Sass (Syntactically Awesome Stylesheets)

Sass est un pré processeur qu'on retrouve régulièrement dans les projets. Un préprocesseur est un programme qui procède à des transformations sur un code source, avant l'étape de traduction proprement dite (compilation ou interprétation).

Sass by Romain Van-Damme 2020

À quoi cela sert-il ?

Il permet d'apporter une modularité aux feuilles de style CSS en plus d'apporter certaines fonctionnalités supplémentaires permettant aux développeurs d'avoir plus de flexibilité.

Attention, il n'ajoute pas de nouvelles fonctionnalités qui ne serait pas possible de faire en CSS "vanilla".

Les fonctionnalités

- **Opérateurs** : Permet de faire des calculs en CSS.
- **Variables** : Comme en programmation, permet de stocker des valeurs comme des couleurs, tailles, etc.
- **Nesting** : C'est une notion d'imbrication qui permet d'ajouter une forme "modulaire" sur l'utilisation du CSS, couplé avec une méthodologie comme **BEM**, cela facilite le travail en groupe.
- **Mixins** : Écrire des morceaux de code réutilisables.
- **Fonctions** : Permet d'écrire des fonctions qui pourront retourner une valeur.
- **Extends** : Utilisation du système d'héritage pour étendre les propriétés d'un élément.
- **Conditions** : La possibilité d'utiliser des conditions comme le : **if**, **each**, **for**, **while.**

Exemple

Opérateurs

Un "opérateur" est un signe ou un mot qui permet de réaliser une opération. Il existe des opérateurs de différents types qui permettent d'effectuer des types d'opérations différentes : opérateurs arithmétiques, opérateurs logiques, opérateurs de concaténation, etc.

Les opérateurs de concaténation Sass

Opérateur	Description
+	Retourne une chaine qui contient les deux expressions de départ concaténées
–	Retourne une chaine qui contient les deux expressions de départ concaténées et séparées par “-“
/	Retourne une chaine qui contient les deux expressions de départ concaténées et séparées par “/“

Les opérateurs arithmétiques Sass

Opérateur	Nom de l’opération associée
+	Addition
–	Soustraction
*	Multiplication
/	Division
%	Modulo (reste d’une division euclidienne)

Les opérateurs de comparaison Sass

Les opérateurs de comparaison nous permettent de comparer différentes valeurs entre elles.

Opérateur	Définition
==	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
>=	Permet de tester si une valeur est supérieure ou égale à une autre

Les opérateurs logiques Sass

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.

Parfois, nos conditions et boucles vont utiliser plusieurs opérateurs ensemble (notamment dans le cas où des opérateurs logiques sont impliqués).

Sass a un ordre de priorité assez standard. Les opérateurs vont être traités dans l'ordre suivant (du plus prioritaire ou moins prioritaire) :

1. Les opérateurs `not` et de concaténation `+, -`, et `/ ;`
2. Les opérateurs arithmétiques `*, / et %` ;
3. Les opérateurs arithmétiques `+ et -` ;
4. Les opérateurs `>, <, >=, <=` ;
5. Les opérateurs `==` et `!=` ;
6. L'opérateur `and` ;
7. L'opérateur `or`.

Variables

Une variable Sass est un conteneur pour une valeur. L'idée est de lier un nom à une valeur puis d'utiliser ensuite ce nom à la place de la valeur dans le code. Le nom utilisé sera ensuite converti automatiquement en la valeur à laquelle il est lié.

Sass vs CSS

Les variables Sass sont différentes des variables ou "propriétés personnalisées ("custom properties") CSS et s'utilisent différemment :

- Les variables Sass sont toutes compilées par Sass. Les variables CSS sont incluses dans la sortie CSS.
- Les variables CSS peuvent avoir différentes valeurs pour différents éléments, mais les variables Sass n'ont qu'une valeur à la fois.
- Les variables Sass sont impératives, ce qui signifie que si vous utilisez une variable puis modifiez sa valeur, l'utilisation antérieure restera la même. Les variables CSS sont déclaratives, ce qui signifie que si vous modifiez la valeur, cela affectera les utilisations antérieures et ultérieures.

