SASS

Table des matières

I. Contexte

Durée : 1 h 15

Environnement de travail : VSCode 1.42

Pré-requis : Notions de CSS, Notions de HTML

Contexte

L'évolution des usages du CSS a fait apparaître qu'avec le temps et la croissance d'une application web, il devient de plus en plus difficile de maintenir celle-ci.

En effet, le CSS est un langage qui ne permet pas beaucoup de structurer son code (comme il est possible de le faire en JavaScript, par exemple). Il faudra alors que le développeur fasse preuve de beaucoup d'auto-discipline pour que le code reste maintenable.

Les problèmes liés au fait que le CSS soit permissif ont pu trouver écho chez certains développeurs, qui ont mis au points des outils pour apporter plus de structure logique à un langage qui n'en possède pas dans son implémentation de base. Nous allons maintenant nous intéresser à un de ces outils, qu'on appelle un préprocesseur CSS, et plus particulièrement sur l'un deux : SASS.

II. Qu'est-ce que SASS ?

Objectifs

- Découvrir les problématiques de CSS
- Découvrir l'outil SASS
- Voir les deux syntaxes de SASS

Mise en situation

Nous allons ici voir ce qu'est un préprocesseur CSS et en quoi il permet un gain de productivité, autant au fil des développements que pendant la période de maintenance et d'évolution d'une application web.

Pour cela, nous définirons la notion de préprocesseur CSS et nous verrons comment cela s'applique pour le SASS.

Définition **Préprocesseur CSS**

Un préprocesseur CSS est un outil permettant d'interpréter du code respectant une syntaxe définie, que l'on appelle méta-langage, et de le traduire en CSS standard.

```scss
/* Fichier source style.scss */

$base-color: #C0C0C0;
$font-stack: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, Helvetica, Arial, sans-serif, "Apple Color Emoji", "Segoe UI Emoji", "Segoe UI Symbol";
$rounded-sm: 4px;

.box {
  color: $base-color;
  font-family: $font-stack;
  border-radius: $rounded-sm;
}
```

```css
/* Fichier style.css généré */

.box {
  color: #C0C0C0;
  font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, Helvetica, Arial, sans-serif, "Apple Color Emoji", "Segoe UI Emoji", "Segoe UI Symbol";
  border-radius: 4px;
}
/*# sourceMappingURL=style.css.map */
```

Il existe plusieurs préprocesseurs CSS, les plus utilisés étant LESS et SASS (*Syntactically Awesome StyleSheet*). Ils permettent de faciliter l'écriture de styles en fournissant au développeur une syntaxe plus puissante.

Sass étend les possibilités CSS classiques en permettant notamment d'utiliser des variables, des règles imbriquées, des mixins, des fonctions ou bien même encore de factoriser le code afin d'améliorer sa réutilisabilité et sa lisibilité.

En effet, nous allons pouvoir rendre modulaire notre style en le découpant en plusieurs fichiers SCSS. Cela facilitera l'ajout, la recherche ou la modification de notre contenu CSS.

La productivité en est donc améliorée à toutes les étapes du développement.

Méthode **Installation de SASS**

Pour utiliser SASS dans un projet d'application web, il est nécessaire d'installer les éléments permettant son fonctionnement. Plusieurs solutions sont utilisables et sont décrites dans la documentation officielle[1].

Nous choisissons la solution intégrée à l'éditeur VSCode, pour lequel il existe un plugin permettant d'exécuter la traduction de SASS vers CSS automatiquement.

Il est tout d'abord nécessaire de suivre les étapes suivantes :

- Installer Node.js[2] sur l'ordinateur :
- Télécharger la dernière version stable puis exécuter le .exe et lancer l'installation. Lors de l'installation il est important de laisser "npm packager manager" et "Add to PATH" cochés.
- S'assurer que Node et son gestionnaire de dépendances (npm) sont bien présents dans le path du système d'exploitation, vérifier avec les commandes `node -v` et `npm -v` qu'elles renvoient toutes les deux un numéro de version.
- Installer le package SASS, dans un terminal, taper la commande `npm install -g sass`. Cette commande installera le package SASS globalement sur l'ordinateur.

Il est maintenant possible d'installer le plugin **Live Sass Compiler** en suivant la procédure habituelle d'installation de plugin sur VSCode.

Une fois le plugin installé, un bouton *Watch sass* apparaîtra dans la barre du bas de VSCode. Cliquer sur le bouton compilera le SASS et fera apparaître deux nouveaux fichiers, un fichier **.css** et un fichier **.css.map**, qui sert de point de repère pour les futures compilations. Pas besoin de cliquer sur le bouton à chaque édition du fichier SASS, le plugin va observer le fichier : dès qu'il détecte une modification, il relancera la compilation en CSS.

Attention, seuls les fichiers .sass ou .scss seront détectés par le plugin Live Sass Compiler et permettront la compilation de ces fichiers en CSS.

