**Cours sur CSS**

1. **Les Bases**
2. **Declaration**

Il y a 03 manières de déclarer du css :

* **In-line :** Ici on insert directement du code css dans du html

<p style="background-color: red;">Du css In-line</p>

Cette façon alourdir le code, rend le code moins lisible, pas très optimise pour l’accessibilité du genre SEO (search Engine Optimisation) : permet d’améliorer le positionnement d’un site web sur les pages de résultats des moteurs de recherche (SERP), on l’appelle aussi **référencement naturel.**

* **Avec balise « Style » :** Ici on peut introduire du css dans des balise **styles**

<style>

    p{

*background-color*: red;

    }

</style>

Cette façon comme la précédente rend le code moins lisible, tous les problèmes de maintenances.

* **Créer un fichier :**  On crée un fichier qui va contenir tous nos codes CSS. Ensuite on vient le linker dans le **head**

<link rel="stylesheet" href="style.css">

1. **ID et Classe CSS**

Un élément html peut contenir plusieurs **classes** mais qu’ un seul **ID**

<p class="monPara monPara2" id="monId">Je suis un P.</p>

**Exple :**

Dans le code CSS :

* Classe : **.monPara{….}**
* Id : **#monId{….}**

1. **Taille en CSS**

Il existe 02 types d’unités :

* Unité **absolue (pixel, etc..)**
* Unité **relative (pourcentages, etc…) au Body**

**NB :** De base le **body**  n’a pas de **hauteur(height),** donc si on veut utiliser une **unité relative** alors on doit mettre un hauteur.

Cependant ici nous voyons que lorsque **width et height** sont en **unité relatives**, il faut automatiquement défini le width et height du body. Auquel cas on n’obtiendra rien.

Ici nous voyons que notre **body**  a une hauteur**(width)** de 250px mais pas de largeur**(height)**.

La **Box**  quant a elle a une  **width relative**  a celle du body, donc **250px/2 = 125px** . On définit aussi la taille de la **hauteur de la Box(100px)**  sinon elle ne pourra pas s’afficher vu que body n’en possède pas.

**NB :** lorsqu’ on veut centrer un conteneur au centre sur **Y**  on doit lui donne une **height (**généralement **100% de son parent**)

**Sans hauteur** le conteneur prend de la hauteur en fonction des éléments qui seront présents dans ce conteneur.

1. **Margin et Padding**

* Margin : C’est la marge externe.
* Padding : C’est la marge interne.
* **Propriété du Margin et Padding :**

Margin/Padding : **haut droite bas gauche**

**NB :** Centre horizontalement un élément html

* margin: **0 auto ;**
* **0** c’est pour le **haut et bas**
* **auto** c est pout **droite et gauche**

**Remarque:** on ne peut pas **centrer Verticalement** car de base il n’y a pas d’espace de **haut en bas**, mais seulement de **gauche à droite**

* box-sizing : border-box ; : C’est la propriété qui permet de prendre en compte les valeurs des Margin et de Padding pour un élément enfin de respecter sa taille d’origine.
* \*{ box-sizing : border-box ; } : grâce au sélecteur universel **\*** on peut appliquer cette propriété a tous les éléments de notre html.

1. **Propriété Display**

* **Display :block ->**  Met seulement un élément sur une ligne à la fois. C’est à dire prend tous les éléments de la ligne. On peut leur mettre une **width**  et une  **height.**
* **Exple:** <div>, <p>, <form>, etc…
* **Display : inline ->**  Met les éléments à la suite des autres sur une ligne. C’est à dire prend seulement les dimensions de son contenu sur la ligne indiquée. On **ne peut pas les mettre un width et height**.
* **Exple:** <span>, <img>, etc…
* **Display : inline-block ->** Permet de mettre les éléments à la suite des autres, mais aussi de les ajouter une **width**  et une  **height**.
* **Display : none ->**  permet de faire disparaitre un élément a l’affichage sur le navigateur, mais il reste présent dans le **DOM**.

1. **Image Responsive**

* **width: 100% ;**
* **height : auto ;**

**width** prendra toujours la **width du body**  et **height** s’adaptera automatiquement.

* **Background-size : cover ; ->** permet d’étirer l’image de couverture pour la rendre responsive.
* **Rendre une image étirable et responsive :**

**NB:**  Ici **height: 100vh**  signifie qu’ on va prendre **100% de la hauteur de notre écran**. Cette opération nous ajoute des marges , c’est donc pour cette raison qu’ on les retire avec **margin : 0**  et **padding : 0**

**NB :**

* **Lorsqu’ un élément parent n’a pas de marge extérieure (margin), c’est celui de son enfant qui est pris en compte.**
* En CSS c’est le dernier élément stylise qui est pris en compte.

1. **Les Spécificités**

* **!important**  est la spécificité ultime ultime.