Illustration de l'utilisation d'une variable et du code généré en CSS après compilation

SCSS Sass

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

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

CSS

```css
.alert {
  border: 1px solid rgba(198, 83, 140, 0.88);
}
```

Scope

Les variables déclarées au niveau supérieur d'une feuille de style sont **globales**. Cela signifie qu'ils peuvent être consultés n'importe où dans leur module après avoir été déclarés. Mais ce n'est pas vrai pour toutes les variables.

Celles déclarées dans des blocs (accolades en SCSS ou code en retrait dans Sass) sont généralement locales et ne sont accessibles que dans le bloc auquel elles ont été déclarées.

SCSS Sass

```scss
$global-variable: global value;

.content {
  $local-variable: local value;
  global: $global-variable;
  local: $local-variable;
}

.sidebar {
  global: $global-variable;

  // This would fail, because $local-variable isn't in scope:
  // local: $local-variable;
}
```

CSS

```css
.content {
  global: global value;
  local: local value;
}

.sidebar {
  global: global value;
}
```

Shadowing

Les variables locales peuvent même être déclarées avec le même nom qu'une variable globale. Si cela se produit, il existe en fait deux variables différentes portant le même nom : une **locale** et une **globale**.

Cela permet de garantir qu'un auteur écrivant une variable locale ne modifie pas accidentellement la valeur d'une variable globale dont il n'a même pas connaissance.

SCSS Sass

```scss
$variable: global value;

.content {
  $variable: local value;
  value: $variable;
}

.sidebar {
  value: $variable;
}
```

CSS

```css
.content {
  value: local value;
}

.sidebar {
  value: global value;
}
```

Si vous devez définir la valeur d'une variable globale à partir d'une portée locale (comme dans un mixin), vous pouvez utiliser l'indicateur `!global.`

Une déclaration de variable marquée comme `!global` sera toujours affectée à la portée globale.

SCSS Sass

```scss
$variable: first global value;

.content {
  $variable: second global value !global;
  value: $variable;
}

.sidebar {
  value: $variable;
}
```

CSS

```css
.content {
  value: second global value;
}

.sidebar {
  value: second global value;
}
```

Flow Control Scope

Les variables déclarées dans les règles de contrôle de flux ont des règles d'étendue spéciales : On ne déclare pas une variable locale.
Au lieu de cela, on attribue simplement une valeur à ces variables. Cela facilite beaucoup l'affectation conditionnelle d'une valeur à une variable ou la création d'une valeur dans le cadre d'une boucle.

SCSS Sass

```scss
$dark-theme: true !default;
$primary-color: #f8bbd0 !default;
$accent-color: #6a1b9a !default;

@if $dark-theme {
  $primary-color: darken($primary-color, 60%);
  $accent-color: lighten($accent-color, 60%);
}

.button {
  background-color: $primary-color;
  border: 1px solid $accent-color;
  border-radius: 3px;
}
```

CSS

```css
.button {
  background-color: #750c30;
  border: 1px solid #f5ebfc;
  border-radius: 3px;
}
```

Nesting

Lors de l'écriture de HTML, vous avez probablement remarqué qu'il a une hiérarchie claire imbriquée et visuelle. CSS, d'autre part, ne le fait pas.

Sass vous permettra d'imbriquer vos sélecteurs CSS d'une manière qui suit la même hiérarchie visuelle de votre HTML. Sachez que des règles trop imbriquées entraîneront une CSS trop qualifiée qui pourrait s'avérer difficile à maintenir et est généralement considérée comme une mauvaise pratique.

Dans cet esprit, voici un exemple de quelques styles typiques pour la navigation d'un site :

SCSS Sass

```scss
nav {
  ul {
    margin: 0;
    padding: 0;
    list-style: none;
  }

  li { display: inline-block; }

  a {
    display: block;
    padding: 6px 12px;
    text-decoration: none;
  }
}
```

CSS

```css
nav ul {
  margin: 0;
  padding: 0;
  list-style: none;
}
nav li {
  display: inline-block;
}
nav a {
  display: block;
  padding: 6px 12px;
  text-decoration: none;
}
```

Vous remarquerez que les sélecteurs `ul`, `li` et `a` sont imbriqués dans le sélecteur de navigation. C'est un excellent moyen d'organiser votre CSS et de le rendre plus lisible.