1 Documentation SASS [en]
2 Site officiel de node js

Syntaxe .sass

Un fichier écrit en SASS s'identifie tout d'abord par son extension. Celle-ci peut être de deux formes : `.sass` et `.scss`.

Chacune d'entre elles emploie une syntaxe différente. Les fichiers `.sass`, qui suivent la syntaxe initiale de SASS, se distinguent entre autres par l'utilisation d'indentations pour structurer le document.

```
a:hover
  cursor: pointer

a:disabled
  color: red
  cursor: default
  pointer-events: none
```

Syntaxe .scss

Les fichiers `.scss`, quant à eux, permettent une écriture de code plus proche de celle du CSS. En effet, tout code valide en CSS3 sera valide pour un fichier `.scss`, c'est la syntaxe la plus utilisée et donc celle que nous utiliserons ici.

```
a:hover {
  cursor: pointer;
}

a:disabled {
  color: red;
  cursor: default;
  pointer-events: none;
}
```

Méthode **Architecture projet SASS**

Afin d'organiser au maximum le style de notre projet, il est préférable de découper au maximum nos fichiers de styles. Une architecture particulière permet d'optimiser cette organisation :

- un fichier main.scss pour importer les fichiers
- un fichier _base.scss comprend les bases de notre styles, les mixins et les variables que nous verrons plus loin dans ce cours.
- un fichier _layout.scss qui comprend les styles de nos différentes sections.
- un fichier _component.scss qui comprend les styles réutilisables de notre projet.

À noter que si le projet prend une certaine ampleur, il sera préférable de découper encore plus notre architecture en transformant nos fichier layout.scss et component.scss en dossier qui contiendront respectivement les fichiers scss appropriés.

Après avoir découpé vos fichiers et afin que votre fichier style.css soit généré, il suffit d'utiliser la commande sass scss/main.scss css/style.css qui permet de générer votre fichier style.css en fonction de vos fichier scss.

Exemple

```
/* fichier main.scss */
    @import 'layout';
    @import 'component';

  /* fichier _layout.scss */
      /* section de l'entête */
      #accueil {
```

```
          margin: 30px auto;
         width: 100%;
        .banner {
             font-size: 2rem;
          }
        }

/* fichier _component.scss */
    .bouton {
      border: 1px solid red;
      border-radius: 5px;
      .bouton:hover {
          border-radius: 0px;
        }
      }
/* fichier style.css généré avec la commande sass scss/main.scss css/style.css */
#accueil {
  margin: 30px auto;
  width: 100%;
}
#accueil .banner {
  font-size: 2rem;
}
.bouton {
  border: 1px solid red;
  border-radius: 5px;
}
.bouton .bouton:hover {
  border-radius: 0px;
}
```

Fondamental

CSS, de par son manque de structure et sa permissivité, rend complexe la maintenabilité du code. Afin de palier ce problème, il existe plusieurs outils, dont SASS, qui appartient à la famille des préprocesseurs et qui expose une syntaxe avec plus d'options de structure, qui vont faciliter le développement. Sass permet le découpage de fichiers SCSS qui se réuniront en un seul CSS après l'exécution d'une ligne de commande. Le code SASS n'est pas compris par le navigateur, et devra donc être traduit en CSS classique par un script. Il est possible d'utiliser des outils comme SassMeister[1] afin de tester en direct le CSS qui sera généré par votre code SCSS.

Voici le lien du plugin live sass compiler à télécharger : Live Sass Compiler[2]

De plus, afin de compléter l'installation, il faut créer un fichier en .scss .

1 https://www.sassmeister.com/
2 https://marketplace.visualstudio.com/items?itemName=glenn2223.live-sass

III. Exercice : Appliquez la notion

Question

[solution n°1 p.25]

Nous allons compiler du code SASS en CSS en utilisant les outils installés précédemment.

Nous allons créer un nouveau projet HTML qui contiendra deux boîtes, avec les classes `.header` et `.main-content`.

Importez le fichier de style CSS généré correspondant au code SASS suivant :

```scss
$background-dark: #222f3e;
$text-light: #feca57;
$text-dark: #202020;
$shadow-light: 4px 4px 10px #d6d6d6;

.header,
.main-content {
  padding: 20px;
}

.header {
  position: relative;
  background: $background-dark;
  color: $text-light;
  margin-bottom: 80px;
  width: 100%;
  height: 100px;
}

.main-content {
  color: $text-dark;
  margin: auto;
  text-align: center;
  box-shadow: $shadow-light;
  width: 100%;
  max-width: 1200px;
  min-height: 500px;
}

```

Indice :

- Repérez le bouton **Watch Sass** dans la barre du bas de VSCode
- Nous utilisons la syntaxe SCSS

IV. Les premières notions

Objectifs

- L'imbrication de sélecteurs
- Les variables en SASS
- L'héritage de styles

Mise en situation

Comme nous l'avons dit, pour faire face aux problèmes liés au manque de structure de CSS, SASS va mettre à notre disposition des structures de code qu'on retrouve normalement sur des langages de programmation. Nous allons commencer par présenter l'imbrication de sélecteurs, avant d'aborder l'utilisation des variables et enfin l'héritage de styles.

