ESGI 4IW et 5IW 2023-2024

Edouard Sombié

Partie 1 : Sass

- Présentation
- Dart-sass (node.js)
- Sass vs scss
- Imbrication
- Variables
- Inclusion et import
- Boucles et conditions
- Fonctions
- Approche BEM
- Mise en pratique

D'autres pré-processeurs css

- less
- stylus
- postCSS

Présentation

Sass est un **pré-processeur css** permettant de compiler des feuilles de styles à partir de variables et de fonctions. Il permet également l'inclusion de fichiers (*partials*), rendant les projets d'intégration plus faciles à maintenir et à faire évoluer. Fini les "recherche et remplace" dans des fichiers css sans fin...
https://sass-lang.com/

La compilation peut être faite via des applications autonomes (ex. Codekit), mais aussi grâce à des bibliothèques JavaScript, Ruby, Swift ou Java.

Dans le cadre de ce cours, nous utiliserons la version js du compilateur qui tourne sur Node.js : le **package sass**, distribution de dart-sass.
https://www.npmjs.com/package/sass

Dart Sass (Node.js)

Installation globale (recommandée) :
```
npm install -g sass
```

Installation dans un projet :
```
npm install --save-dev sass
```

Exécution via une commande sass :
```
sass --watch [--style=compressed] input.scss output.css
```

Exécution via le fichier package.json (recommandée)
```
npm run watch
```

```
{
  "scripts": {
    "watch": "sass --watch input.scss output.css"
    }
}
```

Note :
sass, en tant qu'implémentation de dart-sass, lui ajoute une API js en plus du CLI natif.
https://sass-lang.com/documentation/js-api/

Sass vs scss

Les fichiers de compilation sass peuvent être écrits selon deux syntaxes différentes :

Sass "natif" (.sass)
Syntactically **A**wesome **S**tyle**S**heets

➔ Syntaxe basée sur l'indentation.

```
.button
  display: block
  padding: 1em
```

Scss (.scss)
Sassy **C**ascading **S**tyle**S**heets

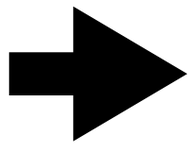
➔ Syntaxe identique au css
(accolades, point-virgule)

```
.button{
  display: block;
  padding: 1em;
}
```

Bien entendu, nous utiliserons la **syntaxe scss** qui nous permettra d'inclure des bibliothèques déjà compilées ou des composants déjà écrits en css ! 😎

Imbrication (nesting)

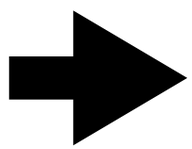
```scss
body{
    margin: 0;
    h1{
        text-transform: uppercase;
    }
}
```



```scss
body{
    margin: 0;
}
body h1{
    text-transform: uppercase;
}
```

& "le sélecteur de parent"

```scss
ul{
    display: flex;
    aside &{
        flex-direction: column;
    }
}
```



```scss
ul{
    display: flex;
}
aside ul{
    flex-direction: column;
}
```

A noter :
L'imbrication est possible aujourd'hui en css mais avec des limitations.
Les sous sélecteurs doivent commencer par un caractère spécial : . # : [* + > ~ &
A éviter pour le moment pour garantir un bon support navigateur.

Source: https://caniuse.com/css-nesting

Trick 🦊 :
Contourner les limitations du sous sélecteur grâce à **&** et **:is()**

```css
body{
    margin: 0;
    & h1{
        text-transform: uppercase;
    }
}
```

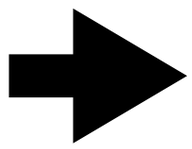
```css
ul{
    display: flex;
    :is(aside) &{
        flex-direction: column;
    }
}
```

Mais aucun intégrateur digne de ce nom n'écrit du css non compilé... 🤓🤔

Variables

```scss
$base-color: #c6538c;
$border-dark: rgba($base-color, 0.88);
$space: 10px;

.alert {
  border: 1px solid $border-dark;
  padding : 3px + $space;
}
```



```scss
.alert {
  border: 1px solid rgba(#c6538c, 0.88);
  padding : 13px;
}
```

On assigne une valeur par défaut à une variable avec le drapeau **!default**

```scss
$main-color : black !default;
```

Note :
Dans les noms de variables, - et _ sont traités de manière identique.
$ma_variable == $ma-variable 🤔

Portée des variables
Les variables déclarées au premier niveau de la feuille de style sont globales.
Celles déclarées dans un sélecteur (entre accolades) sont locales.