Pour aller plus loin, utilisé une méthodologie comme **BEM** ou **SMACSS** par exemple pour plus de modularité.

Mixins

Certaines choses en CSS sont un peu fastidieuses à écrire, en particulier avec CSS3 et les nombreux préfixes de fournisseurs qui existent.

Un mixin vous permet de créer des groupes de déclarations CSS que vous souhaitez **réutiliser** sur l'ensemble de votre site.

Vous pouvez même transmettre des valeurs pour rendre votre mixin plus flexible.

Voici un exemple de transformation.

SCSS Sass

```scss
@mixin transform($property) {
  -webkit-transform: $property;
  -ms-transform: $property;
  transform: $property;
}
.box { @include transform(rotate(30deg)); }
```

CSS

```css
.box {
  -webkit-transform: rotate(30deg);
  -ms-transform: rotate(30deg);
  transform: rotate(30deg);
}
```

Pour créer un mixin, utilisez la directive @mixin et donnez-lui un nom. Nous avons nommé notre transformation mixin.

Nous utilisons également la variable *$property* entre parenthèses afin de pouvoir transmettre une transformation de ce que nous voulons.

Après avoir créé votre mixin, vous pouvez ensuite l'utiliser comme déclaration CSS commençant par @include suivi du nom du mixin.

Functions

Les fonctions vous permettent de définir des opérations complexes sur les valeurs SassScript que vous pouvez réutiliser dans votre feuille de style. Ils facilitent l'abstraction des formules et des comportements courants de manière lisible.

Les fonctions sont définies à l'aide de la @function. Elle est écrite @function <name> (<arguments ...>) {...}.
Le nom d'une fonction peut être n'importe quel identifiant Sass.

Il ne peut contenir que des instructions universelles, ainsi que la règle @return qui indique la valeur à utiliser comme résultat de l'appel de fonction.
Les fonctions sont appelées à l'aide de la syntaxe de fonction CSS normale.

SCSS Sass

```scss
@function pow($base, $exponent) {
  $result: 1;
  @for $_ from 1 through $exponent {
    $result: $result * $base;
  }
  @return $result;
}

.sidebar {
  float: left;
  margin-left: pow(4, 3) * 1px;
}
```

CSS

```css
.sidebar {
  float: left;
  margin-left: 64px;
}
```

Extends/Inheritance

C'est l'une des fonctionnalités les plus utiles de Sass. L'utilisation de @extend vous permet de partager un ensemble de propriétés CSS d'un sélecteur à un autre. Cela aide à garder votre Sass très “clean”.

Dans notre exemple, nous allons créer une simple série de messages pour les erreurs, les avertissements et les réussites en utilisant une autre fonctionnalité qui va de pair avec les classes étendues et substituables.

Une classe d'espace réservé est un type spécial de classe qui ne s'imprime que lorsqu'elle est étendue et peut vous aider à garder votre CSS compilé propre et net.

SCSS Sass ⇒ CSS

```scss
/* This CSS will print because %message-shared is extended. */
%message-shared {
  border: 1px solid #ccc;
  padding: 10px;
  color: #333;
}

// This CSS won't print because %equal-heights is never extended.
%equal-heights {
  display: flex;
  flex-wrap: wrap;
}

.message {
  @extend %message-shared;
}

.success {
  @extend %message-shared;
  border-color: green;
}

.error {
  @extend %message-shared;
  border-color: red;
}

.warning {
  @extend %message-shared;
  border-color: yellow;
}
```

Le code ci-dessus indique à `.message`, `.success`, `.error` et `.warning` de se comporter exactement comme `%message-shared`. Cela signifie que toute classe où `%message-shared` apparaîtra, aura les propriétés suivantes :

```
border : 1px solid #ccc;
padding : 10px;
color : #333;
```

La magie opère dans le CSS généré, où chacune de ces classes obtiendra les mêmes propriétés CSS que *%message-shared*. Cela vous évite d'avoir à écrire plusieurs noms de classe sur des éléments HTML.