Méthode **Imbrication de sélecteurs**

En CSS, la définition de différents styles associés à des enfants d'un élément peut rapidement devenir lourde et répétitive. Grâce à l'utilisation de SASS, nous allons pouvoir la simplifier en intégrant directement la définition des styles des enfants dans la définition du style du parent.

Il devient alors simple de visualiser ce qui sera le style du parent de ce qui sera spécifique à chaque enfant.

Syntaxe

L'imbrication des sélecteurs consiste à respecter une syntaxe CSS valide pour chacun des éléments concernés, mais en intégrant la définition des enfants dans celle du parent.

Exemple

```
.text-element {
  text-align: right;
  margin: 1rem 0;

  .block-element {
    font-size: 3rem;
    line-height: 1em;

    .block-element:before, .block-element:after{
      content: "";
    }
  }

  span {
    font-family: Arial, serif;
    font-size : 0.8rem;
    margin-top: 1rem;
  }
}
/* Code CSS généré*/
  .text-element {
    text-align: right;
    margin: 1rem 0;
   }
   .text-element .block-element {
     font-size: 3rem;
     line-height: 1em;
  }
  .text-element .block-element.block-element:before, .text-element .block-element.block-element:after {
    content: "";
  }
  .text-element span {
     font-family: Arial, serif;
     font-size: 0.8rem;
     margin-top: 1rem;
```

```
36  }
```

Attention

Afin d'éviter de créer trop de spécificités dans les sélecteurs imbriqués (chaque imbrication sera traduite en CSS par un sélecteur classique `.parent > .enfant`), il convient de ne pas aller au-delà de 4 niveaux d'imbrication.

Méthode **Les variables**

Avec SASS, il devient possible de définir des variables pour stocker des codes couleur, des paramètres pour `box-shadow` ou encore un empilement de polices de caractère à utiliser. Ces variables permettent de centraliser le style et de ne pas se répéter. Si nous décidons par la suite de modifier une valeur, il suffira de la modifier à un seul endroit, et la mise à jour sera répercutée partout.

Syntaxe

En SASS, les variables se définissent avec le préfixe **$**.

Exemple

```scss
$header: #0059ff;
$head-font: Helvetica, Arial, sans-serif;
$base-value: 10px;

.form-header {
  background: $header;
}

.main-header {
  background: $header;
  font-family: $head-font;
}

h1 {
  font-family: $head-font;
}

p {
  font-size: $base-value + 10px; // la taille de la police des élément HTML <p> sera de 20px
  width: $base-value * (10+5) - 50px; // la largeur des élément HTML <p> sera de 100px
}

```

Méthode **Les Maths**

Sass permet d'utiliser des formules arithmétiques et mathématiques, notamment addition, soustraction, multiplication et division.

En ce qui concerne les divisions et les multiplications, il faut faire attention: même si SASS prend en charge diverses unités, lorsque celle-ci sont indiquées sur plusieurs valeurs de calcul il peut y avoir des problèmes de génération.

Exemple

```
.addition {
      width: 20% + 80%;
      /* Compiler en CSS */
      width: 100%;
  }
  .soustraction {
      height: 60% - 20%;
      /* Compiler en CSS */
        height: 40%;
    }
    .multiplication {
      width: 20 x 80px;
      /* Compiler en CSS */
        width: 1600px;
    }
    .division {
    width: 60px / 4;
      /* Compiler en CSS */
        width: 15px;
    }
```

Méthode **L'esperluette (&)**

Mettre une esperluette & en préfixe devant un sélecteur le reliera directement au sélecteur parent. Ce raccourci permet d'éviter la répétition de classe.

Exemple

```
/* Code Sass */
      ul {
      list-style-decoration: none;
        text-align: center;
        li {
          display: inline;
          color: red;
             &:hover {
               color: green;
          }
        }
    }
```

Méthode **L'interpolation**

L'utilisation d'une variable en tant qu'instruction CSS se fait en plaçant la variable entre `{ }` et précédée de `#`. C'est ce qu'on appelle l'**interpolation**. Le langage va interpréter l'instruction donnée en remplaçant la variable placée entre accolades, et la remplacer par sa valeur au moment de la compilation.

Exemple

```
$media-small-screen: "screen and (max-width: 480px)";

.list-element {
  @media #{$media-small-screen} {
    max-width: 85%;
  }
```

```
}
/* Code CSS généré */
  @media screen and (max-width: 480px) {
    .list-element {
      max-width: 85%;
    }
  }
```

Méthode **L'héritage**

Il est courant, en CSS, de répéter du code. Certaines propriétés sont très récurrentes, comme `flex` avec `align-items` et `justify-content`. L'héritage nous permet d'écrire ces propriétés une seule fois et de créer une référence vers celles-ci à chaque fois que nous en avons besoin, ce qui évite une duplication de code.