Une variable globale existante est modifiée depuis un scope local avec **!global**

```scss
$main-color : red !global;
```

Types de variables	Modules associés

Number
Deux composants : le nombre lui-même et l'unité
`5.2e3, 16px`
`$value-with-unit : $number * 1px`
	sass:math
	`math.div(100px, 2s)`
	`math.ceil(4.9);`

String
Avec ou sans guillemets
`"Roboto", bold`
	sass:string
	`string.length("Helvetica")`
	`string.unique-id()`

Color
Hexa, rgba, hsla, mot-clé
`red, #777`
	sass:color
	`color.blackness(#e1d7d2)`
	`color.scale(#036, $lightness: -30%)`

List
Tableau de valeurs
`Helvetica, Arial, sans-serif`
`10px 15px 0 0`
`[line1 line2]`
	sass:list
	`list.append(10px 20px, 30px)`
	`list.nth(10px 12px 16px, 2)`

Map
Tableau de clés : valeurs
`("reg": 400, "med": 500, "bold": 700)`
	sass:map
	`map.deep-remove($weights, "reg")`
	`map.get($weights, "medium")`

Calculation Fonctions css de calcul `calc(), min(), max(), clamp()`	

Function
Fonction avec valeur de retour
`@function rem($val){`
`  @return calc($val / 16 * 1rem);`
`}`	

Boolean	
Null	

Concaténation de variables

Pour concaténer la valeur d'une variable (interpolation), on utilise **#{ }**
Cela est utile lorsque l'on veut concaténer une valeur dans une chaîne de caractères.

```
--bg-color : #{$secondary-color};
.col-#{$i} {...}
```

Attention !
Lors de la concaténation, les guillemets d'une valuer string sont supprimés.
Pour éviter cela, il faut envelopper la valeur dans une fonction **meta.inspect()**

```
@use "sass:meta";

$font-family-monospace: Menlo, Consolas, "Courier New", monospace;

:root {
  --font-family-monospace: #{meta.inspect($font-family-monospace)};
}
```

Note : il a fallu ici importer le module sass:meta avec @use. Nous allons voir cela bientôt...

Opérations sur des variables

Deux variables ayant la même unité peuvent être additionnées ou soustraites l'une à l'autre;

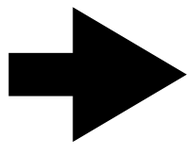
```scss
$padding-base: 1rem;

.btn {
  padding : $padding-base + 0.5rem;
}
```

Les variables peuvent également être multipliées par une valeur.
Par contre, pour une division, il faut utiliser **calc()** ou **math.div()** ;

```scss
@use "sass:math";
$space: 2rem;

.button{
    margin: $space * 2;
    padding: math.div($space, 2);
    line-height: calc($space / 2);
    font-size: calc(#{$space} / 2);
}
```



```scss
.button {
  margin: 4rem;
  padding: 1rem;
  line-height: 1rem;
  font-size: calc(2rem / 2);
}
```

Noms de variables dynamiques

Pour faire référence à un nom de variable en fonction d'une autre variable, on peut, comme dans d'autres langages, passer par un tableau de variables.

```
@use "sass:map";

$theme-colors: (
  "success": #28a745,
  "info": #17a2b8,
  "warning": #ffc107,
);

.warning {
  background-color: map.get($theme-colors, "warning");
}
```

Variables sass vs variables css

Les variables sass sont très différentes des **variables css**, dans la mesure où elles ne sont pas accessibles en *front-end,* elles disparaissent lors de la compilation.
Elles sont utilisées dans le code lorsqu'il est nécessaire de leur appliquer des opérations (transparentize, darken, fonction custom...).

Pour que certaines valeurs soient accessibles du côté front end, on injecte des variables sass dans des variables css, ce qui facilite la création de composants variables (dark/light par exemple).
Se reposer sur des valeurs contenues dans des variables css, appelées aussi "propriétés personnalisées", est une **très bonne pratique**.

```scss
@use "sass:color";
$pink : #ff2896;

:root{
  --pink: #{$pink};
  --pink-light: color.scale( $pink, $lightness: 30%);
}

a{
  color: var(--pink);
  &:hover{
    color: var(--pink-light);
  }
}
```

Les variables css permettent notamment de changer le comportement d'un élément en fonction de son **contexte**.

```scss
@use "sass:color";
$pink : #ff2896;
$blue: #1e293a;

:root{
  --linkColor: #{$pink};
  --linkColor-light: color.scale( $pink, $lightness: 30%);
}

.footer{
  --linkColor: #{$blue};
  --linkColor-light: color.scale( $blue, $lightness: 30%);
}

a{
  color: var(--linkColor);
  &:hover{
    color: var(--linkColor-light);
  }
}
```