Vous pouvez **étendre** la plupart des sélecteurs CSS simples en plus des classes d'espace réservé dans Sass, mais l'utilisation d'espaces réservés est le moyen le plus simple de vous assurer de ne pas étendre une classe imbriquée ailleurs dans vos styles, ce qui peut entraîner des sélecteurs involontaires dans votre CSS.

Notez que le CSS en *%equal-heigts* n'est pas généré, car *%equal-heigts* n'est jamais étendu.

Conditions (Flow Control)

@If, @else

La règle @if est écrite @if <expression> {...}, et elle contrôle si son bloc est évalué ou non (y compris l'émission de styles au format CSS). L'expression renvoie généralement true ou false : si l'expression renvoie vrai, le bloc est évalué et si l'expression renvoie faux, ce n'est pas le cas.

SCSS Sass

```scss
@mixin avatar($size, $circle: false) {
  width: $size;
  height: $size;

  @if $circle {
    border-radius: $size / 2;
  }
}

.square-av { @include avatar(100px, $circle: false);
}
.circle-av { @include avatar(100px, $circle: true);
}
```

CSS

```css
.square-av {
  width: 100px;
  height: 100px;
}

.circle-av {
  width: 100px;
  height: 100px;
  border-radius: 50px;
}
```

Une règle @if peut éventuellement être suivie d'une règle @else, écrite @else {...}. Le bloc de cette règle est évalué si l'expression @if renvoie false.

SCSS Sass

```scss
$light-background: #f2ece4;
$light-text: #036;
$dark-background: #6b717f;
$dark-text: #d2e1dd;

@mixin theme-colors($light-theme: true) {
  @if $light-theme {
    background-color: $light-background;
    color: $light-text;
  } @else {
    background-color: $dark-background;
    color: $dark-text;
  }
}

.banner {
  @include theme-colors($light-theme: true);
  body.dark & {
    @include theme-colors($light-theme: false);
  }
}
```

CSS

```css
.banner {
  background-color: #f2ece4;
  color: #036;
}
body.dark .banner {
  background-color: #6b717f;
  color: #d2e1dd;
}
```

@each

La règle @each facilite l'émission de styles ou l'évaluation du code pour chaque élément d'une liste.
C'est génial pour les styles répétitifs qui n'ont que quelques variations entre eux. Il est généralement écrit @each <variable> dans <expression> {...}, où l'expression renvoie une liste.
Le bloc est évalué tour à tour pour chaque élément de la liste, qui est affecté au nom de variable donné.

SCSS Sass

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

@each $size in $sizes {
  .icon-#{$size} {
    font-size: $size;
    height: $size;
    width: $size;
  }
}
```

CSS

```css
.icon-40px {
  font-size: 40px;
  height: 40px;
  width: 40px;
}

.icon-50px {
  font-size: 50px;
  height: 50px;
  width: 50px;
}

.icon-80px {
  font-size: 80px;
  height: 80px;
  width: 80px;
}
```

Vous pouvez également utiliser @each pour parcourir chaque paire clé / valeur dans un tableau associatif, en l'écrivant @each <variable>, <variable> dans <expression> {...}. La clé est affectée au premier nom de variable et l'élément est affecté au second.

SCSS Sass

```scss
$icons: ("eye": "\f112", "start": "\f12e", "stop": "\f12f");

@each $name, $glyph in $icons {
  .icon-#{$name}:before {
    display: inline-block;
    font-family: "Icon Font";
    content: $glyph;
  }
}
```

CSS

```css
@charset "UTF-8";
.icon-eye:before {
  display: inline-block;
  font-family: "Icon Font";
  content: "";
}

.icon-start:before {
  display: inline-block;
  font-family: "Icon Font";
  content: "";
}

.icon-stop:before {
  display: inline-block;
  font-family: "Icon Font";
  content: "";
}
```

Si vous avez une liste de listes, vous pouvez utiliser @each pour attribuer automatiquement des variables à chacune des valeurs des listes internes en l'écrivant @each <variable ...> dans <expression> {...}.
C'est ce qu'on appelle la **déstructuration**, car les variables correspondent à la structure des listes internes. Chaque nom de variable est attribué à la valeur à la position correspondante dans la liste, ou nul si la liste ne contient pas suffisamment de valeurs.

SCSS Sass

