**Css**

**Overflow :hiddden zidha**

1. **Apparence du text**

* font-size
* font-family
* font-style
* text-align =>Center Or left Or right
  + or text-align=> justify => La valeur **justify** de la propriété CSS **text-align** est utilisée pour justifier le texte à la fois sur le côté gauche et le côté droit de l'élément contenant.

1. **background-image et opacity**

* **background-attachment**: détermine si une image de fond défile avec le contenu d'un élément ou reste fixe à sa position[**scroll,fixed**]
* **background-position :** permet de définir la position initiale d'une image de fond à l'intérieur d'un élément[**top,bottom,left,right,center**].
* **background-repeat :** permet de contrôler la répétition d'une image de fond à l'intérieur d'un élément [repeat,repeat-x,repeat-y,repeat-x,no-repeat].
* **background-size**

**opacity :** est utilisée pour définir le niveau de transparence d'un élément, y compris son contenu et ses enfants. Elle accepte une valeur entre 0 et 1

1. **border and shadow**

* **border : 2px solid red**
* **border-width: 2px;**
* **border-style: solid;**
* **border-color: red;**
* **border-radius: 30% 70% 70% 30% / 30% 58% 42% 70%;**
* **box-shadow :** permet d'ajouter une ombre à un élément HTML ; box-shadow: 6px 6px 25px rgba(0, 0, 0, 0.5);
* **text-shadow :** permet d'ajouter une ombre à un texte à l'intérieur d'un élément HTML ; text-shadow: 3px 3px 0px rgba(0,0,0,0.2);

1. **Les apparences dynamiques**

:hover :active :focus

a:hover {

// Insérer ici les propriétés CSS à appliquer

}

1. **Overflow**

"Overflow" en CSS est utilisé pour contrôler le comportement des éléments qui dépassent les limites de leur conteneur. En français, "overflow" se traduit par "débordement".

Il existe quatre valeurs principales pour la propriété "overflow" en CSS :

1. **visible** : C'est la valeur par défaut. Les contenus débordants sont visibles à l'extérieur de leur conteneur, ce qui peut provoquer un chevauchement avec les autres éléments voisins.
2. **hidden** : Tout contenu débordant est coupé et n'est pas visible à l'extérieur du conteneur. Cela peut entraîner une perte de contenu.
3. **scroll** : Des barres de défilement sont toujours affichées, même si le contenu n'a pas débordé. Si le contenu déborde, les utilisateurs peuvent faire défiler le contenu pour le voir en entier.
4. **auto** : C'est similaire à **scroll**, mais les barres de défilement ne sont affichées que lorsque le contenu déborde réellement.

Exemple d'utilisation en CSS :

.conteneur {

  width: 300px;

  height: 200px;

  overflow: hidden; /\* ou scroll, visible, auto \*/

}

Dans cet exemple, le conteneur aura une largeur de 300 pixels et une hauteur de 200 pixels. Si le contenu de ce conteneur dépasse les dimensions spécifiées, son comportement sera déterminé par la valeur de la propriété **overflow**.

1. **Structurer votre page**

**Une image contenant texte, capture d’écran, Police

Description générée automatiquement**

1. **Le modele des boites et la mise en page**
   1. **Inline et block**

En HTML, la plupart des balises peuvent se ranger dans l'une ou l'autre de ces deux catégories :

* **block**: ce type de balise crée automatiquement un retour à la ligne avant et après ;
* **inline**: ce type se trouve obligatoirement à l'intérieur d'une balise  block  .

Voici quelques valeurs courantes de la propriété **display** :

1. **block** : L'élément est affiché comme un bloc. Il commence sur une nouvelle ligne et occupe toute la largeur disponible horizontalement.
2. **inline** : L'élément est affiché comme un élément en ligne. Il ne provoque pas de saut de ligne et s'aligne à côté des autres éléments en ligne.
3. **inline-block** : L'élément est affiché comme un élément en ligne qui conserve les propriétés d'un élément en bloc. Il démarre sur la même ligne que les autres éléments en ligne, mais peut avoir une largeur, une hauteur, des marges et des paddings définis.
4. **none** : L'élément est complètement retiré du rendu de la page. Il est invisible et n'occupe aucune place dans le flux du document.
5. **flex** : L'élément est affiché comme un conteneur flexible, permettant un positionnement et un dimensionnement flexibles de ses enfants.
6. **grid** : L'élément est affiché comme un conteneur de grille, permettant un positionnement en grille de ses enfants.