Import et inclusion

@import
Cette règle permet d'importer des fichiers scss externes, des *partials* (préfixés d'un underscore) ou des feuilles de styles css.

```
@import "variables"; // fait référence au fichier _variables.scss
@import "vendor/bootstrap-min";
```

Notes :

- Pas besoin de spécifier ./ pour les chemins relatifs.
- La règle **@import est en principe destinée au premier niveau** d'une feuille de style mais peut également être appelée dans un sélecteur.

```
.btn { @import "btn-rules"}
```

Attention ! 🖐
Bien que très utile dans le cadre d'un framework simple articulé autour d'un ficher *master*, **l'utilisation de cette règle est aujourd'hui déconseillée** à cause de certains problèmes qu'elle peut engendrer :

- Un *partial* est importé à chaque appel de la règle @import
- Les variables, fonctions et *mixins* qu'il contient sont globaux, ce qui peut être source de conflit et oblige souvent à préfixer les sélecteurs d'une bibliothèque.

La solution ? **@use** 😎

@use

Comme @import, cette règle permet d'importer des fichiers scss externes, des *partials* (préfixés d'un underscore) ou des feuilles de styles css. Les sources incluses par @use sont appelées **modules**.

```
@use 'foundation/code'; _code.scss
@use "vendor/bootstrap-min";
```

Notes :

- La règle @use doit être appelée **avant toute règle de style**.
- A la grande différence de @import, les variables, fonctions et *mixins* d'un fichier chargé par @use appartiennent à un ***namespace*** (identique au nom du fichier).

```
// src/_borders.scss

$radius: 3px;
```

```
// _card.scss

@use "src/borders";
.card{
    border-radius : borders.radius;
}
```

Namespace *custom*

```
// _card.scss

@use "src/borders" as b;
.card{
    border-radius : b.radius;
}
```

```
// _card.scss

@use "src/borders" as *;
.card{
    border-radius : radius;
}
```

Modules sass natifs (cf page sur les types de variables)

`sass:color, sass:list, sass:map, sass:math, sass:meta, sass:selector, sass:string`

```scss
@use "sass:map";
@media screen and (min-width: #{map.get($breakpoints, lg)}) {...}
```

Configuration de module

Lors de l'importation d'un module, il est possible d'en modifier la configuration, c'est à dire redéfinir les variables et sélecteurs qu'il contient. On utilise l'instruction **with**.

```scss
// _library.scss
$black: #000 !default;
$padding: 1rem !default;

@use "library" with (
  $black: #222,
  $padding: 2rem
}
```

Variables privés

Pour limiter la visibilité d'une variable à un module, il faut la préfixer avec - ou _

`$-radius: 3px;`

@forward
Cette instruction très utile permet de charger les variables, fonctions et mixins d'une feuille de style A vers une feuille de style B qui seront disponibles à l'usage de cette mêle feuille de style B

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

// styleB.scss
@forward "styleA";

// styles.scss
@use "styleB";

li {
  @include styleB.styleA-list-reset;
}
```

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

// stylesheetB.scss
@forward "stylesheetA" as s-*;

// styles.scss
@use "stylesheetB";

li {
  @include stylesheetB.s-list-reset;
}
```

Avec un préfixe custom

Lors d'un @forward, on peut masquer des membres :
`@forward "src/list" hide list-reset, $horizontal-list-gap;`

Réutilisation de code (*snippets*)

@mixin

Les mixins sont une des fonctionnalités les plus puissantes de sass. Ils permettent l'inclusion de blocs de code css grâce à l'instruction **@include**.

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

ul {
    @include resetList;
}
```

```
@mixin resetList($padding) {
    list-style: none;
    margin: 0;
    padding: $padding;
}

ul {
    @include resetList;
}
```

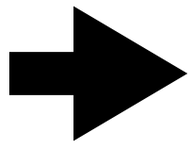
Les mixins peuvent recevoir des **paramètres**.

@extend 😨

Cette instruction, assez proche de @include permet d'étendre une classe ou un placeholder. **Son utilisation est fortement déconseillée**, au profit des mixins, car elle perturbe la structure de la feuille de style et crée des liens entre sélecteurs qui peuvent générer de effets indésirables.

```scss
.no-border {
    border: none;
}

.maClasse {
    font-size: 1rem;
    @extend .no-border;
}
```



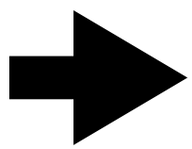
```scss
.no-border, .maClasse {
  border: none;
}

.maClasse {
  font-size: 1rem;
}
```

Avec un **placeholder %**, le sélecteur de base n'est pas compilé.
Mais ça reste pas terrible du tout !

