La méthode consiste à prendre le design comme point de départ...

Bonnes pratiques en CSS : BEM et OOCSS

Source :
https://www.alsacreations.com/article/lire/1641-Bonnes-pratiques-en-CSS--BEM-et-OOCSS.html

https://www.alticreation.com/bem-pour-le-css/

Le concept de OOCSS est de repérer des « objets CSS », c'est-à-dire des « patterns visuels » qui se répètent, et de définir ainsi des classes réutilisables.

La méthode consiste à prendre le design comme point de départ : on repère des répétitions visuelles puis on les nomme.

La sémantique du document n'est donc plus une base de travail,
et des classes CSS nommées selon l'apparence sont autorisées à partir du moment où elles sont génériques.

En cela, OOCSS est en décalage avec les bonnes pratiques des années 2000

OOCSS met en avant deux principes :

1. Le principe de séparation de la structure et de l'apparence
2. Le principe de séparation du conteneur et du contenu.

```
<button class="small-btn"></button>
<button class="large-btn"></button>
```

DEVIENT

```
<button class="btn small-btn"></button>
<button class="btn large-btn"></button>
```

OOCSS, pourquoi c'est bien ?

Les 10 bonnes pratiques

Voici la liste des 10 bonnes pratiques qui forment l'esprit de l'OOCSS :

1. **Créer une bibliothèque par composant**
   Chaque composant (bouton, tableau, lien, image, clearfix, etc.) doit être tel un Lego = combinable et réutilisable à souhait.

2. **Utiliser des styles sémantiques et consistants**
   Le style d'un nouvel élément créé en HTML doit être prévisible.

3. **Designer des modules transparents**
   Le module est le cadre conteneur pouvant être utilisé avec n'importe quel contenu.

4. **Être flexible**
   Les hauteurs et largeurs doivent être extensibles et s'adapter (RWD inside).

5. **Apprendre à aimer les grilles**
   Les grilles permettent de contrôler la largeur, la hauteur se gère par le contenu.

6. **Minimiser les sélecteurs**
   S'en tenir à une spécificité `[0-0-1-0]` permet un meilleur contrôle sur les sélecteurs.

7. **Séparer la structure de l'apparence**
   Il faut distinguer les deux, c'est un principe fondamental de l'OOCSS. Elaborez des objets abstraits pour la structure des blocs et des classes pour habiller ces blocs, indépendamment de leur nature.

8. **Séparer le conteneur du contenu**
   Il faut distinguer les deux, c'est un principe fondamental de l'OOCSS. Construisez des **relations 1:n** en décomposant conteneur et contenu.

9. **Étendre les objets en appliquant de multiples classes aux éléments** Les classes/objets sont autant de Legos qui s'assemblent pour obtenir le résultat désiré.

10. **Utiliser les resets et les fonts YUI**
    C'est un choix relatif du framework OOCSS.

Cela implique que le pilotage de l'apparence se fait depuis le code HTML. Ainsi, lorsqu'une feuille de styles écrite à la manière OOCSS est bien faite,

il devient possible d'ajouter des morceaux entiers dans le design sans toucher à la feuille CSS, juste en réutilisant des objets CSS déjà existants.

Object oriented CSS

Le second principe consiste à éviter des cascades CSS comme `.links-box .title`
car l'apparence du contenu `.title` serait alors couplée au conteneur .links-box.
Une classe `.box-title` sera plus réutilisable.

https://www.nicoespeon.com/fr/2013/05/plongee-au-coeur-de-oocss/

https://www.slideshare.net/stubbornella/our-best-practices-are-killing-us

BEM

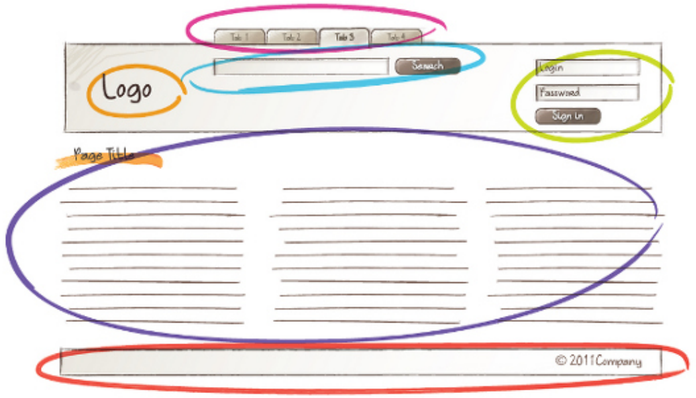
Block, Element, Modifier,

La méthodologie BEM est une solution élaborée par Yandex et publiée en 2010.