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

* 1. **Position**

Dans les méthodes que nous avons vues jusqu'à maintenant, le navigateur dispose les éléments afin qu'ils ne se superposent jamais. Ils font partie d'un flux normal. Mais nous allons maintenant voir une nouvelle propriété :**position**. Elle permet de positionner avec précision des éléments sur la page (et même parfois de les superposer).

* Le **positionnement relatif** permet de décaler l'élément par rapport à sa position normale.
* Le **positionnement absolu** permet de placer un élément n'importe où sur la page (en haut à gauche, en bas à droite, tout au centre, etc.).
* D'autres**types de positionnements** comme **fixed** ou encore **sticky**, qui ont des comportements assez semblables, peuvent être utiles si l'on veut qu'un élément, tel que le menu de navigation par exemple, reste à l'écran.

Le positionnement fonctionne aussi bien sur des balises qui sont  inline  que   block  . Mais dans les faits, vous verrez qu'on l'utilise bien plus souvent sur des balises  block  que sur des balises  inline  .

Il faut d'abord faire son choix entre les trois modes de positionnement disponibles. Pour cela, on utilise la propriété CSS **position**.

Une image contenant texte, Police, capture d’écran, ligne

Description générée automatiquement

Une image contenant texte, capture d’écran, Police, logiciel

Description générée automatiquement

1. **fixed** : L'élément est positionné par rapport à la fenêtre du navigateur, ce qui signifie qu'il reste fixe même lorsque la page est défilée. Les propriétés **top**, **right**, **bottom** et **left** sont utilisées pour spécifier la position de l'élément fixe.
2. **sticky** : L'élément est positionné relativement jusqu'à ce qu'il atteigne une position de défilement spécifiée, puis il devient fixe. Il reste fixe jusqu'à ce que la position de défilement spécifiée soit dépassée.
   1. **Margin et padding**

Une image contenant texte, capture d’écran, Rectangle, ligne

Description générée automatiquement

* 1. Z-index et relative et absolute

L'utilisation de **position: relative** sur un élément parent et **position: absolute** sur un élément enfant crée une relation de positionnement relatif entre les deux éléments. Cela signifie que l'élément enfant est positionné par rapport à son premier parent positionné.

Une image contenant texte, capture d’écran, affichage, logiciel

Description générée automatiquement

1. **Flex**

**display: flex** est utilisée pour créer un conteneur de type flexbox. Elle permet de créer des mises en page flexibles en alignant et en distribuant les éléments de manière dynamique à l'intérieur du conteneur.

**Une image contenant texte, capture d’écran, Police

Description générée automatiquement**

**flex-direction**

Flexbox permet d'agencer ces éléments dans le sens que l'on veut. Avec **flex-direction**, on peut les positionner verticalement ou encore les inverser. Cette propriété CSS peut prendre les valeurs suivantes :

* row  : organisés sur une ligne (par défaut) ;
* column  : organisés sur une colonne ;
* row-reverse  : organisés sur une ligne, mais en ordre inversé ;
* column-reverse  : organisés sur une colonne, mais en ordre inversé.

.container {

display: flex;

flex-direction: column-reverse;

}

**Flex-wrap**

Par défaut, les blocs essaient de rester sur la même ligne s'ils n'ont pas la place, quitte à "s'écraser",  et provoquer parfois des anomalies dans la mise en page (certains éléments pouvant dépasser de leur conteneur). Si vous voulez, vous pouvez demander à ce que les blocs aillent à la ligne lorsqu'ils n'ont plus la place, avec **flex-wrap**.

