohérentes dans notre code.

Par exemple, si on souhaite changer une valeur, comme une couleur ou une taille, il suffira de modifier la valeur de la variable, et cette modification sera répercutée sur tous les endroits où la variable est utilisée.

Cela simplifie la maintenance du code et permet des ajustements rapides.

Les variables

```
$couleur-principale: #FF0000;
$couleur-secondaire: #00FF00;
$espacement: 10px;
$police-principale: "Arial", sans-serif;
$rayon-bordures: 5px;
```

Les variables

```
$couleur-principale: #FF0000;
$couleur-secondaire: #00FF00;
$espacement: 10px;
$police-principale: "Arial", sans-serif;
$rayon-bordures: 5px;
```

```
.section {
    background-color: $couleur-principale;
    color: $couleur-secondaire;
    padding: $espacement;
    font-family: $police-principale;
    border-radius: $rayon-bordures;
}

.bouton {
    background-color: $couleur-secondaire;
    color: $couleur-principale;
    padding: $espacement;
    font-family: $police-principale;
}
```

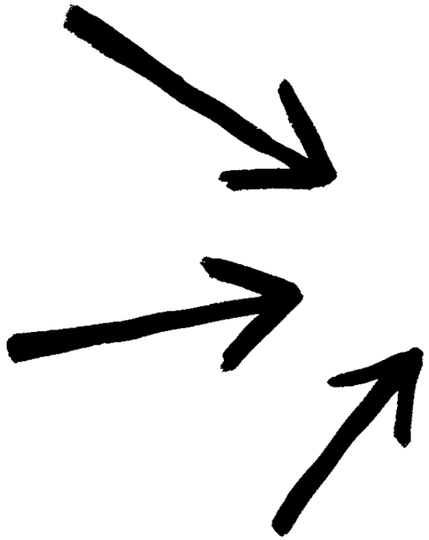
L'IMBRICATION

```scss
.page {
  background-color: #FFFFFF;

  .header {
    padding: 20px;
    background-color: #EFEFEF;

    .logo {
      width: 150px;
      height: 50px;
      background-image: url("logo.png");
    }

    .menu {
      ul {
        list-style: none;
        padding: 0;
        margin: 0;

        li {
          display: inline-block;
          margin-right: 10px;

          a {
            color: #000000;
            text-decoration: none;
```

L'imbrication (ou nesting) est l'une des fonctionnalités clés de SCSS qui permet de structurer et d'organiser le code CSS de manière plus lisible et plus maintenable.

Le nesting permet d'éviter la répétition des sélecteurs CSS et facilite la compréhension de la relation entre les différents éléments.

```scss
.page {
  background-color: #FFFFFF;

  .header {
    padding: 20px;
    background-color: #EFEFEF;

    .logo {
      width: 150px;
      height: 50px;
      background-image: url("logo.png");
    }

    .menu {
      ul {
        list-style: none;
        padding: 0;
        margin: 0;

        li {
          display: inline-block;
          margin-right: 10px;

          a {
            color: #000000;
            text-decoration: none;
```



```scss
.page {
  background-color: #FFFFFF;
}

.page .header {
  padding: 20px;
  background-color: #EFEFEF;
}

.page .header .logo {
  width: 150px;
  height: 50px;
  background-image: url("logo.png");
}

.page .header .menu ul {
  list-style: none;
  padding: 0;
  margin: 0;
}

.page .header .menu ul li {
  display: inline-block;
  margin-right: 10px;
}

.page .header .menu ul li a {
```

LES MIXINS

Les mixins

Les mixins sont une fonctionnalité puissante de SCSS permettant de créer des blocs de code réutilisable, avec ou sans paramètres, dans nos styles.

Elles agissent comme des "morceaux" de code que l'on peut incorporer à différents endroits de notre style, ce qui facilite la création de styles réutilisables et modulaires.