**Exple :** p{ color : red **!important** ;}

* **Inline-css**  est la spécificité ultime
* **Id** ont une spécificité supérieure aux autres
* **Classe** viennent ensuite
* **Sélecteur**  est inferieur a la classe

1. **Héritage**

* Certains styles du parent sont transmis aux enfants

**<div>** est le parent , **<h1>** et deuxième **<p>** sont des enfants. Mais le **premier <p>** est un enfant de <h1> et le **deuxième <p>** est un voisin ou sibling de <h1>

1. **Position**

* Tous les éléments de base en CSS ont la position statique
* Lorsqu’ on change la position d’un élément, on dit que cette élément est  **positionne**  c’est à dire qu’ il a accès a 4 propriétés :
* **Top**
* **Botton**
* **Left**
* **right**
* Comme position nous avons :
* **position: absolute ;** fait sortie l’élément de son flux(**changer sa position initiale**), mais reste relatif à son parent positionné.**(enfant)**
* S’il est en absolut et que son parent n’est pas positionné, alors il sera relatif au parent le plus proche positionne
* S’il est en absolut et que son parent est positionné, alors il sera relatif à son parent.
* Pour **centrer une élément en position absolute :**

**top: 50% ;**

**left: 50% ;**

**transform: translate (-50%, -50%) ;**

* **position: relative ;** fait sortie l’élément de son flux(**garde sa position initiale**), c.-à-d. il va passer devant les autres **relatif à son parent. (parent)**
* **position: fixed ;** il va rester fixe à l’écran, même pendant le scroll, il va scroller aussi
* Les couches de positionnement se fait avec : **z-index : 1 ;** Ce nombre (**1**) représente le niveau de positionnement. **Plus le nombre est grand et plus l’élément sera au-dessus des autres.**

1. **Les Sélecteurs**

* **\*{..} :**  sélecteur **universel** qui a une spécificité **basse.**
  + On pour appliquer une propriété a tous les éléments et généralement pour ça :

\* , ::before, ::after{

*box-sizing*: border-box;

}

* **.parent .enfant {…} :** ici on cible seulement tous les éléments ayant une **classe enfant**  et qui se trouve dans un élément ayant une **classe parent**.
* **selecteur1, selecteur2 :**  on cible à la fois tous les éléments du selecteur1 et selecteur2. Ici **,**  joue le rôle de **ET**
* **.class1**(.parent) **> .class2**(enfants)**:**  on va cibler tous les éléments qui ont la classe **class2** et qui sont enfant directs des éléments qui ont la classe **class1** parente**.**
* **selecteur1**(.parent) **+ selecteur2**(.adjacent)**:** on cible le **selecteur2** qui suit directement le **selecteur1**  au même niveau (**sibling**).
* **selecteur1 ~ selecteur2 :** on cible tous les **selecteur2** qui suivent les **selecteur1.**
* **[attribut=’’valeur’’]{…} :** permet d’utiliser l’attribut d’un élément html comme : **title, src, alt, etc…**

**Exple :**

[title="nom"]{

*background-color*: red;

}

* **input:checked{…} :**  pour les **checkbox**

c’est une  **pseudo classe(la pseudo classe**  représente une action, un état de nos éléments)

* **classe ::first-letter{…} : ::first-letter** est une pseudo classe qui permet de mettre la première lettre en majuscule
* **classe ::first-line{…} : ::first-line** est une pseudo classe qui permet de mettre la première ligne en majuscule
* **classe\_enfant : nth-child(numero de l’enfant){…}:** permet de sélectionner un enfant parmi une liste d’enfant, il doit être imbrique dans un parent. Il suffit de fourni  **le numero de la position de l’enfant. Exple : .item :nth-child(2)**
* **classe\_enfant:not( :nth-cild(numero de l’enfant){…} :** applique tous les styles aux autres enfant , mais pas l’enfant indique par **:nth-child(:nth-cild(numero de l’enfant)**
* **Cas des liste ordonnée ou non-ordonnée :**
* **classe:first-of-type{…} :** sélectionne seulement le  **premier item**  de la liste
* **classe:last-of-type{…} :** sélectionne seulement le  **dernier item**  de la liste

1. **Unité de mesure**

On distingue :

* **Les unités relative :**  qui sont relative au parent, donc **variable.**
* **Pourcentage(%) :**  50%, 20%, etc…
* **Viewportwidth(vw) :** relatif à la **largeur de la fenêtre(page web)**
* **Viewportheigth(vh) :** relatif a la  **hauteur de la fenêtre.**
* **EM :** relatif au text, celui du parent. Mais si le parent n’a pas de taille pour le text, alors il va se référer à la taille **d’élément root**  qui est de **16px(taille de base des éléments html).**
* **REM :** relatif au text de  **l’élément root** qui est de **16px.**

**NB : élément root**  s’écrit ainsi : **html{….}**

* **Les unité absolues :**  qui ne change pas, donc **constant.**
* **Pixel (px) :** 12px, 20px, etc…

1. **Pseudo classes**

* **Active ( classe :active{…} ) :** c’est lorsqu’ on clique sur un élément html que les propriétés sont actives.
* **Focus( classe :focus{…} ) :** C est plus pour les  **élément input.**  Les propriétés sont actives dès qu’ on clique sur l’élément.
* **Hover( classe :hover{…} ) :** les propriétés sont appliquées dès le survol du curseur sur l’élément.
* **input:checked{…} :**  pour les **checkbox**

c’est une  **pseudo classe(la pseudo classe**  représente une action, un état de nos éléments)

* **a :visited{…} :**  applique les propriétés aux élément **lien** lorsqu’ ils ont été cliques.

1. **Pseudo élément**

Il permet de :

* Modifier le contenu d’un élément. **Par Exple** : **la premier lettre ou la première ligne.**
* Rajouter du contenu a un élément sans rajouter d’autres nœuds html. **Par Exple** : je voudrais ajouter du contenu après mon **<h1>** je vais donc pas créer une **<div>**  a la suit pour l’ajouter, je peux cependant utiliser un **pseudo élément ::after**

On peut avoir comme pseudo élément :

* **classe ::first-letter{…} : ::first-letter** est une pseudo classe qui permet de mettre la première lettre en majuscule
* **classe ::first-line{…} : ::first-line** est une pseudo classe qui permet de mettre la première ligne en majuscule
* **classe ::before{content : ‘’ mettre un contenu ’’}** : ajouter le contenu de  **content** avant cette classe/sélecteur
* **classe ::after{content : ‘’ mettre un contenu ’’}** : ajouter le contenu de  **content** après cette classe/sélecteur

**NB :** les **::before et ::after**  ont un **display : inline.**  Mais on peut les rendre en **display : block,**  et cela nous permet donc d’ajouter une **width** , une **height**  et certaines **propriétés css. content**  doit tjr être là.

1. **Float et bug des Floats**

* **float: left/right** -> permet de faire sortie un élément de son **flux** comme **absolut, relative**
* Le **bug des Floats** c’est lorsqu’un élément parent ayant une **width** mais pas de **height** par exemple contient 02 éléments enfants et que ces enfants ont un **float** **positionné**, alors l’élément parent **disparait ( ses propriétés)**

<div class="parent">

        <div class="enfant1"></div>

        <div class="enfant2"></div>

    </div>

***Solution***

* La solution c’est de spécifier qu’ il y a encore un élément après à l’aide de la **propriété clear :both ;**
* De mettre un **pseudo élément ::after**

.parent::after{

*content*: "";

*display*: block;

*clear*: both;

}

* **box-shadow : posX, posY, flou, rgba(rouge, bleu, vert, opacité)**

box-shadow : 0px 15px 50px rgba(0, 0, 0,0.2)

* **shape-outside : circle() ;**  permet de créer des bord en forme de cercle

1. **Animation et transition**

* Propriété **Transform** :

**NB** : Le transforme de fonctionne pas sur les éléments de type **Inline**.

* Propriété **Transition :**
* Propriété **Animation** :
  + Définition de l’animation :
  + Création de cette animation :

1. **Responsive**

* Media Query
* On peut rendre un **text responsive avec le ViewPort :**

**font-size: 8vw;**

1. **FlexBox**

* **Propriétés**

**NB :** Ces propriétés sont appliquées aux parents et qui vont agir sur leurs enfants.

* **justify-content :** aligne les éléments horizontalement.
  + - **flex-start :** Les éléments s'alignent au côté gauche du conteneur.
    - **flex-end :** Les éléments s'alignent au côté droit du conteneur.
    - **center :** Les éléments s'alignent au centre du conteneur.
    - **space-between :** Les éléments s'affichent avec un espace égal entre eux.
    - **space-around :** Les éléments s'affichent avec un espacement égal à l'entour d'eux.
* **align-items :** aligne les éléments verticalement
* **flex-start :** Les éléments s'alignent au haut du conteneur.
* **flex-end :** Les éléments s'alignent au bas du conteneur.
* **center :** Les éléments s'alignent au centre vertical du conteneur.
* **baseline :** Les éléments s'alignent à la ligne de base du conteneur.
* **stretch (**par default**):** Les éléments sont étirés pour s'adapter au conteneur.

**NB :** lorsqu’ on veut centrer un conteneur au centre sur **Y**  on doit lui donne une **height (**généralement **100% de son parent**)

**Sans hauteur** le conteneur prend de la hauteur en fonction des éléments qui seront présents dans ce conteneur.

* **flex-direction :**  Cette propriété CSS définit la direction dans laquelle les éléments sont placés dans le conteneur.
* **row(**par default**) :** Les éléments sont disposés dans la même direction que le texte.
* **row-reverse :** Les éléments sont disposés dans la direction opposée au texte. **Renverse la lecture sur la ligne.**
* **column :** Les éléments sont disposés de haut en bas.
* **column-reverse :** Les éléments sont disposés de bas en haut. **Renverse la lecture sur la colonne**

**NB :** Remarquez que quand la **direction est en colonne**, justify-content change pour la verticale et align-items pour l'horizontale.

* Parfois, inverser l'ordre de la rangée ou la colonne ne suffit pas. Dans ces cas, on peut appliquer **la propriété order** à des éléments individuels. Par défaut, les éléments ont une valeur de 0, mais on peut utiliser cette propriété pour changer la valeur à un entier positif ou négatif.

Order 0

Order -1

Order 1

* **align-self** : propriété a appliqué à un élément individuel, mais cette fois ci accepte les valeurs de la propriété  **align-items** et s’applique seulement à l’élément cible.
* **flex-wrap** : permet de distribuer les éléments en modifiant leurs tailles
* **nowrap (**par default**)** : Tous les éléments sont tenus sur une seule ligne.
* **wrap** : Les éléments s'enveloppent sur plusieurs lignes au besoin.
* **wrap-reverse** : Les éléments s'enveloppent sur plusieurs lignes dans l'ordre inversé.
* **flex-flow**: row wrap , C’est la combinaison entre **flex-direction (row)**  et  **flex-wrap(wrap)**
* **align-content :** détermine l'espace entre les lignes
* **flex-start** : Les lignes sont amassées dans le haut du conteneur.
* **flex-end**: Les lignes sont amassées dans le bas du conteneur.
* **center** : Les lignes sont amassées dans le centre vertical du conteneur.
* **space-between** : Les lignes s'affichent avec un espace égal entre eux.
* **space-around** : Les lignes s'affichent avec un espace égal autour d'eux.
* **Stretch(**par default**)** : Les lignes sont étirées pour s'adapter au conteneur.
* **Gestion des espaces :**
* **flex-grow:<int>** = propriété qui permet de faire l’agrandissement sur l’espace restant en fonction de l’entier

**Exple : flex-grow: 2** , agrandissement de 2 fois plus.

* **flex-shrink:<int>** = propriété qui permet de faire le rétrécissement en fonction de l’entier

**Exple : flex-shrink :0 ,** ne fais aucun rétrécissement

* **flex-basis: <length>** = permet de dire que notre élément prend une **width** de base initialement.

**Exple : flex-basis : 200px //** l’élément aura une **width** de 200px

* **flex : <grow> <shrink> <basis>** = permet de d agrandir, rétrécir
* **flex : auto** // = grow = 1 shrink = 1 basis = 0

Cette propriété s utilise sur **des flex-items**

1. **Grid CSS**

* **Propriétés**
* **grid-column-start : <int>** , Définir la position de début d’un élément de la grille sur les colonnes de la grille.
  + **span <int> :** définir la largeur de la colonne désirée
* **grid-column-end :<int> ,** Définir la position de fin d’un élément de la grille sur les colonnes de la grille.
  + **span <int> :** définir la largeur de la colonne désirée
* **grid-column :<grid-column-start>/<grid-column-end> :** Définir la position d’un élément de la grille sur les colonnes de la grille.
* **Grid-column-gap : <int>,**  permet de créer les gouttières ou espacement entre les colonnes.
* **grid-row-start :<int>,**  Définir la position de début d’un élément de la grille sur les lignes de la grille.
* **grid-row : <grid-row-start>/<grid-row-end>,** Définir la position d’un élément de la grille sur les lignes de la grille.
* **grid-row-gap : <int> ,** Permet de créer une gouttière ou un espacement entre les lignes
* **grid-gap :<int>,** permet de mettre de l’espacement entre les lignes et les colonnes
* **grid-area :<grid-row-start>/<grid-column-start>/<grid-row-end>/<grid-column-end> ,** Définir la taille et la position d’un élément dans la grille.
* **order: <int>(**default 0**)** , Définir l’ordre des éléments dans la grille.
* **grid-template-columns : <length>** ou **<percentage>** ou **<flex>** ou **<fractionnaire(fr)>** ou **max-content** ou **min-content** ou **minmax(min, max),** Définir le dimensionnement et le nom des colonnes de la grille.

**Exple : grid-template-columns: [nom1]50px [nom2]50px;** ici nous avons nommé nos colonnes

* + **repeat(nbre de fois, length) :**  permet de répéter un nbre de fois identique la taille de nos colonnes ou lignes.
  + **max-content**  prend la width maximale du contenu
  + **min-content** prend la width du plus grand mot et coupe les autres mots

**NB :** Lorsque les colonnes sont définies avec des **pixels**, des **pourcentages** ou des **ems**, toutes les autres colonnes définies avec **fr** se répartiront l'espace restant

* **grid-template-rows: <length>** ou **<percentage>** ou **<flex>** ou **<fractionnaire(fr)>** ou **max-content** ou **min-content** ou **minmax(min, max),** Définir le dimensionnement et le nom des colonnes de la grille.

**Exple : grid-template-rows: [nom1]50px [nom2]50px;** ici nous avons nommé nos lignes

* + **repeat(nbre de fois, length) :**  permet de répéter un nbre de fois identique la taille de nos colonnes ou lignes.
* **grid-template : <grid-template-rows>/<grid-template-columns> ,** Définir les dimensions et les noms des lignes et des colonnes de la grille
* **grid-template-areas : <nom sommets> <nom footers>**

*grid-template-areas*: "head1 head2 head3"

                     "foot1 foot2 foot3"

**Exple :**

Ici on crée 03 colonnes de 2 lignes

* + Pour utilise il suffit juste de faire : **grid-area: head2;**
* **Propriété sur le positionnement des éléments  :**
* **justify-items : <center | end | start >** permet de positionner les éléments sur l’axe des X
* **align-items : <center | end | start >** permet de positionner les éléments sur l’axe des Y
* **Propriété sur le positionnement d’un éléments à la fois :**
* **justify-items : <center | end | start >** permet de positionner un seul élément à la fois sur l’axe des X
* **align-self : <center | end | start >**  permet de positionner un seul élément à la fois sur l’axe Y
* **Propriété sur le positionnement du contenu :**
* **justify-content : <center | end | space-between| space-around| start >**  permet de positionner le contenu de notre grille sur l’axe X **.**
* **align-content : <center | end | space-between| space-around| start >**  permet de positionner le contenue de notre grille sur l’axe Y.

**NB :** lorsqu’ on veut centrer un conteneur au centre sur **Y**  on doit lui donne une **height (**généralement **100% de son parent**)

**Sans hauteur** le conteneur prend de la hauteur en fonction des éléments qui seront présents dans ce conteneur.

* **Grille implicite**

Ce sont des grilles qui apparaissent seul avec des tailles différente de celle qu’ on a défini. Pour les corrige on fait ainsi :

* **grid-auto-rows :<taille>**  ceci permet de donner une taille a ces grilles implicite et elles seront sur forme de ligne.
* **Grid-auto-flow : <taille>** ceci permet de donner une taille a ces grille implicites et elles seront sous forme de colonne.
* **Faire du responsive :**
* **auto-fill :**  permet de créer un **nbre** de grille qui sera fonction de la **width\_contenu (divise par) width\_1\_grille = nbre**
* **auto-fit :** permet de créer une grille qui s’adaptera au dimensions **width\_contenu (divise par) width\_1\_grille**

*grid-template-columns*: repeat(*auto-fit*, minmax(150px, 1fr));

1. **Vendors**

Ce sont des préfixe css qui permettent aux propriétés css d’être compatible avec tous les navigateurs.

Nous avons comme préfixe :

* **-o- :** pour Opera
* **-moz- :** Firefox
* **-webkit- :** Chrome, Safari, Android
* **-ms- :** Microsoft Internet Explorer
* <https://caniuse.com/> Site pour connaitre les propriétés supportées par les navigateurs.

Ici nous voyons que nous avons une propriété **filter** sera moins compatible avec le navigateur  **Microsoft IE (-ms-),**  c’est pour cette raison qu on ajoute ce vendor : **-ms-filter : …..**  Cette dernier va seulement gérer les navigateurs Microsoft et les autres seront gérés normalement.

* **Fusion des marges :**  C’est un problème qui apparait généralement lorsque 02 bloc se suivent directement et utilise les margin adjacents c’est à dire : **box1**  utilise  **margin-Bottom**  et **box2**  utilise **margin-top** donc a se moment c est celui qui aura la  **marge la plus grande dont sa marge sera prise en compte.**
* Pour retirer cette fusion de marge : il faut ajouter un contenu entre nos 02 boxes. Par exemple on peut utiliser : un **padding, display : grid, display : flex** ou bien un

border : 1px solid transparent

* **Projet final**
* Lorsqu’ on télécharge un repository sur GitHub, on ouvre le **package.json** au niveau des dépendance , on peut installer les librairie nécessaires.

Ici on voit qu’ on a juste une dépendance : **Sass**

* Installer le **package.json** avec : **npm init** renseigner les infos.
* Installer le **préprocesseur sass : npm install sass**
* Regarder maintenant toutes les changement dans le fichier **style.scss** et compiler le tout dans le fichier **style.css :**

**Sass --watch Styles/style.scss Styles/style.css**

**sass --watch** **scss**:**css**générera un fichier appelé **style.css** dans le répertoire **css,** donc notre fichier **Sass**  se trouve dans le repertoire **scss**

1. **SASS**

**Markup** est parfois utilisé comme une forme abrégée pour dire HTML, qui signifie **Hyper Text Markup Language.**

* **BEM** est l’acronyme de **bloc, élément, modificateur**
* **Un bloc** est un composant, ou une section d’une page qui est autonome et peut fonctionner indépendamment du reste de la page. Cela peut être **un header, un footer, un conteneur, un menu ou même un bouton**. Le but est simple : **vous pourriez supprimer tout ce qu’il y a autour de votre bloc et vous pourriez toujours comprendre de quoi il s’agit.**

On nomme un bloc en décrivant sa **fonction(**ce que fait le bloc, la description générale du bloc**)**

**Exple : .proj-prev**

* Le nom d’un **élément** doit indiquer deux choses :
  + son **bloc parent**, suivi d’un **double underscore** (aussi appelé “**dunders**”) ;
  + la **fonction** de l’élément (ce que fait cet élément)

**Exple : .proj-prev\_\_heading**

* **Les modificateurs** modifient l’apparence d’un bloc mais aussi d’un élément. Voyez-les comme des sélecteurs qui créent différentes versions d’un bloc ou d’un élément. Besoin de changer la **taille**, la **couleur**, la **police**, etc. d’un élément, tout en le gardant tel quel ? C’est un modificateur qu’il vous faut !.

**Pour nommer le modificateur**: vous devez préciser le **bloc** (ou **l’élément**) qu’il modifie, y ajouter **deux tirets** suivi du **style graphique** de votre modificateur

**Exple :** **.proj-prev--mint** , cette classe va modifier le bloc en changeant sa couleur de fond en vert.

**Les préprocesseurs** sont des outils, ou des programmes, qui vont vous permettre de générer du CSS à partir des fichiers écrits dans la syntaxe du préprocesseur que vous avez choisi.

**Le nesting** est le fait d’imbriquer les sélecteurs l’un dans l’autre en créant une hiérarchie, exactement comme en HTML.

Exemple de code SCSS avec **nesting** a gauche et code CSS a droite

* **les fonctionnalités des préprocesseurs :**
* **Les variables :** Les variables vous permettent de stocker des valeurs répétées fréquemment, comme les **couleurs** et les **mensurations**, dans un élément unique que vous pouvez réutiliser à travers l’ensemble de votre code.

Elle se définie comme suit : **$nomVariable : valeur ;**

Ici nous avons définis une variable : **$mint : #15DEA5 ;** qui sera appelé plus tard pour définir la couleur du Background.

**NB :** Nommer la variable en fonction de **son rôle** plutôt que son contenu

**Exple :**  **$color-primary : valeur ;**

* **Les @mixin :**

Ils fonctionnent comme des variables a la seule différence qu’ elles peuvent stocker non pas seulement des **valeurs** mais aussi des **propriétés css et de bloc de codes.**

* Pour l’utiliser dans nos bloc, nous faisons comme suit :

**@include nom\_mixin ;**

* **@mixin avec argument :**

On peut définir une @mixin avec un argument sans valeur par default ou un argument ayant une valeur par default comme suit :

**@mixin mixin-name($arg : default\_value, $propriete\_css){**

**css-property: $arg;**

**#{$propriete\_css}: $arg;**

**}**

**#{$propriete\_css}:** correspond au **nom de la propriété css. Exple : Background, color, font-size, etc…**

* Pour l’utiliser on fait comme suit :

**@include nom\_mixin($variable) ;**

* **Extensions :**

Les extensions fonctionnent comme des mixin a la seul différence qu’ ils ne font pas la répétition de code après compilation sur le css.

* La declaration se fait comme suit :

**.nom\_selecteur ou .extension{**

**propriété-css : value ;**

**}**

* Utilisation se fait comme suit :

**@extend . nom\_selecteur ou . extension ;**

**NB :** Mixin duplique un ensemble de règles alors que @extend duplique le sélecteur.

En utilisant un le **préfixe pourcentage(%)** devant votre sélecteur plutôt que le **point** qu’on utilise d’habitude pour les classes, Sass crée un **placeholder.**

De cette façon il n’apparaitra plus dans la liste de nos éléments( confère cadre en rouge ci-dessus).

* **Fonction :**

Ce sont des bouts de code préfabriqués qui effectuent des tâches, comme prendre un argument, le modifier et renvoyer la nouvelle valeur sans que vous n’ayez à le faire vous-même.

* **Couleurs :**

On peut exprimer les couleurs en :

* **RGB(**couleur **Rouge**, couleur **Verte,** couleur **Blue) ou en Hexadécimal**

**Exple : #15DEA5 🡪 RGB( 21, 222, 165)**  car 15 (hex) = 21,

DE = 222 et A5 = 165

* **HSL :**

Qui est l’abréviation de **hue**, **saturation** et **lightness** (teinte, saturation et clarté). Alors que la fonction **rgb()** se rapporte aux proportions de **rouge**, **vert** et **bleu** dans une couleur, la fonction **hsl()** mesure où se situe la couleur dans le nuancier et à quel point elle est vive et claire :

* **Les boucles :** Les boucles, qui automatisent les tâches répétitives telles que la **création d’une série de modificateurs de couleurs**, par exemple, vous épargnent un vrai calvaire tout en gardant une codebase plus petite et plus simple à gérer.

* **Boucle for** : **@for()**

**Exple : @for**($var **from** 1 **through** n)**{**

**.item#{**$var**}**{ // On accède à la propriété css : .item1 , .item2, …….

font-size :30px +$var\*10 ;

**}**

* **#{$var}** ceci correspond à une **interpolation en Sass.** C’est utiliser quand on veut afficher une valeur sous forme de chaine de caractère( un peu comme avec le **print** dans python)
* **Les structures conditionnelles (ou conditions) :**
* **@if ( condition ) {...}**
* **Combinateurs :**

Un élément **enfant** a donc un lien direct et immédiat avec son élément parent alors qu’un lien **descendant**, non. Vous avez un lien direct et immédiat avec vos parent. Cependant, vous êtes le descendant de vos grands-parents : votre lien n’est pas direct et immédiat

* **Utilisation de l’esperluette** (**&**)

Mettre une esperluette en préfixe devant un sélecteur le reliera directement au **sélecteur parent** sans recourir à des combinateurs.

**Exple :** Nous voulons utiliser un pseudo sélecteur **:hover** sur notre **<li>**

**Code Base en CSS**

**Code Base en SCSS**

* **Bugs du nesting**

Avant on se servait du nesting pour imbriquer nos propriétés css de tel manière a respecté la structure du html, mais on se retrouve vite confronte aux problèmes de **bug de spécificité des éléments** quand on déjà a plus de **2 niveaux d indentation .**

* **Solution :**

Pour résoudre cela on se sert donc de  **l’esperluette** (**&**) sur les éléments enfants.

Code Base en CSS

Code Base en SCSS

On peut créer deux classe dans un sélecteurs en incluant juste la deuxième dans la premier avec un esperluette.

Ici nous avons créés deux classes dans un sélecteur et pourra être appelé par exemple : **<div class= ‘’btn--outline btn--disabled'’></div>**

* **Le modèle 7-1**
* **“7”**correspond aux sept directories thématiques (des dossiers, en langage de développeur) pour ranger vos fichiers, qui sont regroupés dans un fichier ;
* **“1”** est un fichier .scss unique qui se compile sous forme de feuilles de style CSS pour votre site.

Nous avons comme fichier :

* **Base :** contient les fichiers qui définissent les fondations de votre site, par exemple la police de caractères et les normes que vous voulez appliquer sur tout votre site, telles que le box-sizing **;**
* **Utils :** contiendra lesvariables, fonctions, mixins et les %placeholders pour les extensions (si vous en utilisez) ;
* **Layout** (mise en page) : est le dossier où vous mettez vos blocs BEM qui contiennent ce qui est réutilisable, par exemple un header pour les mises en page de grande taille ou un footer ;
* **Composants :** est utilisé pour ranger les blocs BEM qui sont plus indépendants, comme les boutons ;
* **Pages :** contient des règles spécifiques à une seule page qui ne seront pas réutilisées ailleurs ;
* **Themes :** c’est ici que vous stockez le code thématique, par exemple un style customisé pour Noël ou pour l’été. On ne l’utilisera pas dans notre site;
* **Vendors** (tiers) : est un directory pour des feuilles de style externes comme Bootstrap ou jQuery UI. En gros, il s’utilise pour tout CSS venant de l’extérieur.

**NB :** Lorsque vous séparez votre code, les fichiers individuels font tous partie d’une codebase globale, du coup Sass les appelle des **partiels**. Pour indiquer à Sass qu’un fichier est un partiel, vous devez lui ajouter le préfixe **underscore** (**\_**). **Exple : \_variables.scss**

Nous devons maintenant :

* Couper les variables de notre fichier principale pour le colle dans le fichier  **\_variables.scss**
* Dans le fichier principale faire une importation :

**@import "./utils/variables";**

L’ordre d’import est le suivant :

* **1. Utils** :
  + Variables.
  + Fonctions.
  + Mixins.
  + Placeholders.
* **2. Feuilles de style de tiers** (Vendors) (si vous en avez).
* **3. Base.**
* **4. Composants.**
* **5. Layout.**
* **6. Pages.**
* **7. Thèmes.**
* **Etapes pour lancer Sass**
* Installer NodeJs
* Installer Sass ( **-g** , est mis pour global) : **npm -g install sass**
  + **Installer Sass dans Vscode :**
* Installer l’extension **Live Sass Compiler** dans Vscode
* Ouvrer l’invite de commande
* Initier un fichier JSON : **npm package.json**
* Créer un fichier de package : **npm init**
* Installer Sass maintenant : **npm install -g sass**
* Savoir la version de Sass : **sass –version**
* Dans le **package.json** au niveau de **script** entrez la cmd :

**sass --watch ./sass/main.scss:./public/css/style.css**

**--watch :**  est un flag npm, il observe si un changement opère et s’il en voit, il recompilera et mettra à jour le fichier CSS.

**./sass/main.scss :**  correspond au fichier Sass sur lequel on travaille.

**./public/css/style.css :**  correspond au fichier après compilation.

* Lancer Sass avec : **npm run sass**

04 Mode de compilation :

* **nested**(mode indentation):

**"sass": "sass --watch ./sass/main.scss: ./public/css/style.css --style nested"**

* **expanded**(mode css habituel):

**"sass": "sass --watch ./sass/main.scss : ./public/css/style.css --style expanded"**

* **compact**(mode plus petit):

**"sass": "sass --watch ./sass/main.scss : ./public/css/style.css --style compact"**

* **compressed**(mode minifier):

**"sass": "sass --watch ./sass/main.scss : ./public/css/style.css --style compressed"**

* **Warning : @warn ‘’message’’ ;**

**Warning** avertit l’utilisateur mais ne stoppe pas le code

* **Error : @error ‘’message’’ ;**

**Le Error** stoppe immédiatement le code quand il y a erreur.

* **Debug : @debug ‘’valeur’’ ;**

**Le Debug** permet d’afficher les valeurs pour ainsi voir les résultats.

* **Les listes et maps :**
  + **Liste :**

**Une liste** de valeurs permet de **regrouper** des valeurs dans une seule variable.

On l’a définie soit avec des **espaces ou des virgules, avec ou sans parenthèses** :

La compilation se présente comme suit :

* **Pour utiliser**  les éléments d’une liste, on se sert de **nth(nom\_liste, indice)**

On veut accéder à la **valeur 2rem** qui se situe à la **4 position**. En Sass on commence toujours à compte à partie de l’indice **1**

* Map :

Les Maps sont très semblables aux listes, mais elles assignent à chaque valeur un nom sous forme d’une **paire clé/valeur.**

La syntaxe d’écriture des maps est la suivante :

**Exple :**

* Pour accéder à un élément d’un map, on se sert de : **map-get(nom\_map, cle)**

**Exple** d’utilisation des map dans la gestion des couleurs, une map dans autre map :

* On peut se servir d’une mixin pour accéder aux valeurs  de notre map **$txt-input-palette**:
* **Les boucles**

La boucle **@each** a la syntaxe suivante :

En utilisant les boucles on peut facilement accéder au valeur du map  **$txt-input-palette**

* Mais si nous voulons maintenant utiliser les clés de **$txt-input-palette** comme **:pseudo sélecteur**, alors on va se servir de  **l’interpolation.**

**NB : interpolation** c est le fait de substituer la valeur d une variable dans une chaine de caractère. Un peu comme en **python**(f-string), JavaScript etc…

**Syntaxe : #{variable}**

* **Resultat Obtenu**
* **Ajout des Breakpoint pour le responsive**

Sass nous permet de placer les **media queries** directement dans les sélecteurs :

* On peut se servir des maps pour stocker nos Breakpoint et ainsi a l’aide des mixins faire facilement du **@media queries**
* Mais là nous voyons que notre code est code en **dur** dans la mixin et ce n’est pas optimal.
* **Solution :** On va devoir utilise **@content** pour palier a cela. Car @content va être remplace par le code qui sera écrite plus tard dans la mixin

Code Compiler

**Exple :**

**NB :** lorsqu’ on veut centrer un conteneur au centre sur **Y**  on doit lui donne une **height (**généralement **100% de son parent**)

**Sans hauteur** le conteneur prend de la hauteur en fonction des éléments qui seront présents dans ce conteneur.

* **Autoprefixer pour créer du code adapté à tous les navigateurs**

Lorsqu’ on utilise par exemple **FlexBox**, au lieu de simplement définir la propriété **display** comme étant flex, on doit lister tous les préfixes :

* **-webkit** est pour Chrome et Safari,
* **-ms** pour Microsoft,
* **-moz** pour Firefox

**Autoprefixer** est un plugin qui vous sauvera des -webkit- et des -moz-. Il fait exactement ce que son nom indique : il ajoute automatiquement des préfixes dans votre CSS. Il vous suffit de lui fournir une feuille CSS et il la passera en revue pour y ajouter les préfixes là où il faut.

* **Installation de Autoprefixer :** dans le terminal entre
* **npm install autoprefixer postcss postcss-cli -g**
* **PostCSS-CLI** est un outil de ligne de commande que vous utiliserez pour faire tourner Autoprefixer.
* Aller dans le **package.json** pour ajouter un nouveau script, cle **prefix** et valeur  **chemin du fichier css** suivi du package **postcss** de Autoprefixer et enfin  **le chemin du new fichier css préfixé**
* Maintenant nous devons vérifier les versions des navigateurs. Pour être sûr on demandera qu’ il prenne les **4 derniers versions des navigateurs lorsqu’ il vérifiera la compatibilité des feuilles de style css.** Dans ce cas on va ajouter une cle **browserslist** et aura pour valeur  **last 4 versions** dans le fichier package.json
* Pour lancer Autoprefixer on utilise la commande suivante :
* **npm run prefix**

**Exercice :** Mettez les valeurs dans une map en attribuant une clé à chacune, puis utilisez une boucle @each pour générer des sélecteurs ayant la structure suivante : **.article\_\_heading—color**

***Solution :***

* Structure HTML
* Convertis la taille de base du **html** de **16px** en **10px**:

html {

    // 10px div by def of 16px

*font-size*: 62.5%;

}

* **Excercices en CSS:**

Cette ligne permet de retirer les erreurs dans VsCode cause par CSS

*"css.validate"*: false,

* Exercice 1 : **Alignement**
* 1. A l’aide de de flexBox.
* 2. Avec grid :
* 3. Avec **position relative**  sur le parent et **position absolut**  sur l’enfant et ensuite on fait le translate(-50px, -50px).
* 4. A l’aide de flexBox et des margin
* 5. A l’aide d’une hauteur de ligne(**line-height : taille du conteneur**)
* 6. A l’aide du **display : table**

**NB :** Cette methode n’est plus utilisée de nos jours !!!

* Exercice2 : **NavBar**
* Pour Envoyez un élément a **gauche** on se sert du **margin-right : auto ;**
* Pour envoyez un élément a **droite** on se sert du **margin-left : auto ;**
* **el: first-of-type** dit ici que : dans la **classe el**  ou **élément el** prend chaque premier élément de même type
* Ici on donne des taille relative aux div **link-container et empty-layout** ainsi ils prendront chacun la width maximale de l’écran , mais de base sur les **flex** on a **skrink :1** c est a dire que les éléments se partagerons l’espace disponible.