Syntaxe

Sass nous permet grâce à la règle @extend de mettre en place un système d'héritage d'un sélecteur à un autre.

Pour l'utiliser, nous allons définir un ensemble de propriétés CSS qui seront liées à un sélecteur. Puis avec le @extend, nous allons partager cet ensemble de propriétés à d'autres sélecteurs.

Correctement appliqué, cela permet d'organiser son code en groupant les règles plutôt que les sélecteurs. La maintenabilité du code en est donc d'autant plus optimisée.

Exemple

```
.large-font {
  font-size: 2em;
}

.information {
  @extend .large-font;
  min-height: 18rem;
  background-color: #fff;
  color: blue;
}

.action {
  @extend .large-font;
  background-color: red;
  color: #fff;
}

%flex-centered {
  display: flex;
  align-items: center;
  justify-content: center;
} // Cet élément est un placeholder, il n'existe pas en tant que tel dans le DOM

.icon-wrapper {
  @extend %flex-centered;
  position: absolute;
  top: -10px;
  right: 5px;
  width: 20px;
  height: 20px;
  border-radius: 20px;
```

```
  font-size: 0.8rem;
}
/* Code CSS généré*/
  .large-font, .action, .information {
    font-size: 2em;
  }
  .information {
    min-height: 18rem;
    background-color: #fff;
    color: blue;
  }
  .action {
    background-color: red;
    color: #fff;
  }
  .icon-wrapper {
    display: flex;
    align-items: center;
    justify-content: center;
  }
  .icon-wrapper {
    position: absolute;
    top: -10px;
    right: 5px;
    width: 20px;
    height: 20px;
    border-radius: 20px;
    font-size: 0.8rem;
  }
```

Fondamental

Grâce à SASS, le CSS devient proche d'un langage de programmation à part entière, et il devient possible de factoriser des éléments par l'utilisation de variables ou de l'héritage. De plus, il permet de simplifier la définition de styles imbriqués en autorisant une syntaxe plus claire et concise.

V. Exercice : Appliquez la notion

Question

[solution n°2 p.25]

À l'aide des notions que nous avons vues dans cette partie, vous allez créer et styliser trois boutons de différentes couleurs.

- Vous utiliserez un des sélecteurs imbriqués si possible
- Vous stockerez au minimum les couleurs dans des variables
- Vous utiliserez un héritage depuis un *placeholder* pour centrer le texte dans le bouton

VI. Notions avancées

Objectifs

- Les mixins
- Les fonctions
- Les conditions
- Les boucles

Mise en situation

Dans le cas d'une application de plus grande ampleur, les variables et l'héritage peuvent ne pas être suffisants. Nous allons maintenant voir une série d'outils qui permettent d'industrialiser la création de styles. Nous allons apprendre comment créer des fonctions et des structures de contrôle, comme des conditions et des boucles.

Méthode **Les mixins**

En plus de permettre une factorisation des opérations répétitives grâce à l'utilisation de fonctions, SASS nous donne la possibilité de factoriser des blocs de style complets par l'utilisation de mixins.

Un **mixin** peut être vu comme une variable contenant un ensemble de styles, à laquelle il est possible de passer des arguments. Lors de son utilisation, l'appel à un mixin retourne l'ensemble des styles spécifiés en fonction des paramètres fournis.

Syntaxe

La définition d'un mixin se fait par l'instruction @mixin, suivie du nom du mixin et de ses paramètres entre parenthèses. Son contenu est défini ensuite entre deux accolades.

L'inclusion d'un mixin se fait par le mot-clé @include suivi du nom de la mixin.

Exemple

```
@mixin button {
      width: 12.5rem;
      display: block;

      font: {
      size: 1.25rem;
      }

      text: {
        decoration: none;
        align: center;
      }
    }

    .info, .valid {
      @include button;
      color: green;
      &:hover {
        background-color: darken(green, 10%);
      }
    }
   /* Code CSS généré */
.info,
  .valid {
   width: 12.5rem;
```

```
  display: block;
  font-size: 1.25rem;
  text-decoration: none;
  text-align: center;
  color: green;
}
.info:hover,
.valid:hover {
  background-color: #004d00;
}
```

Exemple

```
@mixin button($color, $background-color, $hover-color) {
    width: 12.5rem;
    display: block;
    color: $color;
    background-color: $background-color;

    font: {
      size: 1.25rem;
  }

    text: {
      decoration: none;
      align: center;
    }

    &:hover {
      background-color: $hover-color;
    }
  }

  .info, .valid {
    @include button(green, null, darken(green, 10%));
  }
 /* Code CSS généré */
.info,
.valid {
    width: 12.5rem;
     display: block;
     color: green;
     font-size: 1.25rem;
     text-decoration: none;
      text-align: center;
}
.info:hover,
.valid:hover {
    background-color: #004d00;
}
```

Méthode **Les fonctions**

De la même manière que le CSS propose des fonctions utilisées comme valeurs pour certaines propriétés (comme `rgb()` qui renvoie une couleur à partir de trois paramètres), SASS embarque lui aussi un ensemble de fonctions.

