**Cascading style Sheets (CSS)**

**Jean-Claude AZIAHA**

**SOMMAIRE**

1-Qu’est-ce que le CSS ?

2-Comment l’utilise-t-on dans un fichier HTML ? (Règles ou normes à suivre)

3-Syntaxe

3-a) Sélecteurs

3-b) Propriétés

3-c) Valeurs

4-Les polices

4-a) Les polices de bases

4-b) Intégration de nouvelles polices

4-c) Autres propriétés liées à la police

5-Les textes

6-Les couleurs

7-Le model des boites

7-a) Eléments de type block et inline

7-b) padding border margin

8-Position des éléments (Partie 1)

8-a) Propriété position : static / relative / absolute / fixed / sticky

8-b) Propriété float / clear

9-Position des éléments (Partie 2) : Le flexbox

9-a) Qu’est-ce que le flexbox ?

9-b) Direction des éléments flexibles

9-c) Alignement des éléments flexibles

9-d) Taille des éléments flexibles

10- Position des éléments (Partie 3) : Les grilles

10-a) Qu’est-ce qu’une grille ?

10-b) Créer une grille et positionner des éléments dans cette grille

10-c) Aligner et espacer les éléments dans une grille

10-d) Etendre un élément particulier d’une grille

10-e) Autres propriétés utiles

11- Le Responsive Design

11-a) Principes du Responsive Design

11-b) La balise « meta viewport »

11-c) Les media queries

12-Aller plus loin

13- Projet concret regroupant la plupart des notions apprises.

1. **Qu’est ce que le CSS ?**

Le CSS est un acronyme et qui signifie en réalité Cascading Style Sheets.

C’est un langage de programmation. Il a pour fonction de mettre en forme, du contenu déjà écrit et structuré grâce au HTML.

En gros, tout ce qui est du design, gestion des couleurs, taille de polices… (comme cela se fait dans un document word par exemple) sera fait sur une page web avec du CSS.

**2-Comment l’utilise t – on ?**

\* Méthode 1 : Stricte séparation du code HTML et du code CSS

–Un fichier, **page.html** par exemple qui **contiendra** notre code **HTML**

–Un autre fichier, par exemple **style.css** qui **contiendra** notre code **CSS**

Cette manière est la plus recommandée car elle permet de faire une distinction claire entre le code HTML et CSS.

Ce qui favorise donc la bonne lecture du code. Cela nous fait gagner du temps lorsque nous voulons changer des informations ou détecter une erreur par exemple.

Toutefois, il va falloir **relier** le code **CSS** écrit à part, à notre code **HTML** afin qu’il soit pris en compte. On l’indique donc de la manière suivante.

**Exemple:**

Une image contenant moniteur, écran, table, assis

Description générée automatiquement

Sur la photo de notre exemple ci-dessus, à la ligne 6, nous avons utilisé l’élément link qui permet de relier un fichier CSS à notre fichier HTML souhaité grâce à l’attribut href qui indique le chemin qui mène à l’emplacement du fichier et rel pour spécifier que c’est une feuille de style ; Autrement dit, un fichier CSS.

Nous pouvons créer autant de fichiers HTML et CSS que nous voulons.

Nous pouvons également relier un fichier CSS à plusieurs fichiers HTML.

Nous verrons tout ça concrètement dans la pratique.

Voici un exemple de ce que peut contenir un fichier CSS :

*Une image contenant photo, noir, blanc, tenant

Description générée automatiquement*

Voici le contenu du fichier CSS que nous avons relié à notre fichier HTML montré un peu plus haut.

Le CSS a aussi sa propre syntaxe. Ce code ci-dessus dit de mettre la couleur de l’arrière-plan de notre page en rouge.

\*Methode 2 : Ecrire notre code CSS dans le <head> de notre fichier HTML

Une image contenant capture d’écran, écran, moniteur, téléphone

Description générée automatiquement

Pour réaliser cela, on a donc besoin de l’élément html <style></style> et l’on va pouvoir mettre à l’intérieur, notre code CSS.

\*Methode 3 : Ecrire notre code CSS directement dans la balise ouvrante de l’élément souhaité

Une image contenant moniteur, écran, ordinateur, portable

Description générée automatiquement

**3-Syntaxe**

Voici la syntaxe de base pour appliquer un style à un élément.

xxxxx

{

propriété1: valeur ;

propriété2: valeur ;

propriété3: valeur ;

…

}

*sélecteur*

**3-1) Les sélecteurs naturels**

Les sélecteurs naturels représentent tous les éléments HTML que nous avons utilisé dans le body de notre fichier HTML (ex : h1, p, div, table…)

**Exemple :**

Ce code veut dire que tous les **titres h1** de ma page HTML à laquelle j’ai relié ce fichier CSS auront une couleur **rouge**.

**Une image contenant mètre, horloge

Description générée automatiquement**

**On peut appliquer un style à plusieurs éléments**

**Exemple :**

Ce code signifie que l’on veut affecter une **couleur rouge** à tout ce qui est titre **h1, h2, p** de ma page html.

Une image contenant noir, mètre, horloge, blanc

Description générée automatiquement

**Pour appliquer un style à un élément contenu dans un autre**

**Exemple :**

Ce code veut dire : pour **chaque** élément **div** de ma page qui **contient** un ou des **paragraphes**, mets la couleur de ce ou ces paragraphes en **bleu**.

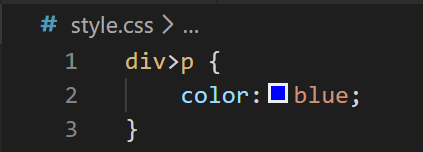
Une image contenant mètre, noir, orange, écran

Description générée automatiquement

**Pour appliquer un style à un élément qui a un parent en particulier**

**Exemple :**

Ce code applique une couleur **bleue** à tous les **paragraphes** qui sont des **enfants directes** de notre **div** qui est donc le parent.



**Pour appliquer un style à un élément qui en suit un autre**

**Exemple :**

Ce code demande à ce que **chaque paragraphe** qui **suit** un **div** prenne une couleur **bleue**.

Une image contenant noir, sombre, moniteur, rue

Description générée automatiquement

C’est super tout ça car cela fonctionne mais un problème se pose.

Lorsque j’ai 5 paragraphes dans mon code html et je ne veux cibler que le troisième afin de lui affecter une couleur bleue, comment je fais ?

C’est dans ce sens qu’interviennent les sélecteurs artificiels.

**3-2) Les sélecteurs artificiels**

Le terme sélecteur artificiel pour simplement signifier que l’on ne va pas appeler un élément HTML par son nom d’origine (ex : p, div, h1…)

Nous allons plutôt les cibler grâce aux **identifiants (id)** et aux **classes (class).**

De cette manière nous pouvons agir sur n’importe quel élément sélectionné de notre fichier.

**Pour appliquer un style à l’élément qui a un attribut id="identifiant"**

**Exemple :**

**-**Dans le fichier html, voici la syntaxe

Une image contenant intérieur, assis, moniteur, ordinateur

Description générée automatiquement

**-**Puis dans le fichier CSS

Une image contenant noir, moniteur, horloge, blanc

Description générée automatiquement

Il faut donc mettre un **#** accompagné de la **valeur** donné à l’id.

Ici j’ai donné à mon id, la valeur ‘identifiant’ mais je pouvais aussi mettre n’importe quoi d’autre.

La **particularité** d’utiliser **l’identifiant** est qu’il ne doit **cibler** **qu’un seul** et unique élément à la fois. Ce qui est logique.

Par conséquent, on ne peut pas avoir 2 identifiants avec la même valeur.

Cependant, lorsque nous voulons cibler 2 paragraphes parmi les 5 sur notre page HTML par exemple, afin d’y apporter les mêmes modifications, comment pouvons-nous faire ? Puisque l’identifiant ne cible qu’un élément à la fois.

Première solution : Donner un identifiant à chaque paragraphe puis agir là-dessus en CSS. Cela fonctionne mais prend trop de temps.

C’est dans cette logique qu’interviennent les classes.

Deuxième solution : Donner le même nom de classe aux paragraphes dont nous voulons modifier la couleur du texte puis agir là-dessus en CSS.

**Pour appliquer un style à un ou plusieurs éléments qui ont un attribut *class="cible"***

**Exemple :**

**-**Dans le fichier html, voici la syntaxe

Une image contenant téléphone, téléphone mobile, moniteur, écran

Description générée automatiquement

**-**Puis dans le fichier CSS

Une image contenant noir, moniteur, écran, signe

Description générée automatiquement

Il faut donc mettre un **.** **suivi** de la **valeur** de la classe.

**4-Les polices**

**4-1) Les polices de bases**

Les propriétés de type **font** vont nous permettre d’agir sur la police d’écriture.

Dans ce sous-chapitre, nous allons voir la propriété font-family.

Il sert à indiquer le nom de famille de notre police.

Une image contenant capture d’écran

Description générée automatiquement

Une image contenant capture d’écran

Description générée automatiquement

Une image contenant capture d’écran

Description générée automatiquement

Les 3 images ci-dessus ont été récupérées sur le site [www.w3schools.com](http://www.w3schools.com)

Voyons concrètement comment utiliser une font-family dans notre code CSS.

Une image contenant dessin, alimentation

Description générée automatiquement

Ce code veut dire : pour tous les titres **h1, h2** et **p** de notre page html, je veux qu’ils aient un couleur **rouge** et une **font-family** qui sera **Georgia, ‘Times …**

Pourquoi en avons-nous listées plusieurs ?

Il faut toujours prendre l’habitude d’en mettre plusieurs au cas où le premier cité (Georgia dans notre cas) venait à ne pas être reconnu par un navigateur, la seconde font-family (‘Times New Roman’) qui prendra le relais.

**4-2) Intégration de nouvelles polices**

Il peut arriver que vous vouliez utiliser une famille de police qui n’est pas par défaut présente dans le CSS.

<https://fonts.google.com/>

**Exemple :**

**Une image contenant capture d’écran

Description générée automatiquement**

Une image contenant alimentation, dessin

Description générée automatiquement

**4-3) Autres propriétés**

* **font-size** : Pour modifier la taille de la police.

Nous utiliserons les **pixels(px)** et les **pourcentages(%)** essentiellement pour ce cours.

Toutefois, il faut savoir qu’il en existe d’autres (Voir la dernière rubrique : Allez plus loin)

Exemple :

Une image contenant noir, sombre, tenant, écran

Description générée automatiquement

Une image contenant mètre, noir, horloge

Description générée automatiquement

* **font-weight** : Pour modifier le poids de notre police.

**bold** pour mettre en gras

**normal** pour une valeur par défaut

* **font-style** : Pour modifier l’inclinaison de notre police.

**normal** : valeur par défaut, les caractères seront droits

**italic** : la police va s’afficher en italique

**oblique** : la police va être tordue pour être rendue de façon oblique.

**5- les textes**

* **text-align** qui va nous permettre de gérer l’alignement du texte
  + **left** : Le texte sera aligné contre le bord gauche de l’élément qui le contient. C’est l’alignement par défaut.
  + **center** : Le texte sera centré dans l’élément qui le contient
  + **right** : Le texte sera aligné contre le bord droit de l’élément qui le contient
  + **justify** : Le texte sera justifié (les écarts entre les mots vont être calculés de façon à ce que chaque ligne de texte occupe la même largeur).
* **text-transform** qui va nous permettre de gérer la casse du texte (le fait que le texte soit en majuscules ou en minuscules)
  + **none** : Il n’y a aucune transformation du texte. Cela peut être utile pour annuler une transformation. C’est la transformation par défaut
  + **lowercase** : Transforme tout le texte d’un élément en minuscules
  + **uppercase** : Transforme tout le texte d’un élément en majuscules
  + **capitalize** : Transforme la première lettre de chaque mot en majuscule
* **text-decoration** qui va nous permettre d’ajouter des éléments de décoration autour du texte comme un trait de soulignement par exemple
  + **underline** : ajoute un trait de soulignement au texte
  + **overline** : ajoute un trait au dessus du texte
  + **line-through** : ajoute un trait qui va barrer le texte
  + **underline**  **overline**: ajoute un trait de soulignement et un trait de surlignement au texte.
* **text-indent** qui va nous permettre de définir l’indentation d’un texte

Nous allons ici pouvoir passer une valeur en **px**, **em**, **%** etc. Les valeurs en **%** vont être exprimées en fonction de la largeur de l’élément parent. Notez que les valeurs négatives sont acceptées.

* **text-shadow** qui va nous permettre d’ajouter des ombres autour d’un texte avec **4 valeurs.**

1. Déplacement **horizontal** de l’ombre par rapport au texte. En passant une **valeur positive**, l’ombre est projetée à **droite** du texte. En passant une **valeur négative**, l’ombre est projetée à **gauche**. (Cette valeur est obligatoire).
2. Déplacement **vertical** de l’ombre par rapport au texte. En passant une **valeur positive**, l’ombre est projeté **sous le texte**. En passant une valeur **négative**, l’ombre est projetée **au-dessus**. (Cette valeur est obligatoire).
3. Au rayon de **flou** de l’ombre. Un flou Gaussien est utilisé ici : plus la valeur est grande, plus l’ombre sera étendue et floue. (Cette valeur est facultative)
4. A la couleur de l’ombre.

* Les 3 propriétés ci-dessous vont nous servir à gérer les espaces entre les lignes et caractères
* La propriété **line-height :** Définir les espaces entre les lignes
* La propriété **letter-spacing :** Définir les espaces entre caractères
* La propriété **word-spacing :** Définir les espaces entre les mots

Voyons tout cela à travers un exemple

Une image contenant capture d’écran, moniteur, écran, table

Description générée automatiquement

Une image contenant capture d’écran, téléphone

Description générée automatiquement

**6-Les couleurs**

On peut donner une couleur à un texte, à l’arrière-plan d’un élément structurant…

Pour se faire il faut procéder par l’une des **3 méthodes** ci-dessous.

* Un nom de couleur (en anglais) ex : p {color : red ;}

Il y en a environ 130 au total.

* Une notation **RGB** et/ou **RGBa** ex : p {color : rgb(63, 8, 165) ;}

Cette notation s’avère très utile lorsqu’on veut appliquer une couleur qui n’existe pas dans les 130 de départ.

En effet, RGB est une abréviation de Red Green Blue.

Le principe est donc de **mélanger**, dans l’ordre, du rouge du vert et du bleu pour **obtenir** une couleur **unique**. La valeur minimale est **0** est celle maximale **255** pour chacune des couleurs.

Il est également possible d’ajouter une **opacité** à la couleur. Pour cela, il faut utiliser la syntaxe **rgba**(…, …, …, 0.3). 0.3 dans cet exemple est la valeur de l’opacité. Elle possède une valeur comprise entre 0 et 1.

Exemple : p {color : rgb(63, 8, 165, 0.6) ;}

* Une notation hexadécimale ; ex : p {color :#C3C3C3 ;}

Même logique que le rgb sauf qu’ici, **on va combiner des lettres et de chiffres pour indiquer une couleur**. Elle est composée de 6 caractères qui contiennent des lettres de **0** à **9** ou des chiffres de **A** à **F**. Ces caractères sont toujours précédés du signe **#**.

**Ces lettres ou chiffres fonctionnent deux par deux**. Les 2 premiers indiquent une quantité de rouge, les 2 suivants une quantité de vert, et les 2 derniers une quantité de bleu.

**7-le model des boites**

Tous les éléments HTML définissent des boîtes mais certains sont dit « éléments de type bloc » et d’autres sont dit « éléments de type en ligne ».

* Eléments de type bloc (*block*)

<header> <section> <footer> <aside> <article> <div> <h1>

<h2> <h3> <h4> <h5> <h6> <p> <table> <tr> <td> <th> <ol>

<ul> <li> <dd> <dl> <dt> <form> <hr /> …

* Eléments de type en-ligne (*inline*)

<img> <a> <em> <strong> <kbd> <br /> <span>…

Elément bloc ? en-ligne ? Quelle différence?

Les éléments, s’ils sont de type bloc ou s’ils sont de type inline **se**

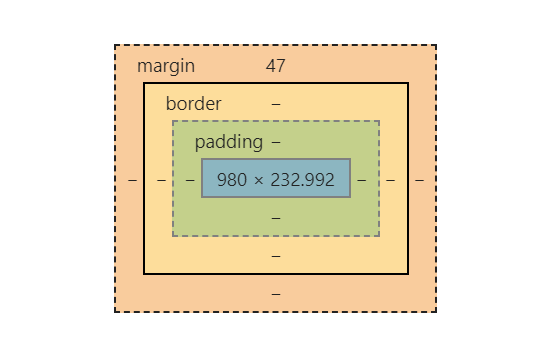
**positionnent différemment** les uns par rapports aux autres :

Éléments « bloc/block » : Ce sont des éléments qui prennent toute la largeur de l’élément parent. Ces éléments se positionnent donc **les uns sous les autres** ; ils peuvent contenir d’autres éléments « bloc » ou des éléments « en-ligne ».

Éléments « en-ligne/inline » : Ces éléments prennent juste l’espace en largeur dont il a besoin, en fonction de son contenu. Ces éléments se positionnent les **uns à côté des autres** ; ils peuvent s’imbriquer entre eux mais ne peuvent pas contenir d’éléments « bloc ».

Tout élément HTML est donc une « boîte » et possède :

* **Un contenu** (de largeur **width**)
* **Une marge interne** (**padding**)
* **Une marge externe** (**margin**)
* **Une bordure** (**border**)



**Exemple 1** :



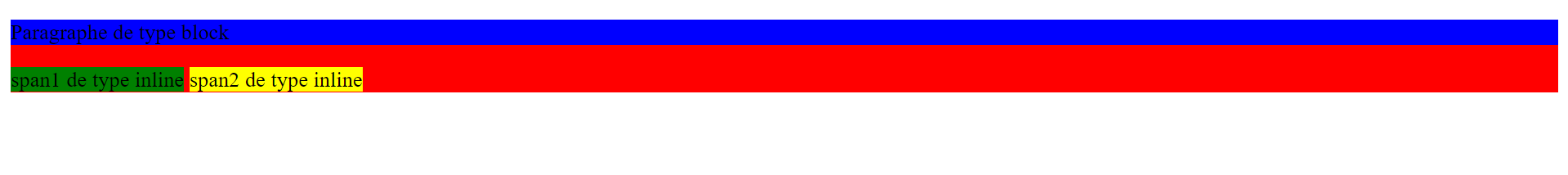
Une image contenant moniteur, écran, télévision, téléphone

Description générée automatiquement

Une image contenant photo, noir, tenant, signe

Description générée automatiquement

**Exemple 2** :



Une image contenant écran, moniteur, télévision

Description générée automatiquement

Une image contenant capture d’écran, moniteur, écran, portable

Description générée automatiquement

Il est possible de définir une valeur pour le coté spécifique des marges ou bordures.

On peut donc avoir :

* margin-left, margin-right, margin-top, margin-bottom
* border-left, border-right, border-top, border-bottom
* padding-left, padding-right, padding-top, padding-bottom

Exemple:

Une image contenant moniteur, écran, table, téléphone

Description générée automatiquement

Une image contenant écran, téléphone

Description générée automatiquement

Une image contenant debout

Description générée automatiquement

**8-Position des éléments (partie 1)**

**8-a) Propriété position**

La propriété position va nous permettre de positionner nos éléments sur une page.

Une fois le type de positionnement défini avec position, nous allons pouvoir donc positionner un élément à un endroit précis dans une page grâce aux propriétés top, left, bottom et right.

Il y en a 5 :

**Position : static**

C’est la position par défaut de tous les éléments HTML. Les éléments restent toujours dans le flux normal de la page mais de manière statique. On ne va donc pas pouvoir les déplacer grâce au left, right, top et bottom.

**position : relative**

Cette position permet d’indiquer au CSS que nous voulons avoir la possibilité de modifier cet élément comme on le souhaite par rapport à son élément parent. Les éléments restent toujours dans le flux de la page. On va donc pouvoir les déplacer maintenant grâce au left, right, top et bottom.

**position : absolute**

Cette position nous permet également de modifier l’élément sur lequel il est appliqué. Cependant, il va faire sortir l’élément du flux de la page.

S’il est placé dans un élément parent, on pourra le déplacer grâce au left, right, top et bottom par rapport à cet élément. Sinon, il est placé directement par rapport au body.

**position : fixed**

Cette position nous permet de fixer l’élément sur la page. C’est-à-dire qu’elle ne bougera pas même si nous scrollons la page. Elle fait également sortir l’élément du flux de la page.

**position : sticky**

Ceci définit dans un premier temps, la position de l’élément à relative et reste donc dans le flux de la page.

Une fois que l’élément est pris en compte ou que le scroll arrive au niveau de l’élément, il change et prend une position fixe. Cette position fait à ce moment sortir l’élément du flux de la page.

**z-index**

Ce n’est pas vraiment une position mais il faut que je vous en parle parce que c’est une bonne astuce qui va nous permettre de résoudre des conflits entre les positions.

En effet, lorsque nous affectons plusieurs positions sur différents éléments de notre page, il peut arriver que certains s’affichent au-dessus des autres alors que nous, nous voulons l’inverse.

C’est là que le z-index va intervenir et va être un outil puissant afin de résoudre le problème.

Pour se faire, il faut appliquer la propriété z-index à l’élément A dont on souhaite qu’il prenne le dessus sur l’élément B. La valeur du z-index de A doit être supérieur à la valeur du z-index de B.

**pratiquons**

Voyons donc tout ça à travers des exemples afin de mieux visualiser les choses.

**8-b) propriété float /clear**

**float**

La propriété float appliquée à un élément va le transformer un élément de type inline.

Cette propriété est super pratique car elle va nous permettre de déplacer l’élément sur la largeur en fonction de son élément parent.

float possède 3 valeurs :

**Left** : pour placer les éléments à gauche

**Right** : pour placer les éléments à droite

**None** : pour signifier que cet élément n’est pas flottant

**Clear**

Cette propriété va empêcher un élément de se positionner à côté d’un élément flottant.

Elle peut prendre principalement 4 valeurs :

**left** :Empêche l’élément sur lequel elle est appliquée de se positionner à coté d’un élément flottant à gauche

**right** Empêche l’élément sur lequel elle est appliquée de se positionner à côté d’un élément flottant à droite.

**both** Empêche l’élément sur lequel elle est appliquée de se positionner à côté d’un élément qu’il soit flottant à gauche ou à droite.

**none** Permet aux éléments de se positionner à côté des éléments flottant.

Exemple :

Une image contenant moniteur, écran, téléphone, table

Description générée automatiquement

Une image contenant capture d’écran, téléphone

Description générée automatiquement

**9-POSITION DES éLéMENTS (PARTIE2) :**

**9-A) Le flexbox**

Le Flexbox est un outil puissant qui va nous permettre de positionner nos éléments plus facilement.

Pour se faire, il faut d’abord indiquer sur l’élément dont on souhaite modifier la position de ses enfants un display : flex

**9-b) Direction des éléments flexibles**

Après, il faut indiquer la direction dans laquelle les éléments vont être alignés.

Avant d’indiquer cette direction, il faut comprendre qu’il existe 2 axes lorsqu’on utilise la position flex à savoir : **L’axe principal et l’axe secondaire**.

**L’axe principal** peut être horizontale ou verticale. Pour cela, il faut faire :

-flex-direction : row (axe horizontal de la gauche vers la droite)

-flex-direction : row-reverse (axe horizontal de la droite vers la gauche)

-flex-direction : column (axe vertical de la gauche vers la droite)

-flex-direction : column-reverse (axe vertical de la droite vers la gauche)

**9-C) l’alignement des éléments**

Lorsque nous appliquons une « flex-direction » à un élément, ses enfants se positionnent par défaut à gauche.

Ce n’est pas toujours ce que l’on a envie de faire. On peut ainsi pouvoir aligner les éléments avec des espacements différents mais toujours dans la même direction.

Pour modifier l’alignement de ses enfants, il faut utiliser la propriété **justify-content** en faisant :

-justify-content : flex-start (les éléments enfants se positionnent à gauche dans leur conteneur ou élément parent)

-justify-content : flex-end (les éléments enfants se positionnent à droite dans leur conteneur ou élément parent)

-justify-content : center (les éléments enfants se positionnent au centre de leur conteneur ou élément parent)

-justify-content : space-between (Les éléments sont répartis dans le conteneur. Les éléments se trouvant au bord du conteneur vont s’y coller)

-justify-content : space-around (Les éléments sont répartis dans le conteneur. La taille de l’espace entre le conteneur et un élément contre le bord du conteneur sera deux fois plus petite qu’entre deux éléments).

-justify-content : space-evenly (Les éléments sont répartis dans le conteneur. L’espace entre le bord du conteneur et un élément sera le même que celui entre deux éléments).

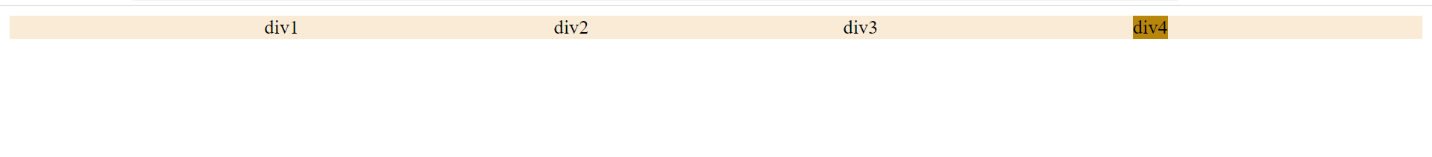
Exemple :

Une image contenant table, moniteur, écran, téléphone

Description générée automatiquement

Une image contenant capture d’écran

Description générée automatiquement



Ainsi, nous venons d’utiliser l’outil flex pour positionner les éléments d’un conteneur dans la direction que nous voulons et nous avons espacé ou aligné ces éléments en fonction de la direction choisie.

Jusque-là, nous avons pu positionner nos éléments en fonction de l’axe principal. Nous pouvons aller plus loin et utiliser l’axe secondaire pour définir de nouveaux alignements à nos éléments enfants. Pour se faire, il me faut utiliser la propriété **align-items**

align-items :stretch : Valeur par défaut. Les éléments vont s’étirer dans leur axe secondaire jusqu’à remplir tout l’espace disponible ;

* align-items :flex-start : Les éléments vont être placés au début de leur conteneur en fonction de l’axe secondaire ;
* align-items :flex-end  : Les éléments vont être placés à la fin de leur conteneur en fonction de l’axe secondaire ;
* align-items :center  : Les éléments vont être placés au milieu de leur conteneur en fonction de l’axe secondaire ;
* align-items :baseline  : Les éléments vont être alignés dans leur axe secondaire de telle sorte à ce que leurs lignes de base (ligne imaginaire sur laquelle est écrit le texte) soient alignées.

**10- LES GRILLES**

**10-a) Qu’est- ce -qu’une grille ?**

Les grilles fonctionnent pratiquement de la même manière que les flexbox.

Cependant, elles sont plus puissantes car elles nous permettent d’accéder à des propriétés que les flexbox ne possèdent pas.

**10-b) Créer une grille et y positionner des éléments**

Pour créer une grille il faut appliquer la propriété

display : grid ;

ou

display : inline-grid ;

Pour y positionner des éléments :

grid-template-columns : Cette propriété peut prendre une valeur auto, en pixels(px), en pourcentage(%) ou en fraction(fr).

Elle permet de définir le nombre de colonnes présentes sur chaque ligne de la grille. Elle définit donc implicitement le nombre de lignes total de la grille.

grid-template-rows : Cette propriété peut également prendre une valeur auto, en pixels(px), en pourcentage(%) ou en fraction(fr).

Elle permet de définir la hauteur d’une ou plusieurs lignes de la grille.

**10-c) Aligner et espacer les éléments dans une grille**

**Aligner les éléments dans une grille**

**En fonction de l’axe principal**

-justify-content : start (les éléments enfants se positionnent à gauche dans leur conteneur ou élément parent)

-justify-content : end (les éléments enfants se positionnent à droite dans leur conteneur ou élément parent)

-justify-content : center (les éléments enfants se positionnent au centre de leur conteneur ou élément parent)

-justify-content : space-between (Les éléments sont répartis dans le conteneur. Les éléments se trouvant au bord du conteneur vont s’y coller)

-justify-content : space-around (Les éléments sont répartis dans le conteneur. La taille de l’espace entre le conteneur et un élément contre le bord du conteneur sera deux fois plus petite qu’entre deux éléments).

-justify-content : space-evenly (Les éléments sont répartis dans le conteneur. L’espace entre le bord du conteneur et un élément sera le même que celui entre deux éléments).

**En fonction de l’axe secondaire**

-align-content : start (les éléments enfants se positionnent à gauche dans leur conteneur ou élément parent)

-align-content : end (les éléments enfants se positionnent à droite dans leur conteneur ou élément parent)

-align-content : center (les éléments enfants se positionnent au centre de leur conteneur ou élément parent)

-align-content : space-between (Les éléments sont répartis dans le conteneur. Les éléments se trouvant au bord du conteneur vont s’y coller)

-align-content : space-around (Les éléments sont répartis dans le conteneur. La taille de l’espace entre le conteneur et un élément contre le bord du conteneur sera deux fois plus petite qu’entre deux éléments).

-align-content : space-evenly (Les éléments sont répartis dans le conteneur. L’espace entre le bord du conteneur et un élément sera le même que celui entre deux éléments).

**Espacer les éléments d’une grille :**

grid-gap : valeur en pixels

grid-row-gap : valeur en pixels

grid-column-gap : valeur en pixels

**10-d) étendre un élément particulier d’une grille**

Il faut choisir un élément précis d’une grille et lui appliquer :

grid-row : numéro de ligne de début / numéro de ligne de fin

grid-column : numéro de colonne de départ / numéro de colonne de fin

**10-e) Autres propriétés**

flex-basis : Etendre les éléments enfants d’un conteneur-flex

Elle peut prendre une valeur en pixels et pourcentage

flex-grow : Etendre les éléments d’un conteneur flex afin qu’ils puissent occuper l’espace restant disponible.

Elle peut prendre 2 valeurs :

flex-grow : 0 ; Dans ce cas, les éléments ne s’étendent pas.

flex-grow : 1 ; Dans ce cas, les éléments s’étendent.

flex-shrink : Donner la capacité aux éléments de se rétrécir.

flex-shrink : 1 ; Dans ce cas, les éléments peuvent se rétrécir.

flex-shrink : 0 ; Dans ce cas, les éléments peuvent s’agrandir.

**11-Le Responsive Design**

**11-a) Principes du Responsive Design**

Aujourd’hui, il est tout simplement impossible de penser créer un site internet sans tenir compte de la taille de l’écran de chacun des appareils que les gens utiliseront pour accéder à votre site.

Comme différents appareils, nous avons principalement :

-Les téléphones mobiles

-Les tablettes

-Les ordinateurs

-Les Smart TV

Qui dit donc Responsive Design voudrait tout simplement dire : adapter l’affichage de votre site sur tous les différents écrans.

**11-b) La balise « meta viewport »**

C’est la première chose à faire pour rendre son site responsive.

<meta name= "viewport" content="width=device-width, initial-scale=1.0,

user-scalable=no" />

**11-c) Les media queries**

Grace aux media-queries nous allons pouvoir définir l’affichage de notre site en fonction de la taille des différents écrans sur lesquels nos utilisateurs se connecteront.

Il faut donc utiliser la règle CSS : @media



Nous verrons concrètement comment cela fonctionne par la pratique, dans le projet qui va suivre.

**12-Aller plus loin**

Nous venons de voir à travers ce cours l’essentiel de ce qu’il faut savoir pour acquérir de solides bases en CSS3.

Cependant, cela n’est pas suffisant car il existe énormément de propriétés encore intéressantes à connaitre.

Allant dans ce sens, vous pourrez consulter les livres présents sur la plate-forme eni.fr ; Openclassroom.fr

**13- Projet concret regroupant la plupart des notions apprises**

Nous allons créer un site pour Pokémon avec les différentes pages requises.

Ce projet a été conçu principalement pour mettre en pratique la plupart des notions vues dans ce cours et en découvrir de nouvelles à travers la pratique.