```scss
$icons:
  "eye" "\f112" 12px,
  "start" "\f12e" 16px,
  "stop" "\f12f" 10px;

@each $name, $glyph, $size in $icons {
  .icon-#{$name}:before {
    display: inline-block;
    font-family: "Icon Font";
    content: $glyph;
    font-size: $size;
  }
}
```

CSS

```css
@charset "UTF-8";
.icon-eye:before {
  display: inline-block;
  font-family: "Icon Font";
  content: "";
  font-size: 12px;
}

.icon-start:before {
  display: inline-block;
  font-family: "Icon Font";
  content: "";
  font-size: 16px;
}

.icon-stop:before {
  display: inline-block;
  font-family: "Icon Font";
  content: "";
  font-size: 10px;
}
```

@for

La règle @for, écrite @for <variable> de <expression> à <expression> {...} ou @for <variable> de <expression> à <expression> {...}, compte vers le haut ou vers le bas à partir d'un nombre (le résultat de la première expression) à un autre (le résultat de la seconde) et évalue un bloc pour chaque nombre entre les deux.

Chaque numéro en cours de route est affecté au nom de variable donné.

SCSS Sass

```scss
$base-color: #036;

@for $i from 1 through 3 {
  ul:nth-child(3n + #{$i}) {
    background-color: lighten($base-color, $i * 5%);
  }
}
```

CSS

```css
ul:nth-child(3n + 1) {
  background-color: #004080;
}

ul:nth-child(3n + 2) {
  background-color: #004d99;
}

ul:nth-child(3n + 3) {
  background-color: #0059b3;
}
```

@while

La règle @while, écrite @while <expression> {...}, évalue son bloc si son expression renvoie true. Ensuite, si son expression renvoie toujours vrai, il évalue à nouveau son bloc. Cela continue jusqu'à ce que l'expression renvoie finalement `false`.

SCSS Sass

```scss
/// Divides `$value` by `$ratio` until it's below `$base`.
@function scale-below($value, $base, $ratio: 1.618) {
  @while $value > $base {
    $value: $value / $ratio;
  }
  @return $value;
}

$normal-font-size: 16px;
sup {
  font-size: scale-below(20px, 16px);
}
```

CSS

```css
sup {
  font-size: 12.36094px;
}
```

Module

Vous n'êtes pas obligé d'écrire tous vos règles Sass dans un seul fichier. Vous pouvez le diviser comme vous le souhaitez avec la règle @use.

Cette règle charge un autre fichier Sass en tant que module, ce qui signifie que vous pouvez faire référence à ses variables, **mixins** et **fonctions** dans votre fichier Sass avec un espace de noms basé sur le nom de fichier.
L'utilisation d'un fichier inclura également le CSS qu'il génère dans votre sortie compilée !

SCSS Sass

```scss
// _base.scss
$font-stack:    Helvetica, sans-serif;
$primary-color: #333;

body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

```scss
// styles.scss
@use 'base';

.inverse {
  background-color: base.$primary-color;
  color: white;
}
```

CSS

```css
body {
  font: 100% Helvetica, sans-serif;
  color: #333;
}

.inverse {
  background-color: #333;
  color: white;
}
```

Notez que nous utilisons @use **'base';** dans le fichier ***styles.scss***. Lorsque vous utilisez un fichier, vous n'avez pas besoin d'inclure l'extension de fichier. Sass est intelligent et le découvrira pour vous.

Installation

Pour exécuter Sass en global (recommandé), vous devez avoir installé Ruby sur votre système. Sur les nouvelles versions de Linux et OSX, Ruby est déjà préinstallée.

Pour l'installer avec le gestionnaire chocolatey pour windows :

```
choco install ruby
```

Pour linux sous Ubuntu/Debian :

```
$ sudo apt-get install ruby-full
```

Pour macOS avec le gestionnaire homebrew

```
$ brew install ruby
```

Vous pouvez sinon à partir du Github, télécharger le dossier Dart Sass et l'importer directement dans votre projet, et grâce à une ligne de commande, lancer la compilation du/des fichier(s) en direction d'un **résultat**. Lancer un **watcher** (veilleur) qui vérifiera chaque changement apporté au fichier ***.scss***

Soit vous l'installez en globale, et vous pouvez vous en servir pour tous vos projets.
Soit vous l'installer en locale, mais vous devrez le réinstaller pour un nouveau projet.

Exemple de commande qui permet de lancer sass à partir du dossier dart-sass

```
./dart-sass/sass sass/style.scss style.css
```

La première partie correspond au fait qu'on se place dans le dossier dart-sass contenant le fichier qui permet de lancer des lignes de commandes.

La deuxième partie indique que Sass doit compiler dans le dossier `sass` le fichier `sass style.scss` en fichier `style.css` (**résultat**). Regardez bien les extensions.

Pour éviter d'avoir à retaper la commande à chaque changement sur le fichier **.scss** afin de compiler, vous avez la possibilité d'établir ce qu'on appelle un "watcher", qui va vérifier chaque changement à la volée.

```
./dart-sass/sass sass/style.scss style.css --watch
```

Installer n'importe où (autonome)

Je vous recommande de regarder la documentation officielle, pour être sûr d'être à jour.

Avec le gestionnaire de packages npm (fonctionne sur tous les systèmes) :

```
npm install -g sass
```

Avec chocolatey (windows)

```
choco install sass
```

Homebrew (macOS)

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

Pour l'exécuter

En ligne de commande :

```
sass --watch input.scss output.css
```

Donc le `input.scss` c'est le fichier qu'on modifie et que Sass va se charger de compiler en fichier `output.css`

Pour prendre le input dans un dossier spécifique et le mettre dans un autre dossier, nous pouvons utiliser cette commande

```
sass --watch app/sass:public/stylesheets
```

On remarque que l'extension n'est pas spécifié ici, Sass comprend automatiquement que la première partie avant le **:** correspond au fichier à compiler, et la deuxième partie au fichier transformé (output).

IDE

Les IDE moderne intègrent souvent des "addons" capable de faire tourner sass sans ligne de commande :

- PhpStorm
- Visual studio code
- Atom
- Autres...

Task Runner

Une autre possibilité souvent utilisée en entreprise, c'est l'utilisation des "task runner". Ces outils permettent d'automatiser des tâches comme la compilation de fichier `.scss` en `.css`

- Webpack
- Gulp
- Grunt

L'inconvénient c'est que la première configuration de ce genre d'outil est **difficile** et **longue** lorsqu'on est débutant, surtout pour webpack. Je recommande vivement de d'abord commencé par l'utilisation d'une ligne de commande **ou** avec un addon d'IDE.

À savoir

L'extension `.SASS` correspond à la syntaxe originale de Sass. Elle fonctionne sans `;` et `{}` uniquement à l'indentation, elle est néanmoins difficile à relire lors de gros projet.

Du coup une autre syntax est apparue, celle qui correspond à l'extension `.scss` qui signifie **Sassy CSS** elle a été adopté par la grande majorité des développeurs car on y retrouve la syntaxe de base comme en CSS avec les avantages du Sass.

Sass ou Less

SASS et LESS, les deux permettent l'utilisation de variables. La différence clé entre SASS et LESS, est que SASS est basé sur Ruby, alors que LESS utilise JavaScript.

Même cela ne donne à aucun des pré-processeurs un avantage sur l'autre. SASS est beaucoup plus populaire.

Mais cela pourrait être dû au fait que SASS est un peu plus âgé. À l'origine, LESS était pris en charge par le framework front-end réputé Bootstrap, qui reposait sur le préprocesseur le plus récent. Mais avec la version 4, le projet a officiellement basculé vers SASS, ce qui a encore renforcé la popularité de SASS. (Source)

Pour en savoir plus…

Voici donc quelques liens utile :

- https://sass-lang.com/documentation/at-rules/css
- https://www.alsacreations.com/article/lire/1717-les-preprocesseurs-css-c-est-sensass.html
- https://waytolearnx.com/2019/04/difference-entre-sass-et-less.html
- https://www.youtube.com/watch?v=MOstrhqpIsI
- https://fr.wikipedia.org/wiki/Pr%C3%A9processeur_CSS