```scss
%no-border {
    border: none;
}

.maClasse {
    font-size: 1rem;
    @extend %no-border;
}
```



```scss
.maClasse {
  border: none;
}

.maClasse {
  font-size: 1rem;
}
```

Boucles et conditions

Sass offre les traditionnelles boucles **@for** et **@while**.

-
```
@for $i from 1 through 6 {
    .p-#{$i} {
        padding : $i * 0.5rem;
    }
}
```
// la dernière itération est incluse

-
```
@for $i from 1 to 7 {
    .p-#{$i} {
        padding : $i * 0.5rem;
    }
}
```
// la dernière itération est exclue

-
```
@while $value > $base {
  $value: math.div($value, $ratio);
}
@return $value;
```
// **Cette boucle est à éviter** si possible,
au profit de @for ou @each car elle augmente le temps de compilation et repose sur des valeurs true/false assez strictes (0 != false par exemple)

La boucle **@each** pour parcourir listes et maps

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

-
```
@each $name, $size in $breakpoints {       // map
    .container-#{name} {
        width: $size;
    }
}
```

Le cas d'une liste de listes (**destructuration**)

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

@each $name, $glyph, $size in $icons {
  .icon-#{$name}:before {
    content: $glyph;
    font-size: $size;
  }
}
```

Les conditions

Elles peuvent être évaluées avec les règles suivantes :

- `@if{}`
- `@else{}`
- `@else if{}`

Que du classique...

```
@if $light-theme {
    background-color: $light-background;
    color: $light-text;
  } @else {
    background-color: $dark-background;
    color: $dark-text;
  }
```

Fonctions

Sass propose de nombreuses fonctions natives permettant de modifier des valeurs ou d'évaluer des variables.

Certaines sont globales :
```
darken($red, 30%);
```

D'autres font partie de modules sass natifs :
```
color.mix($color1, $color2, $weight: 50%);
```

Liste complète : https://gist.github.com/AllThingsSmitty/3bcc79da563df756be46

@function
On peut également créer des **fonctions *custom*** qui traiteront une valeur d'entrée afin de renvoyer un résultat.

```
@use "sass:math";
@function rem($pxVal) {
    @return math.div($pxVal, 16) * 1rem;
}
```

L'approche BEM

Même si elle ne date pas d'hier, cette méthode est considérée comme une bonne pratique (pas que par moi...), elle repose sur les notions de :

- **Block**
  entité autonome
  `.card` ou `.navigation`

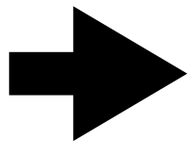
- **Element**
  partie d'un block, y faisant référence directement
  `.card__title` ou `.navigation__item`

- **Modifier (modificateur)**
  "drapeau" appliqué à un block ou un élément permettant de modifier son apparence ou on comportement
  `.card--dark` ou `.navigation--vertical`

Elle permet de créer un cadre rigide de nommage et de s'affranchir des problèmes causés par les styles en cascade (générère une forme de scope local des règles).

Element

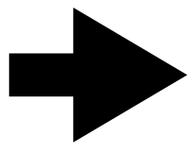
```scss
.card{
    display: flex;
    flex-direction: column;
    &__title{
        font-size: 1.2rem;
    }
    &__image{
        object-fit: cover;
    }
}
```



```scss
.card{
    display: flex;
    flex-direction: column;
}
.card__title{
    font-size: 1.2rem;
}
.card__image{
    object-fit: cover;
}
```

Modificateur

```scss
.btn{
    font-size: 1rem;
    padding: 1em;
    &--sm{
        font-size: 0.7rem;
    }
    &--lg{
        font-size: 2rem;
    }
}
```



```scss
.btn{
    font-size: 1rem;
    padding: 1em;
}
.btn--sm{
    font-size: 0.7rem;
}
.btn--lg{
    font-size: 2rem;
}
```

Le cas du "petit-enfant"

L'approche BEM permet une écriture simple des composants et une lecture claire du code css généré. Toutefois, l'imbrication de classes peut amener à des noms à rallonge du type `.navigation__item__link__icon`

Voici ce qu'en dit le créateur de la méthode :

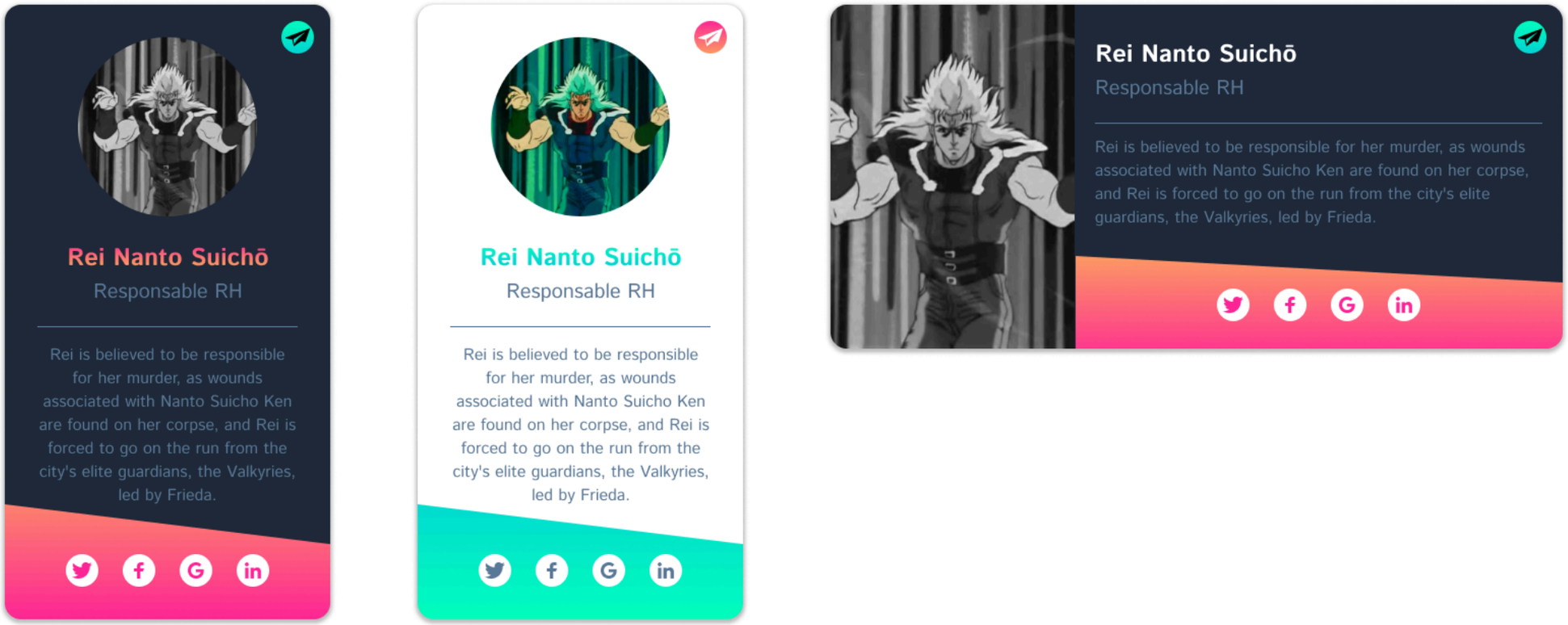
> “BEM methodology doesn’t recommend to use elements within elements in class names. You don’t need to resemble DOM structure in naming. Having one level structure makes refactoring much easier.”
> — Vladimir Grinenko, Yandex

```
.navigation{
    /* */
    &__item{
        /* */
        &__link{
            /* */
            &__icon{
                /* */
            }
        }
    }
}
```

```
.navigation{
    /* */
    &__item{
        /* */
    }
    &__link{
        /* */
    }
    &__icon{
        /* */
    }
}
```

Mise en pratique

C'est le moment d'appliquer ce que l'on vient de voir : https://www.figma.com/file/HV3Sc7JoEMGcR0gCkvUcU3/TP1-sass?type=design&node-id=0-1&mode=design&t=KLZHxAeuaRUg8npC-0



Pour les consignes et le chemin à suivre, rendez-vous sur myGES, dans la rubrique supports de cours : TP1-sass 🤟

D'autres pré-processeurs

Bien entendu, sass n'est pas le seul pré-processeur css utilisé aujourd'hui, bien qu'étant le plus populaire. Voici trois de ses "concurrents"... histoire de soigner sa culture générale.