La liste complète des fonctions est disponible sur la page de documentation sur les modules[1].

Exemple

```scss
$brand-color: hsl(219, 100%, 50%);

.text-element {
  color: darken($brand-color, 10%) // Assombri la couleur $brand-color de 10%.
  background: lighten($brand-color, 5%) // Eclairci la couleur $brand-color de 5%
  width: round(33vw) // Arrondi la valeur numérique passée en paramètre
}
/* Code CSS généré */
    .text-element {
     color: #0047cc;
     background: #1a6aff;
     width: 33vw;
    }
```

Méthode **La structure de données list**

SASS met à disposition deux structures de données qui s'apparentent à des tableaux ou des objets en JavaScript.

`List` permet de donner une série de valeurs à une variable. Il est possible d'itérer sur cette liste, comme nous le verrons dans le paragraphe sur `@each`.

```scss
$sizes: 40px, 50px, 80px;
```

Plusieurs fonctions permettent alors d'interagir avec cette liste : il est possible d'ajouter, de supprimer des éléments et d'exécuter du code pour chaque valeur, par exemple.

Méthode **La structure de données map**

`Map` est une autre manière de structurer une liste de valeurs, proche du fonctionnement d'un objet JavaScript. Pour chaque élément, une clé est associée à une valeur.

```scss
$colors: (
  'success': $success,
  'default': $default,
  'error': $error,
);
```

Méthode **Les fonctions personnalisées**

SASS nous permet également de définir nos propres fonctions personnalisées, ce qui améliore la réutilisabilité et réduit le code dupliqué.

1 Documentation SASS built-in modules [en]

Syntaxe

L'utilisation d'une fonction SASS se fait de la même manière que pour une fonction CSS, c'est-à-dire par son nom, suivi des paramètres souhaités entre parenthèses à la place d'une valeur de propriété.

La création d'une fonction personnalisée se fait grâce à l'utilisation du mot-clé `@function`, suivi du nom de la fonction et de ses paramètres entre parenthèses. Le contenu de la fonction est ensuite écrit entre deux accolades et doit se terminer par l'instruction `@return`, qui permet de renvoyer la valeur obtenue.

Exemple

```
@function table-width($columns, $column-size) { // Calcule la taille d'un tableau en fonction du nombre de colonnes et de leur taille.
  @return $columns * $column-size;
}

.table-quote {
  width: table-width(5, 50px);
}
/* Code CSS généré */
    .table-quote {
      width: 250px;
    }
```

Les structures conditionnelles

SASS met à disposition les outils nécessaire pour appliquer un style de manière conditionnelle. Il est possible d'appliquer cette logique au CSS grâce à l'application de tests booléens et d'instructions conditionnelles.

Méthode **Les tests booléens**

De manière générale, un test booléen consiste en une instruction retournant vrai ou faux selon qu'il est vérifié ou non. En ce qui concerne SASS, un test booléen peut aussi retourner la valeur `null`, auquel cas il est considéré comme faux.

Syntaxe

La mise en place d'un test booléen se fait par l'utilisation d'opérateurs de comparaison à ajouter entre les deux valeurs à comparer.

En SASS, les opérateurs booléens sont les suivants :

- == ou != pour vérifier respectivement **l'égalité** ou **l'inégalité** entre les valeurs à droite et à gauche
- < ou > pour vérifier si la valeur à gauche de l'opérateur est **inférieure ou supérieure** à celle de droite
- <= ou >= pour vérifier si la valeur à gauche de l'opérateur est **inférieure ou égale**, ou **supérieure ou égale**, à celle de droite

Il est possible de cumuler plusieurs tests booléens en les séparant par des opérateurs logiques :

- `and` : le test est vrai si chaque condition séparée par `and` est vraie
- `or` : vrai si au moins une des conditions séparées par `or` est vraie
- `not` : test vrai si les conditions après `not` sont fausses

Il existe trois instructions conditionnelles disponibles avec SASS : les instructions `@if` et `@else if` suivies d'un test conditionnel, et l'instruction `@else` qui sera vérifiée si les conditions précédentes n'ont pas été vraies.

Syntaxe

```scss
@mixin button($color, $background-color, $hover-color) {
  &:hover {
    @if $background-color == 'white' {
      background-color: darken($hover-color, 5%);
    }

    @else if $background-color == 'red' {
      background-color: #00a900;
    }

    @else {
      background-color: lighten($hover-color, 8%);
    }
  }
}

.info, .valid {
  @include button(green, "red", green);
}
/* Code CSS généré */
    .info:hover, .valid:hover {
      background-color: #00a900;
    }
```

Les boucles

Pour éviter de devoir répéter un ensemble d'instructions identiques, SASS autorise l'utilisation de structures itératives, ou boucles.

SASS définit trois types de boucles différentes : **while**, **for** et **each**.