BEM a deux faces : il s'agit d'abord d'une méthode déclarative de l'interface utilisateur servant à décrire un « arbre BEM », les outils open source de Yandex travaillent ensuite sur cet arbre.

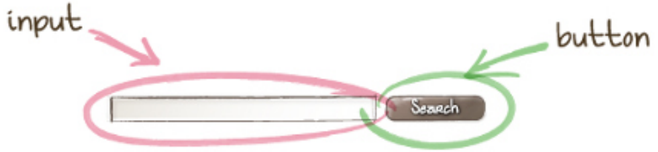
BEM apporte également une convention de nommage des classes CSS qui a gagné une certaine popularité.

Un bloc est une entité indépendante, une « brique » de l'application ou de la page Web. Un bloc forme son propre contexte autonome. Ci-dessous des exemples de blocs dans une illustration tirée du site officiel :



Ce qui compose un page ou une application Web peut toujours être rangé dans une arborescence de blocs, d'éléments et de modificateurs.

Un élément est une partie d'un bloc. Le contexte d'un élément est celui du bloc. Ci-dessous deux exemples empruntés toujours au site officiel :



BEM envisage la composition d'une page web en deux types de «composants» :

Le bloc est un composant «parent» contenant un ou plusieurs éléments. Le bloc peut être indépendant, et lorsque pris hors contexte d'une page spécifique, garde du sens.
par exemple : un menu, un pied de page (footer), un menu latéral (sidebar).

L'élément est un composant appartenant à un bloc. Il faut considérer un élément comme l'enfant d'un bloc.
par exemple : le titre d'un bloc, une page d'un menu.

Enfin BEM introduit le concept de modificateur.

Le **modificateur** va introduire une notion de comportement. Il se modifie en fonction du contexte de la page ou d'une action de l'utilisateur. Le modificateur peut aussi bien être appliqué à un bloc qu'à un élément.

par exemple : un fond de couleur différente pou une page spécifique, un élémen rendu visible (ou caché) après un clic de l'utilisateur.

Ces règles impliquent de préfixer les noms des éléments par leur contexte. Venons-en à la convention de nommage des classes CSS.

Le site officiel prend soin de préciser que seuls comptent les concepts, la syntaxe restant libre. L'équipe de BEM utilise pour sa part une syntaxe à base de underscores :

- block-name
- block-name_modifier_name
- block-name__element-name
- block-name__element-name_modifier_name

```css
.search-box {
  height: 300px;
  width: 300px;
}
.search-box_light {
  background-color: #DEF;
  color: #777;
}
.search-box__btn {
  padding: 4px;
}
.search-box__btn_max_visible {
  font-weight: bold;
}
```

.e code HTML :

```html
<div class="search-box search-box_light">
  <!-- (input field here) -->
  <button class="search-box__btn search-box__btn_max_visible">Search</button>
</div>
```

Jne cascade est utilisée lorsqu'un modificateur de bloc a un effet sur un élément :

```css
.search-box_light .search-box__btn {
  background-color: #9AB;
}
```

BEM réduit les risques de conflits de nommage

Un des problèmes que l'on rencontre rapidement en rédigeant du CSS, c'est de se retrouver avec des noms de classes similaires.

La méthodologie BEM impose un nommage des classes qui élimine quasiment les risques de conflits CSS.

- sans BEM :

```
/* titre de la page nommée simplement .titre /*
.titre {
     font-size: 15px;
     color: red;
}

/* titre d'un article dans la page également appelé .titre voici donc un conflit, deux classes qui se superposent ! */

.titre {
     font-size: 30px;
     color: blue;
}
```

- avec BEM :

```
/* titre de la page nommée .page__titre /*
.page__titre {
     font-size: 15px;
     color: red;
}

/* titre d'un article dans la page appelé .article__titre . On a éliminé le conflit en nommant suivant une convention nos éléments. */
.article__titre {
     font-size: 30px;
     color: blue;
}
```

BEM réduit les risques de conflits liés à la spécificité

La spécificité en CSS est une logique épineuse avec laquelle il vaut mieux éviter de s'amuser.

Un exemple de conflit de spécificité sans BEM :

Un exemple de conflit de spécificité sans BEM :

```
/* le CSS */
h3 { color: green; }
h3#main { color: orange; }
h3.navigation { color: yellow;}
```

```
/* le HTML*/
<h3 class="navigation" id="main">Titre de niveau trois</h3>
```

À votre avis de quel couleur sera le titre h3 ?
Si vous avez de bonnes notions de CSS, vous le savez déjà. Dans tous les cas, si on avait appliqué la méthodologie BEM, nous ne nous poserions même pas la question.

Résolution du conflit de spécifité avec BEM :

```
/* le CSS */
.title-h3 { color: green; }
.main__title-h3 { color: orange; }
.navigation__title-h3 { color: yellow;}
```

```
/* le HTML*/
<h3 class="title-h3  navigation__title-h3">Titre de niveau trois</h3>
```

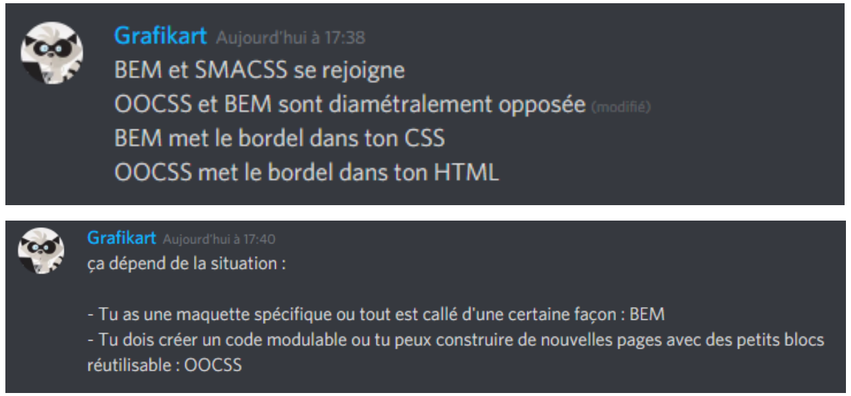
Le BEM proscrit l'utilisation de l'attribut `id`. On vient donc de résoudre le conflit majeur de l'exemple. Enfin j'ai choisi de surcharger le style par défaut du titre h3 appelé `.title-h3` avec la classe `.navigation__title-h3`. (Si des déclarations CSS entre en conflits au sein de 2 ou plusieurs classes, ce sera la dernière classe déclarée qui appliquera ses déclarations.)

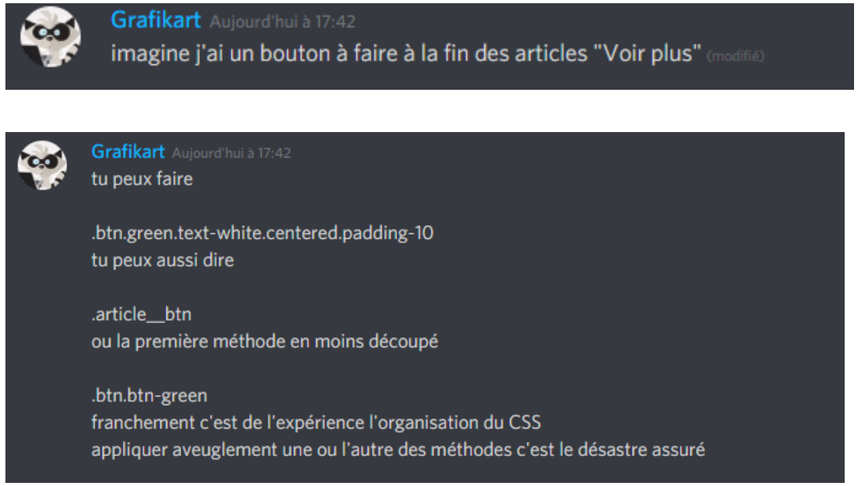
BEM est extrêmement modulaire et réutilisable :

En résolvant les deux conflits vus ci-dessus, BEM impose une méthodologie qui rend la rédaction de CSS plus rigide mais aussi plus sûre. Il est facile d'utiliser les différentes classes produites dans toutes les pages sans se soucier des conflits.

On envisage alors le code comme un assemblage de briques réutilisables, modulaires et extensibles. Il est même possible de réutiliser certaines briques, d'un projet à un autre, sans rencontrer de conflits.

Bonus :


Grafikart Aujourd'hui à 17:38
BEM et SMACSS se rejoigne
OOCSS et BEM sont diamétralement opposée (modifié)
BEM met le bordel dans ton CSS
OOCSS met le bordel dans ton HTML
Grafikart Aujourd'hui à 17:40
ça dépend de la situation :
- Tu as une maquette spécifique ou tout est callé d'une certaine façon : BEM
- Tu dois créer un code modulable ou tu peux construire de nouvelles pages avec des petits blocs réutilisable : OOCSS


Grafikart Aujourd'hui à 17:42
imagine j'ai un bouton à faire à la fin des articles "Voir plus" (modifié)
Grafikart Aujourd'hui à 17:42
tu peux faire
.btn.green.text-white.centered.padding-10
tu peux aussi dire
.article__btn
ou la première méthode en moins découpé
.btn.btn-green
franchement c'est de l'expérience l'organisation du CSS
appliquer aveuglement une ou l'autre des méthodes c'est le désastre assuré