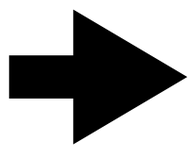
https://lesscss.org/

Leaner CSS
Mixins, variables, fonctions, bibliothèques, très proche de sass mais moins utilisé...

```
> lessc styles.less styles.css
```

```less
@bg: black;
@bg-light: boolean(luma(@bg) > 50%);

div {
  background: @bg;
  color: if(@bg-light, black, white);
}
```



```css
div {
  background: black;
  color: white;
}
```

D'autres pré-processeurs

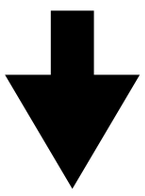
https://stylus-lang.com/

Syntaxe très (trop ?) souple, gestion des prefixes, commandes CLI très complètes, plugins...

```
> stylus --compress < some.styl > some.css
```

```
font-size = 14px
font-stack = "Lucida Grande", Arial, sans-serif

body
  font font-size font-stack
```

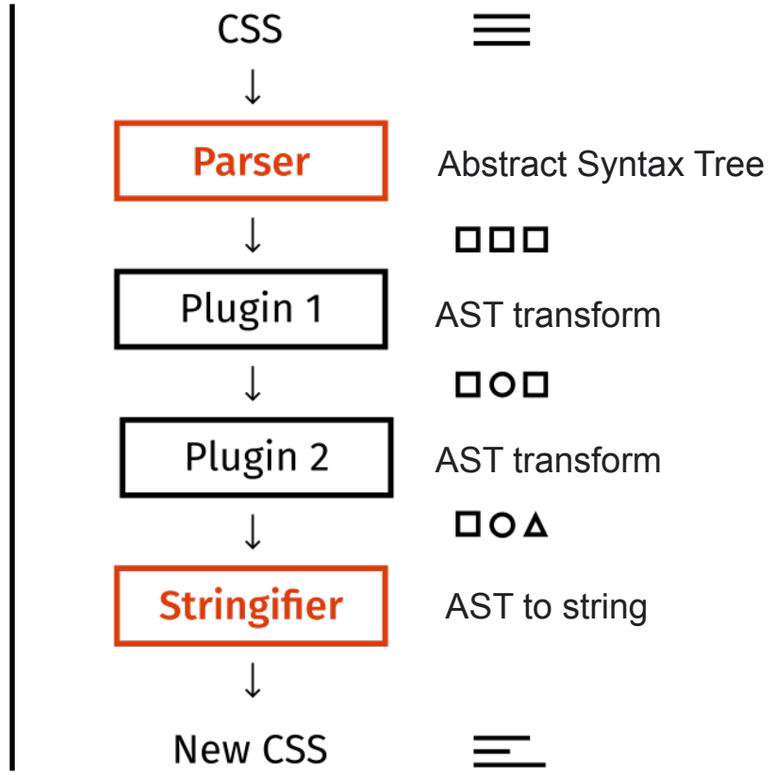


```
body {
  font: 14px "Lucida Grande", Arial, sans-serif;
}
```

D'autres pré-processeurs

https://postcss.org/

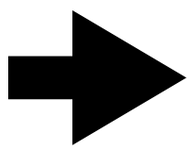
Pas réellement un pré-processeur comme sass ou less, plutôt un **outil js de manipulation de la syntaxe css**.
Input css, autoprefixer, modules css, plugins...
Le pré-processeur sur lequel repose **Tailwind css** 👌




```
> postcss --use autoprefixer -o main.css css/*.css
```

```
.cfx {
  @util clearfix;
}

.rounded-top {
  @util border-top-radius(4px);
}
```



```
.cfx:after {
  content: '';
  display: block;
  clear: both;
}

.rounded-top {
  border-top-left-radius: 4px;
  border-top-right-radius: 4px;
}
```

Maintenant que l'on a vu l'outil qui permet de créer des frameworks css, remontons d'un niveau et intéressons-nous à un framework déjà compilé que l'on pourra éditer à l'aide de postCSS.

Tailwind css

- Présentation
- Créer un projet Tailwind css
- Mise en pratique

...

Présentation

Framework css "bas-niveau" basé sur des **utility classes**, créé en 2017 par Adam Wathan. La version actuelle est 3.3.3

```
text-sm, rounded-lg, pt-2
```

Bien que "bas-niveau", ce framework propose des composants prêts à l'emploi ainsi que des classe *helpers* permettant la création de composants *custom*.
Il repose sur un fichier de configuration js et est extensible via des plugins.

Comme vu précédemment, ce framework est basé sur **postCSS**. Il peut toutefois être utilisé avec d'autres pré-processeurs en tant que plugin postCSS, même si cet usage est déconseillé...

Il est clairement le framework css le plus en vogue actuellement de par sa vitesse de mise en œuvre, sa souplesse et la lisibilité qu'il offre (parfois...).
Il est certainement une très bonne solution pour la création rapide d'une interface et apparaît (à mon humble avis) comme un outil complémentaire de sass qui, pour sa part, offre des fonctionnalités plus puissantes mais certains écueils aussi (beaucoup à écrire, nommage...).

https://tailwindcss.com/

SASS

Les +

- Markup html plutôt léger et lisible
- Fonctions, mixins, conditions et boucles 💪
- Imbrication de règles
- On peut tout faire ! Une approche BEM ou des utility classes, ou les deux !

Les -

- Tout le css doit être écrit *from scratch*
- Règles de nommage vite confuses : composant agnostique ou pas ? utilitaire ou pas ? 🤔
- On peut tout faire, donc aussi du "pas fou fou"
- Markup et style se font séparément (la séparation des préoccupations - *separation of concerns* - est un mythe)
- Comme avec du css traditionnel, on peut se retrouver à batailler contre des règles globales (!important... mouais...)

Tailwind css

Les +

- Beaucoup de helpers et composants disponibles *out of the box*
- Utility classes propices à une intégration cohérente
- Css plus compact : pas toujours nécessaire d'ajouter du code *custom*
- Pas de souci de contexte : le style est dans le html
- Pas de souci de nommage si on se repose uniquement sur des utility classes
- Pas d'effet de bord lors des changements sur un composant
- L'intégration peut être plus rapide
- Plus de styles en cascades... parfois source de bugs ou d'ajout de *!important* inopportuns
- Se combine très bien aux composants des frameworks tels que React, Vue ou Svelte 🤩

Les -

- Markup html vite **très touffu**
- Utility classes peuvent engendrer un manque de cohérence entre les composants...
- Nécessaire de se familiariser avec les utiity classes avant d'être productif
- Plus de styles en cascades... parfois utiles 😅

Vous avez dit "touffu" le html ?

Save

```
<button type="submit" class="rounded-md bg-indigo-600 px-3
py-2 text-sm font-semibold text-white shadow-sm hover:bg-
indigo-500 focus-visible:outline focus-visible:outline-2
focus-visible:outline-offset-2 focus-visible:outline-
indigo-600">Save</button>
```

Tailwind css

```
<button type="submit" class="btn btn--indigo">Save</button>
```

BEM :)

Il ne va pas falloir avoir peur de faire des copier/coller. 😛
Et j'espère qu'on ne va pas me demander de modifier le border-radius des boutons de mon site... 😭

Tailwind css propose bien une règle **@apply**, mais pour le coup, l'utiliser est vraiment un non-sens... On en reparlera.

Surtout **Tailwind css est fait pour être utilisé avec des framework *front-end* basés sur des composants autonomes** ! React, Vue, Svelte ou des moteurs comme Blade...

Mais un css optimisé !

Grâce à une compilation *Just in Time* (**JIT**), **seules les classes utilisées sont écrites dans le fichier compilé.** En effet, lors de la compilation via postCSS, Tailwind css passe en revue toutes les classes contenues dans les templates déclarés dans le fichier de configuration (paramètre *content*) et ne conserve que celles-ci !

Tailwind offre également la possibilité d'importer une base de feuille de styles reposant sur *modern-normalize*.

Enfin, Tailwind peut facilement minifier la sortie css grâce au flag `--minify`

Et ne perdons pas de vue qu'on n'aura **quasiment pas de css à écrire** !!

Créer un projet Tailwind css

Installation (via nde.js)

```
> npm install -D tailwindcss
> npx tailwindcss init
```