Méthode **La boucle while**

La boucle `while` est une structure itérative qui s'exécute en fonction de la vérification d'un test booléen initial. Les instructions fournies seront exécutées tant que la condition de ce test est vraie.

Syntaxe

L'écriture d'une boucle `while` se définit par l'utilisation du mot-clé `@while` suivi d'un test booléen entre parenthèses. Tant que la condition sera vérifiée, les instructions entre accolades seront exécutées.

Attention

Il est important de s'assurer que la condition initiale devient invalide au bout d'un certain nombre d'itérations, au risque de se retrouver dans une boucle infinie, c'est-à-dire dont les instructions s'exécutent indéfiniment et bloquent le navigateur.

Exemple

```scss
/* Code SCSS */
$value: 10;
$test: 0;

@while($test < $value) { // La condition est vraie tant que $test est inférieur à $value, c'est à dire inférieur à 10
```

```
  $test: $test + 1; // test s'incrémente de 1 à chaque itération de la boucle
} // Lorsque $test atteint 10 alors la condition devient fausse et la boucle s'interrompt

.element {
  width: $test+px; // Lors de l'évaluation de $test sa valeur est de 10
}
c
```

Méthode **La boucle for**

La boucle `for` est une structure itérative qui s'exécute depuis une valeur de départ jusqu'à une valeur de fin, en retournant à chaque itération la valeur actuelle. Cela signifie, qu'à chaque passage de la boucle, nous pouvons connaître la valeur actuelle de l'itération.

Syntaxe

L'écriture d'une boucle `for` se définit par l'utilisation du mot-clé `@for` et suit la forme suivante : `@for $i from valeur_depart through valeur_fin`.

Ici, `$i` est la variable servant à contenir la valeur itérée, `valeur_depart` définit la valeur de l'itération de départ et `valeur_fin` définit la valeur de l'itération de fin.

Pour chacune des itérations, les instructions entre accolades seront exécutées.

Exemple

```
/* Code SCSS */
$test: 0;

@for $i from 0 through 10 {
  $test: $i;
}

.element {
  width: $test+px;
}
/* Code CSS généré */
    .element {
      width: 10px;
    }
```

Méthode **La boucle each**

La boucle `each` est une structure itérative qui s'exécute pour chaque élément d'une liste passée en paramètre, en retournant à chaque itération la valeur de la liste sur laquelle elle est en train d'itérer.

Syntaxe

L'écriture d'une boucle `each` se définie par l'utilisation du mot-clé `@each` et suit la forme : `@each $value in $values`.

`$values` est une liste d'éléments et `$value` est la valeur de l'itération actuelle de la boucle.

Exemple

```scss
/* Code SCSS qui permet de gérer le rythme des titres */
    @each $titre, $fonte in (h1, 2.5rem), (h2, 2rem), (h3, 1.5rem)
   {
     #{$titre} {
      font-size: $fonte;
        }
    }

    /* Code CSS généré */
    h1 {
      font-size: 2.5rem;
    }
    h2 {
      font-size: 2rem;
    }
    h3 {
    font-size: 1.5rem;
   }
```

Fondamental

L'utilisation de fonctionnalités avancées de SASS nous ouvre beaucoup de possibilités en termes de factorisation du code et de maintenabilité, tout en conservant une grande simplicité syntaxique. En effet, l'utilisation de mixins généralistes permet de regrouper les styles d'éléments visuellement similaires, tandis que les fonctions, les structures conditionnelles ou les boucles regroupent les fonctionnalités identiques d'un élément à un autre.

VII. Exercice : Appliquez la notion

Question

[solution n°3 p.27]

En vous basant sur l'exercice précédent, où nous avions construit trois boutons en SASS, vous allez factoriser la définition du bouton afin de pouvoir réutiliser ce code plus simplement.

- Le style du bouton doit venir d'une mixin avec le paramètre de couleur
- Une condition sera utilisée pour que le texte du bouton soit affiché en blanc seulement quand le bouton est bleu
- Dans le cas contraire, le texte devra être de la couleur de fond du bouton, mais 20 % plus clair, calculé automatiquement

Vous vous baserez sur le code du premier exercice.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="style.css">
    <title>Document</title>
</head>
<body>
    <div class="test">
        <button class="btn info">I'm blue</button>
        <button class="btn error">I'm red</button>
        <button class="btn success">I'm green</button>
    </div>
```

```
</body>
</html>
```

```
%flex-centered {
  display: flex;
  align-items: center;
  justify-content: center;
}

$info: #0059ff;
$success: #00b894;
$error: #d63031;
$white: #ffffff;

.btn {
  @extend %flex-centered;
  border-radius: 4px;
  border: none;
  padding: 10px 30px;
  margin: 5px;
  color: $white;

  &.info {
    background: $info;
  }

  &.error {
    background: $error;
  }

  &.success {
    background: $success;
  }
}

```

Indice :

Utilisez `lighten()` pour éclaircir une couleur.

VIII. Auto-évaluation

A. Exercice final

Exercice 1

[solution n°4 p.28]

Exercice

Qu'est-ce-qu'un préprocesseur ?

- ○ C'est une pièce à installer à côté du processeur de l'ordinateur
- ○ C'est un outil qui transforme du code d'un langage de programmation vers un autre
- ○ C'est ce qui permet de faire une boucle sur une liste d'éléments CSS

Exercice

Le préprocesseur SASS peut s'utiliser avec la/les syntaxe(s)...

- [ ] GLASS
- [ ] SASS
- [ ] SCSS
- [ ] LESS

Exercice

Sélectionnez le code valide pour la déclaration d'une variable en SASS.

- ○ const fonts = -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, Helvetica, Arial ;
- ○ #{$fonts} : -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, Helvetica, Arial ;
- ○ fonts : -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, Helvetica, Arial ;
- ○ $fonts : -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, Helvetica, Arial ;

Exercice

L'héritage en SASS se fait en utilisant la syntaxe...

- ○ @extend
- ○ @inherit
- ○ @fromParent

Exercice

En SASS, on peut hériter...

- [ ] D'une autre classe
- [ ] Uniquement du sélecteur parent
- [ ] D'un placeholder
- [ ] Uniquement si l'élément duquel on souhaite hériter est dans la même imbrication

Exercice

La fonction SASS qui permet d'assombrir une couleur est...

- ○ lighten()
- ○ blacken()
- ○ darken()
- ○ Il n'existe pas de fonction pour assombrir une couleur

Exercice

SASS met à disposition plusieurs manières de stocker des éléments pour les réutiliser dans le code :

- [ ] Un map
- [ ] Une list
- [ ] Un tableau
- [ ] Une variable

Exercice

Il est possible d'effectuer des boucles en SASS de plusieurs manières.

- [ ] Avec @foreach
- [ ] Avec @each
- [ ] Avec @for
- [ ] Avec @while
- [ ] Avec @loop

Exercice

Une mixin ne prend aucun paramètre.

- ○ Vrai
- ○ Faux

Exercice

Quels sont les opérateurs de condition valides en SASS ?

- [ ] @case
- [ ] @if
- [ ] @else
- [ ] @or
- [ ] @else if

B. Exercice : Défi

Dans le but de mettre en pratique tous les éléments que nous avons vus sur SASS, vous allez écrire le code permettant de générer un ensemble de classes *utilities*. Une classe *utility* est une classe CSS qui vient appliquer un seul style sur un élément HTML, découper ses styles de manière très atomiques et permet, en les cumulant, d'obtenir un style complet.

Question

[solution n°5 p.30]

- Vous définirez une liste de 3 couleurs que vous nommerez *info*, *success*, et *error* respectivement bleu, vert, et rouge
- Pour ces couleurs, vous ferez générer par SASS les classes utilities permettant d'utiliser ces couleurs en couleur de texte, en couleur de fond et en bordure
- Vous devrez utiliser des variables, une boucle et une mixin

Indice :

La boucle `@each` semble adaptée dans cette situation.

Solutions des exercices

p. 7 **Solution n°1**

index.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="style.css">
    <title>Document</title>
</head>
<body>
  <header class="header">
    Je suis le header
  </header>

  <main class="main-content">
    Je suis le contenu principal
  </main>
</body>
</html>
```

style.css

```css
.header,
.main-content {
  padding: 20px;
}

.header {
  position: relative;
  background: #222f3e;
  color: #feca57;
  margin-bottom: 80px;
  width: 100%;
  height: 100px;
}

.main-content {
  color: #202020;
  margin: auto;
  text-align: center;
  -webkit-box-shadow: 4px 4px 10px #d6d6d6;
          box-shadow: 4px 4px 10px #d6d6d6;
  width: 100%;
  max-width: 1200px;
  min-height: 500px;
}
/*# sourceMappingURL=style.css.map */
```

p. 12 **Solution n°2**

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="style.css">
    <title>Document</title>
</head>
<body>
    <div class="test">
        <button class="btn info">I'm blue</button>
        <button class="btn error">I'm red</button>
        <button class="btn success">I'm green</button>
    </div>
</body>
</html>
```

```scss
%flex-centered {
  display: flex;
  align-items: center;
  justify-content: center;
}

$info: #0059ff;
$success: #00b894;
$error: #d63031;
$white: #ffffff;

.btn {
  @extend %flex-centered;
  border-radius: 4px;
  border: none;
  padding: 10px 30px;
  margin: 5px;
  color: $white;

  &.info {
    background: $info;
  }

  &.error {
    background: $error;
  }

  &.success {
    background: $success;
  }
}

```

p. 19 **Solution n°3**

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="style.css">
    <title>Document</title>
</head>
<body>
    <div class="test">
        <button class="btn info">I'm blue</button>
        <button class="btn error">I'm red</button>
        <button class="btn success">I'm green</button>
    </div>
</body>
</html>
```

```scss
%flex-centered {
  display: flex;
  align-items: center;
  justify-content: center;
}

$info: #0059ff;
$success: #00b894;
$error: #d63031;
$white: #ffffff;

@mixin myButton ($color) {
  @extend %flex-centered;
  border-radius: 4px;
  border: none;
  padding: 10px 30px;
  margin: 5px;
  background: $color;

  @if $color == $info {
    color: $white;
  }

  @else {
    color: lighten($color, 20%);
  }
```

```
27 }
28
29 .btn {
30   &.info {
31     @include myButton($info);
32   }
33
34   &.error {
35     @include myButton($error);
36   }
37
38   &.success {
39     @include myButton($success);
40   }
41 }
42
```

Exercice p. 20 **Solution n°4**

Exercice

Qu'est-ce-qu'un préprocesseur ?

- ○ C'est une pièce à installer à côté du processeur de l'ordinateur
- ◉ C'est un outil qui transforme du code d'un langage de programmation vers un autre
- ○ C'est ce qui permet de faire une boucle sur une liste d'éléments CSS

Exercice

Le préprocesseur SASS peut s'utiliser avec la/les syntaxe(s)...

- ☐ GLASS
- ☑ SASS
- ☑ SCSS
- ☐ LESS

Exercice

Sélectionnez le code valide pour la déclaration d'une variable en SASS.

- ○ const fonts = -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, Helvetica, Arial ;
- ○ #{$fonts} : -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, Helvetica, Arial ;
- ○ fonts : -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, Helvetica, Arial ;
- ◉ $fonts : -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, Helvetica, Arial ;

Exercice

L'héritage en SASS se fait en utilisant la syntaxe...

- ◉ @extend
- ○ @inherit
- ○ @fromParent

Exercice

En SASS, on peut hériter...

- ☑ D'une autre classe
- ☐ Uniquement du sélecteur parent
- ☑ D'un placeholder
- ☐ Uniquement si l'élément duquel on souhaite hériter est dans la même imbrication

Exercice

La fonction SASS qui permet d'assombrir une couleur est...

- ○ lighten()
- ○ blacken()
- ◉ darken()
- ○ Il n'existe pas de fonction pour assombrir une couleur

Exercice

SASS met à disposition plusieurs manières de stocker des éléments pour les réutiliser dans le code :

- ☑ Un map
- ☑ Une list
- ☐ Un tableau
- ☑ Une variable

Exercice

Il est possible d'effectuer des boucles en SASS de plusieurs manières.

- ☐ Avec @foreach
- ☑ Avec @each
- ☑ Avec @for

- [x] Avec @while
- [ ] Avec @loop

Exercice

Une mixin ne prend aucun paramètre.

- ○ Vrai
- ◉ Faux

Exercice

Quels sont les opérateurs de condition valides en SASS ?

- [ ] @case
- [x] @if
- [x] @else
- [ ] @or
- [x] @else if

p. 22 **Solution n°5**

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="style.css">
    <title>Document</title>
</head>
<body>
    <div class="test">
        <div class="color-info">text blue</div>
        <div class="color-error">text red</div>
        <div class="color-success">text green</div>

        <div class="background-info">background blue</div>
        <div class="background-error">background red</div>
        <div class="background-success">background green</div>

        <div class="border-info">border blue</div>
        <div class="border-error">border red</div>
        <div class="border-success">border green</div>
    </div>
</body>
</html>
```

```css
.test div {
  box-sizing: border-box;
  width: 400px;
  height: 60px;
  margin: 5px 0;
  padding: 5px 10px;
}
```

```scss

$info: #0059ff;
$success: #00b894;
$error: #d63031;
$white: #ffffff;

$colorsMap: (
  'info': $info,
  'success': $success,
  'error': $error
);

@mixin colorUtilities($property) {
  @each $name, $value in $colorsMap {
    &-#{$name} {
      #{$property}: $value;
      @if $property == 'border-color' {
        border-width: 2px;
        border-style: solid;
      }
    }
  }
}

.color {
  @include colorUtilities(color);
}

.background {
  @include colorUtilities(background-color);
}

.border {
 @include colorUtilities(border-color);
}
```

```scss
.test div {
  -webkit-box-sizing: border-box;
          box-sizing: border-box;
  width: 400px;
  height: 60px;
  margin: 5px 0;
  padding: 5px 10px;
}

.color-info {
  color: #0059ff;
}

.color-success {
  color: #00b894;
}

.color-error {
  color: #d63031;
}

.background-info {
  background-color: #0059ff;
```

```
}

.background-success {
  background-color: #00b894;
}

.background-error {
  background-color: #d63031;
}

.border-info {
  border-color: #0059ff;
  border-width: 2px;
  border-style: solid;
}

.border-success {
  border-color: #00b894;
  border-width: 2px;
  border-style: solid;
}

.border-error {
  border-color: #d63031;
  border-width: 2px;
  border-style: solid;
}
/*# sourceMappingURL=style.css.map */
```