Les mixins

```scss
@mixin button {
  padding: 10px 20px;
  font-size: 14px;
  background-color: #007bff;
  color: #ffffff;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}

.button {
  @include button;
}
```

Les mixins

```scss
@mixin button($bg-color, $text-color) {
  padding: 10px 20px;
  font-size: 14px;
  background-color: $bg-color;
  color: $text-color;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}

.primary-button {
  @include button(#007bff, #ffffff);
}

.secondary-button {
  @include button(#ffffff, #007bff);
}
```

LES OPÉRATEURS MATHÉMATIQUES

Les opérateurs mathématiques

En SCSS, les opérateurs mathématiques permettent de réaliser des calculs, des comparaisons et des opérations logiques directement dans le code SCSS.
Cela facilite la manipulation des valeurs et l'adaptation dynamique de tes styles.

Opérateurs Mathématiques :

- Addition (+) : permet d'additionner deux valeurs.
- Soustraction (-) : permet de soustraire une valeur d'une autre.
- Multiplication (*) : permet de multiplier deux valeurs.
- Division (/) : permet de diviser une valeur par une autre.
- Modulo (%) : permet de calculer le reste de la division entière entre deux valeurs.

Les opérateurs mathématiques

```scss
$width: 200px;
$height: 300px;

.element {
  width: $width + 50px;
  height: $height - 20px;
  font-size: $width * 0.8;
  line-height: $height / 2;
  left: $width % 50;
}
```

LES OPÉRATEURS LOGIQUES

Les opérateurs logiques

Les opérateurs logiques en SCSS permettent de réaliser des opérations de comparaison et des opérations logiques.

Opérateurs logiques:

- Égal (==) : vérifie si deux valeurs sont égales.
- Différent (!=) : vérifie si deux valeurs sont différentes.
- Supérieur (>) : vérifie si une valeur est supérieure à une autre.
- Inférieur (<) : vérifie si une valeur est inférieure à une autre.
- Supérieur ou égal (>=) : vérifie si une valeur est supérieure ou égale à une autre.
- Inférieur ou égal (<=) : vérifie si une valeur est inférieure ou égale à une autre.

Les opérateurs logiques

```scss
$backgroundColor: '#ffffff';
$textColor: '#000000';
$hasBackground: true;
$hasTextColor: false;

.element {
  background-color: if($hasBackground, $backgroundColor, 'transparent');
  color: if($hasTextColor, $textColor, 'inherit');
}
```

Les opérateurs logiques

Je souhaite afficher un indicateur visuel pour chaque tâche en fonction de son état (terminée ou non terminée). Les tâches terminées doivent avoir un fond vert et un texte barré, tandis que les tâches non terminées doivent avoir un fond rouge.

Voici les instructions pour l'exercice :

1. Déclarer deux variables **$completedTaskColor** et **$incompleteTaskColor** pour représenter les couleurs de fond des tâches terminées et non terminées respectivement.
2. Déclarer une variable booléenne **$isCompleted** pour représenter l'état de la tâche.
3. Utiliser l'opérateur logique **if** pour définir la couleur de fond des éléments portant la classe **.task** en fonction de l'état de la tâche. Si la tâche est terminée (**$isCompleted** est **true**), utilise **$completedTaskColor**, sinon utilise **$incompleteTaskColor**.
4. Utiliser l'opérateur logique if pour définir le style de texte en fonction de l'état de la tâche. Si la tâche est terminée (**$isCompleted** est true), utilise la propriété **text-decoration: line-through**, sinon utilise la valeur par défaut.

LES PARTIALS

Les partials

Lorsqu'on travaille sur des projets de grande envergure, il peut être utile d'organiser son code en plusieurs fichiers pour une meilleure maintenabilité et réutilisabilité.

Les **partials** en SCSS sont une fonctionnalité qui permet de diviser le code SCSS en plusieurs fichiers, puis de les importer dans un fichier principal pour les compiler en un seul fichier CSS.

Les partials

Étape 1 : Créer les partials

Un fichier partial en SCSS est un fichier qui commence par un underscore (_) dans son nom de fichier. Par exemple, **_variables.scss**, **_mixins.scss**, **_header.scss**, etc.

Les fichiers partials contiennent généralement des sections spécifiques du code, telles que les variables, les mixins, les styles spécifiques à une section du site, etc.

Les partials

Étape 2 : Importation des partials

Dans le fichier SCSS principal, on peut importer les partials à l'aide de la directive @import.

Lorsqu'on importe un fichier SCSS partiel, on n'inclut pas l'underscore (_) ni l'extension (.scss) dans le nom de fichier.

SCSS recherchera automatiquement les fichiers avec ces noms et les importera dans le fichier principal.

```
sass/
  |_ abstracts/
  |    |_ _variables.scss
  |    |_ _functions.scss
  |    |_ _mixins.scss
  |    |_ _placeholders.scss
  |
  |_ base/
  |    |_ _reset.scss
  |    |_ _typography.scss
  |    |_ _utilities.scss
  |
  |_ components/
  |    |_ _buttons.scss
  |    |_ _forms.scss
  |    |_ _carousel.scss
  |    |_ ...
  |
  |_ layout/
  |    |_ _header.scss
  |    |_ _footer.scss
  |    |_ _sidebar.scss
  |    |_ ...
  |
  |_ pages/
  |    |_ _home.scss
  |    |_ _about.scss
  |    |_ _contact.scss
  |    |_ ...
  |
  |_ themes/
  |    |_ _theme-default.scss
  |    |_ _theme-dark.scss
  |    |_ ...
  |
  |_ vendors/
  |    |_ _bootstrap.scss
  |    |_ _jquery-ui.scss
  |    |_ ...
  |
  |_ main.scss
```

Les partials

Étape 2 : Importation des partials

```scss
// Abstracts
@use 'abstracts/variables';
@use 'abstracts/functions';
@use 'abstracts/mixins';
@use 'abstracts/placeholders';

// Base
@use 'base/reset';
@use 'base/typography';
@use 'base/utilities';

// Components
@use 'components/buttons';
@use 'components/forms';
@use 'components/carousel';

// Layout
@use 'layout/header';
@use 'layout/footer';
@use 'layout/sidebar';

// Pages
@use 'pages/home';
@use 'pages/about';
@use 'pages/contact';

// Themes
@use 'themes/theme-default';
@use 'themes/theme-dark';

// Vendors
@use 'vendors/bootstrap';
@use 'vendors/jquery-ui';
```

```
sass/
  |_ abstracts/
  |   |_ _variables.scss
  |   |_ _functions.scss
  |   |_ _mixins.scss
  |   |_ _placeholders.scss
  |
  |_ base/
  |   |_ _reset.scss
  |   |_ _typography.scss
  |   |_ _utilities.scss
  |
  |_ components/
  |   |_ _buttons.scss
  |   |_ _forms.scss
  |   |_ _carousel.scss
  |   |_ ...
  |
  |_ layout/
  |   |_ _header.scss
  |   |_ _footer.scss
  |   |_ _sidebar.scss
  |   |_ ...
  |
  |_ pages/
  |   |_ _home.scss
  |   |_ _about.scss
  |   |_ _contact.scss
  |   |_ ...
  |
  |_ themes/
  |   |_ _theme-default.scss
  |   |_ _theme-dark.scss
  |   |_ ...
  |
  |_ vendors/
  |   |_ _bootstrap.scss
  |   |_ _jquery-ui.scss
  |   |_ ...
  |
  |_ main.scss
```

Note importante :

Depuis la version 1.23 de Sass, l'utilisation de @use est recommandée plutôt que @import.

Les partials

Étape 3 : Compilation des partials

Lorsqu'on compile le code SCSS, le compilateur SCSS (par exemple, Sass, node-sass, ou tout autre outil utilisé) détecte les importations des partials et les inclut dans le fichier CSS compilé final.

Cela signifie qu'on obtient un seul fichier CSS résultant de l'ensemble des partials importés.

PROJET

TRAVEL TOGETHER