Déclaration des fichiers de template (dans le fichier tailwind.config.js)

```
module.exports = {
  content: ["./src/**/*.{html,js}"],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

Ajout des *directives* au fichier source

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

Ecouter les modifications du fichier source

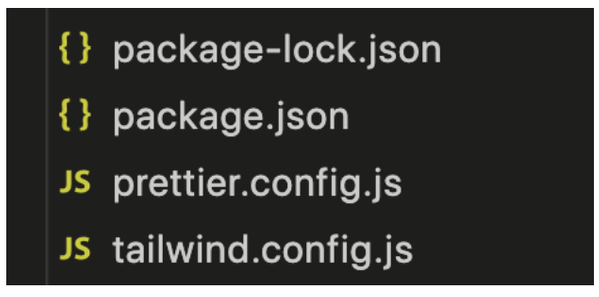
```
> npx tailwindcss -i ./src/input.css -o ./dist/output.css --watch
```

Des outils utiles, voire indispensables

On va commencer par utiliser Visual Studio Code...
C'est là que l'on retrouve les extensions (officielles) les plus utiles :

- **Tailwind CSS IntelliSense**
  autocomplete, linting, hover preview
  https://marketplace.visualstudio.com/items?itemName=bradlc.vscode-tailwindcss

- **Prettier + prettier-plugin-tailwindcss**
  Auto-format html/css + tri des classes tailwind*
  https://marketplace.visualstudio.com/items?itemName=esbenp.prettier-vscode
  https://github.com/tailwindlabs/prettier-plugin-tailwindcss

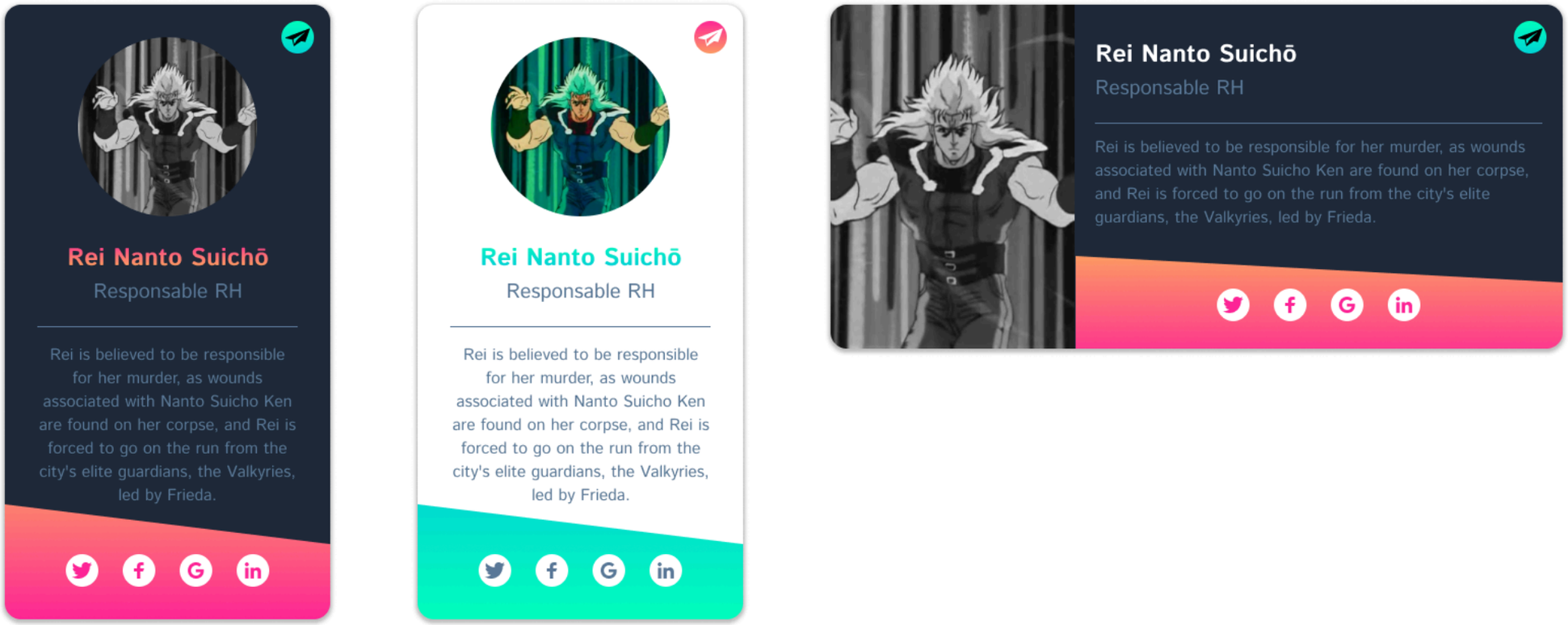


*Etant donné le nombre importants de classes Tailwind css sur un élément, il faut suivre l'ordre de classes préconisé par tailwind css. https://tailwindcss.com/blog/automatic-class-sorting-with-prettier#how-classes-are-sorted

Sans oublier la liste des classes : **https://tailwindcomponents.com/cheatsheet/**

Mise en pratique

C'est le moment d'utiliser le framework pour recréer notre composant du TP1 !
https://www.figma.com/file/HV3Sc7JoEMGcR0gCkvUcU3/TP1-sass?type=design&node-id=0-1&mode=design&t=KLZHxAeuaRUg8npC-0



Pour les consignes et le chemin à suivre, rendez-vous sur myGES, dans la rubrique supports de cours : TP2-tailwind 🤘

Bibliographie

Sass
https://sass-lang.com/

Nesting CSS
https://webkit.org/blog/13813/try-css-nesting-today-in-safari-technology-preview/

BEM
https://getbem.com/introduction/