Voilà les différentes valeurs de flex-wrap :

1. nowrap  : pas de retour à la ligne (par défaut) ;
2. wrap  : les éléments vont à la ligne lorsqu'il n'y a plus la place ;
3. wrap-reverse  : les éléments vont à la ligne, lorsqu'il n'y a plus la place, en sens inverse.

.container {

display: flex;

flex-wrap: nowrap;

/\* OU wrap;

OU wrap-reverse; \*/

}

**Justify-content**

**Alignez sur l'axe principal avec la propriété  justify-content**

Pour changer leur alignement, on va utiliser **justify-content**, qui peut prendre ces valeurs :

* **flex-start**: alignés au début (par défaut) ;
* **flex-end**: alignés à la fin ;
* **center**: alignés au centre ;
* **space-between**: les éléments sont étirés sur tout l'axe (il y a de l'espace entre eux) ;
* **space-around**: idem, les éléments sont étirés sur tout l'axe, mais ils laissent aussi de l'espace sur les extrémités.

On écrit alors :

.container {

*display*: *flex*;

*justify-content*: flex-start /\* OU flex-end

OU center

OU space-between

OU space-around; \*/

}

**Align-content**

#### ****Alignez sur l'axe secondaire avec****align-items

Si nos éléments sont placés dans une direction horizontale (ligne), l'axe secondaire est... vertical. Et inversement : si nos éléments sont dans une direction verticale (colonne), l'axe secondaire est horizontal.

La propriété **align-items**  permet de changer leur alignement sur l'axe secondaire, grâce aux valeurs :

* **stretch**: les éléments sont étirés sur tout l'axe (valeur par défaut) ;
* **flex-start**: alignés au début ;
* **flex-end**: alignés à la fin ;
* **center**: alignés au centre ;
* **baseline**: alignés sur la ligne de base (semblable à  flex-start  ).

Pour ces exemples, nous allons partir du principe que nos éléments sont dans une direction horizontale (mais n'hésitez pas à tester aussi dans la direction verticale !).

.container {

*display*: *flex*;

*justify-content*: center;

*align-items*: center;

}

1. **Grids**

De la même manière que pour Flexbox, lorsque vous utilisez CSS Grids, vous pouvez imaginer un conteneur, une sorte de "carton" dans lequel vous allez mettre les éléments de votre grid. Sauf que, cette fois-ci, vous allez avoir besoin de déclarer la propriété **display: grid;** :

.conteneur {

*display*: grid;

}

Ensuite, il faut préciser quelques éléments supplémentaires si on veut que ça fonctionne. Pour ce faire, il s'agit de se demander comment représenter nos éléments. Lorsque vous créez un tableau, vous réfléchissez aux colonnes et aux rangées qu'il faudra créer. Eh bien, c'est la même chose ici !

On va définir les colonnes et les rangées avec deux propriétés :

1. **grid-template-columns** pour le nombre de colonnes, et la largeur de chacune d'entre elles.
2. **grid-template-rows** pour le nombre de rangées, et la hauteur de chacune d'entre elles.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

.conteneur {

*display*: grid;

grid-template-columns: 100px 300px 200px;

}

**Une image contenant texte, capture d’écran, Rectangle, ligne

Description générée automatiquement**

.conteneur {

*display*: grid;

grid-template-columns: 200px 200px 200px;

grid-template-rows: 100px 150px 200px;

}

**Une image contenant texte, capture d’écran, ligne, diagramme

Description générée automatiquement**

**Gap**

La propriété CSS **gap** permet de créer des espacements entre vos éléments. Si vous voulez garder les mêmes distances entre les rangées et les colonnes sans avoir à vous compliquer la vie, vous précisez simplement une valeur.

On écrit alors :

.conteneur {

*display*: grid;

grid-template-columns: 200px 200px 200px;

grid-template-rows: 100px 150px 200px;

gap: 10px;

}

Et on obtient :

**Une image contenant texte, capture d’écran, Rectangle, carré

Description générée automatiquement**

**Fr unite**

**Une image contenant texte, capture d’écran, affichage, ligne

Description générée automatiquement**

**Definissez la taille des éléments de votre grid**

**grid-column or grid-row**

**Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement**

**Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement**