Info 8 - Html Css – partie 3

On se propose dans ce tp de continuer l'apprentissage du langage Css. L'objectif est à présent d'explorer les méthodes de positionnement flex et grid qui sont apparues plus récemment.

Les propriétés Css suivantes seront vues dans cette activité. **Il s'agira ensuite de pouvoir les maitriser** :

- Technique « flex » : display : *flex*
  - définition de l'axe principal : *flex-direction*
  - alignement des blocs sur l'axe principal : *justify-content* et *justify-self*
  - alignement des blocs sur l'axe secondaire : *align-items* et *align-self*
  - dimension extensible d'un élément pour remplir l'espace disponible : *flex*
  - Retour à la ligne su dépassement : *flex-wrap*
  - Espacement : *gap*
  - Soulignement : *text-decoration*

- Média Queries :
  - Css conditionnel : *@media screen*

- Technique « grid » : display : *grid*
  - Définition du modèle de grille : *grid-template-columns et grid-template-rows*
  - Positionnement d'un élément : *grid-colums* et *grid-row*
  - Algorithme de répartition automatique : *grid-auto-flow*
  - Alignement des blocs sur les 2 axes : *align-items , align-self , justify-items* et *justify-self*

- Autres propriétés :
  - Remplissage du contenant d'une image : *object-fit*

1- Decouverte de « Flex » :

⇨ Dans *Visual Studio Code*, ouvrir dans un répertoire de travail, 3 nouveaux fichiers textes, l'un sera nommé *decouverte.html*, le second *decouverteFlex.css* et le troisième *decouverteGrid.css*

⇨ **Copier-coller** le script html ci-dessous dans le fichier *decouverte.html* :

```
<!DOCTYPE html>
<html lang="fr">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <link rel="stylesheet" href="decouverteFlex.css">
        <title>Découverte flex</title>
    </head>
    <body>
        <div class="parent">
            <div class="enfant un">🐸 Élément 1</div>
            <div class="enfant deux">🐱 Élément 2</div>
            <div class="enfant trois">🦄 Élément 3</div>
            <div class="enfant quatre">🐶 Élément 4</div>
            <div class="enfant cinq">🐻 Élément 5</div>
            <div class="enfant six">🐿 Élément 6</div>
            <div class="enfant sept">🦆 Élément 7</div>
            <div class="enfant huit">🦉 Élément 8</div>
            <div class="enfant neuf">🦊 Élément 9</div>
        </div>
    </body>
</html>
```

⇨ Dans *Visual Studio Code*, avec un clic droit sur l'onglet du fichier *.html,* copier le chemin de ce fichier. Ouvrir le navigateur *Firefox* et coller ce lien dans la barre d'URL afin de pouvoir lire ce fichier.

⇨ Compléter le fichier d*ecouverteFlex.css* , afin d'ajouter une bordure noire de 1px, sur tous les éléments de la page.

Normalement vous obtenez :

Élément 1
Élément 2
Élément 3
Élément 4
Élément 5
Élément 6
Élément 7
Élément 8
Élément 9

Les 9 éléments contenus dans le bloc de class *.parent* sont des éléments de type BLOCKS. Ils occupent automatiquement chacun la largeur du bloc parent qui est ici égale à la largeur de la page.

⇨ Donner la valeur flex à la propriété display pour le bloc *.parent* :

```
*{border:1px solid black;}
.parent{
    display:flex;
}
```

Normalement vous obtenez :

Élément 1 Élément 2 Élément 3 Élément 4 Élément 5 Élément 6 Élément 7 Élément 8 Élément 9

… on constate que chacun des éléments *.enfant* a pris la largeur de son contenu et s'est positionné en ligne.

⇨ Toujours sur le bloc *.parent* , donner la valeur *column* à la propriété *flex-direction* :

```
.parent{
    display:flex;
    flex-direction: column;
}
```

… on constate que les éléments .enfant se positionnent en colonne.

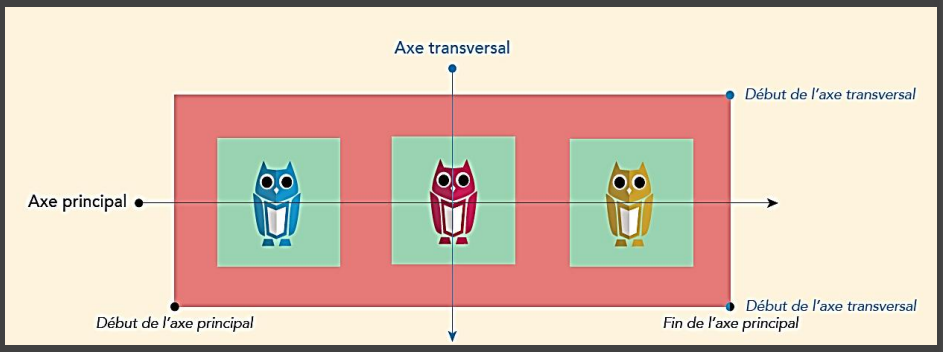
⇨ Donner à présent la valeur *row* à la propriété *flex-direction* :

```
.parent{
    display:flex;
    flex-direction: row;
}
```

… on retrouve le positionnement précédent. En effet la valeur par défaut de cette propriété *flex-direction* est *row* .

Point Cours :

L'axe principal est défini par la propriété `flex-direction:`



⇨ Toujours sur le bloc *.parent* , tester successivement les valeur *center , flex-end , flex-start , space-between* et enfin *space-around,* pour la propriété *justify-content* .

La propriété *justify-content* permet de gérer le positionnement des éléments enfants suivant l'axe principal. Dans la suite on conservera space-around :

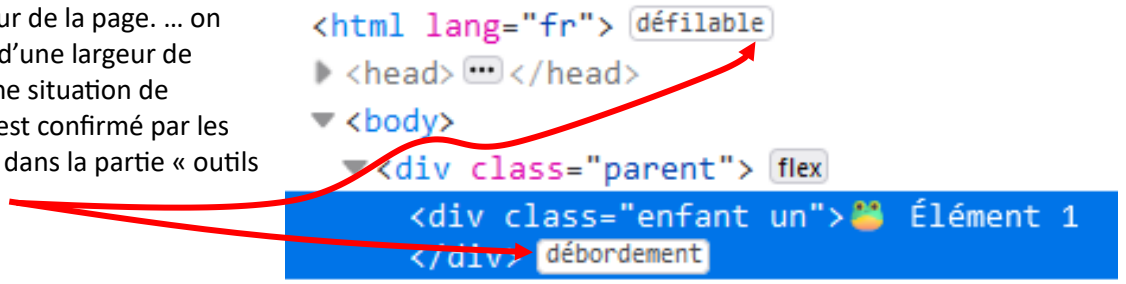
```
.parent{
    display:flex;
    flex-direction: row;
    justify-content:space-around;
}
```

⇨ Toujours sur le bloc *parent*, tester la propriété « *gap* » qui permet de définir une distance minimale entre les éléments enfants.

```
.parent{
    display:flex;
    flex-direction: row;
    justify-content:space-around;
    gap:10px;
}
```

⇨ Avec un clic droit sur la page, choisir « *inspecter* » pour ouvrir les outils du développeur et actionner la vue adaptative.

⇨ Diminuer la largeur de la page. … on constate qu'à partir d'une largeur de 600px , on est sur une situation de débordement. Cela est confirmé par les indications données dans la partie « outils du développeur » :



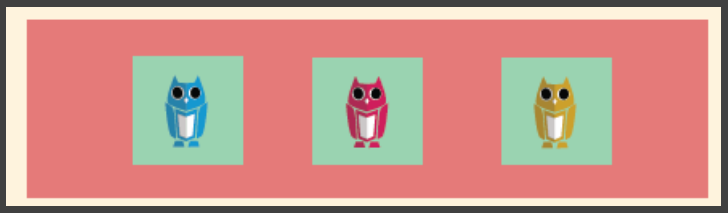
⇨ Pour autoriser un positionnement sur plusieurs lignes, il faut donner la valeur *wrap* à la propriété *flex-wrap* , toujours sur le bloc parent. … tester pour des largeurs d'écran jusqu'à 100px.

Remarque : jusqu'à présent, seul le bloc parent a été traité.

```
.parent{
    display:flex;
    flex-direction: row;
    justify-content:space-around;
    gap:10px;
    flex-wrap: wrap;
}
```

Point Cours :

- Le positionnement sur l'axe principal est géré par la propriété `justify-content:` appliquée au bloc parent



- La distance minimale entre éléments est gérée par la propriété `gap:`
- Le positionnement sur plusieurs lignes est géré par la propriété `flex-wrap:`

La largeur des différents éléments enfants est ici définie par rapport au contenu de chacun d'eux. Il est possible d'obtenir des largeurs variables afin que les éléments enfants puissent se coller les uns aux autres.

Cette possibilité est gérée par la propriété *flex* appliquée aux éléments enfants cette fois-ci :

⇨ Pour tous les éléments enfants repérés par la class ''enfant'', donner la valeur 1 à la propriété *flex* …. et observer ce qui se passe pour des tailles d'écran différentes.

```
.enfant {
    flex:1;
}
```

```
.enfant {
    flex:1;
}
.deux {
    flex:2;
}
```

⇨ tester de la même façon en donnant la valeur 2 pour cette propriété *flex* à l'élément repéré par la class ''deux'' …. On constate que cet élément là est 2 fois plus large que les autres pour lesquels *flex* est à 1.

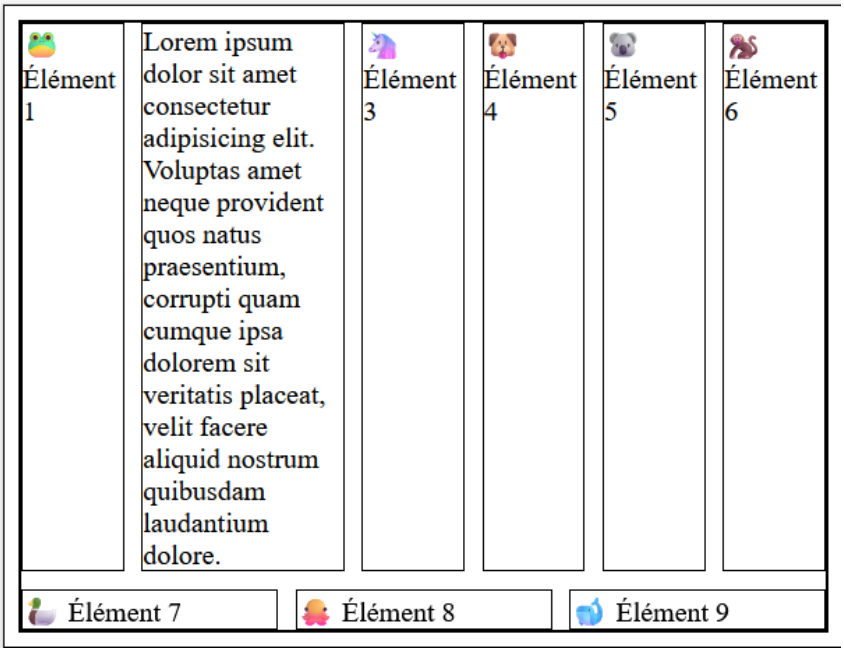
Point Cours :

- La largeur des éléments enfants peut être définie comme d'habitude avec la propriété `width:`
- Avec la flexbox, il est possible d'avoir des largeurs flexibles qui permettent aux éléments enfants de se coller. On utilise alors la propriété `flex:` sur les éléments enfants.

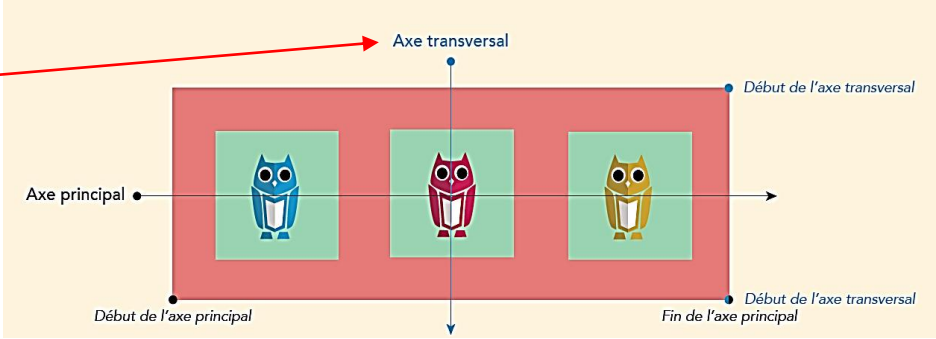
Dans notre exemple, les éléments enfants ont tous la même hauteur. Malheureusement, cette situation ne se reproduit pas souvent.

⇨ Remplacer dans le fichier html, le contenu de l'élément enfant de class ''deux'' par un lorem …

… on constate que les autres éléments enfants prennent tous la même hauteur.



Le positionnement des éléments suivant **l'axe transversal** est géré globalement par la propriété `align-items:` appliquée au bloc parent ou individuellement, sur chacun des éléments enfants, par la propriété `align-self:`



⇨ Sur le bloc parent, tester la propriété *align-items* avec les valeurs *center*, *flex-start*, *flex-end* et *stretch* (qui est la valeur par défaut).

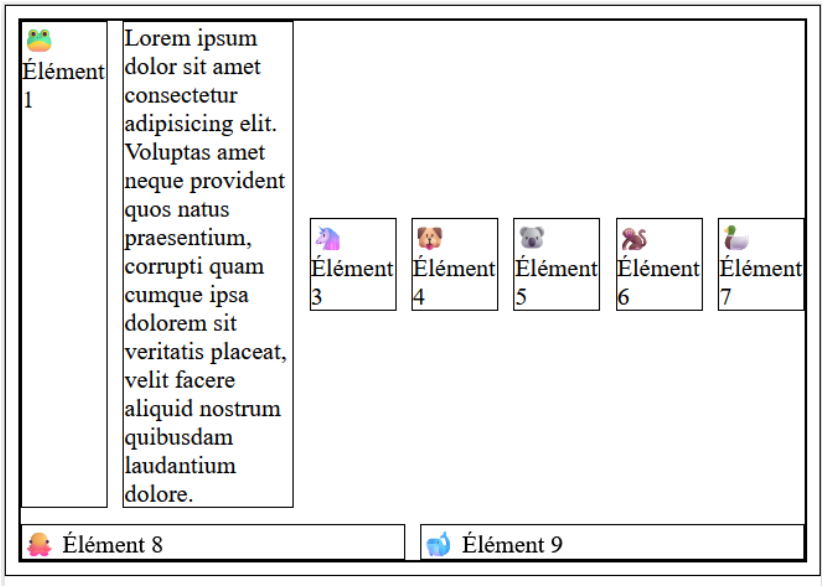
…. laisser finalement pour la suite, la valeur *center*.

```css
.parent{
    display:flex;
    flex-direction: row;
    justify-content:space-around;
    gap:10px;
    flex-wrap: wrap;
    align-items: center;
}
```

⇨ Sur un bloc enfant, par exemple celui de classe ''un'', donner la valeur *stretch* à la propriété *align-self* :

```css
.un{
    align-self: stretch;
}
```

On obtient finalement :



Point Cours :

- Pour gérer le positionnement sur l'axe transversal, on utilise la propriété `align-items:` appliqué au bloc parent.
- On peut aussi gérer ce positionnement individuellement, pour chacun des élément enfant. On applique alors sur cet élément précis, la propriété `align-self:`

Pour améliorer le coté responsive de la page, il est possible d'ajouter un média queries. Par exemple :

En donnant la valeur *column* à la propriété *flex-direction* , on définit l'axe vertical comme **axe principal**. Cela permet d'avoir une page plus adaptée aux écrans de smartphones.

```css
@media screen and (max-width: 400px) {
    .parent{
        flex-direction: column;
        align-items: stretch;
    }
    .un{
        align-self: center;
    }
}
```

2- DECOUVERTE DE « GRID » :

On se propose de travailler avec le même fichier *decouverte.html* pour obtenir à peu près la même page, mais à présent, en utilisant la méthode « ***grid*** ».

⇨ Modifier dans le fichier *decouverte.html* , le nom du fichier contenant le css :

```
<link rel="stylesheet" href="decouverteGrid.css">
```

⇨ Compléter le fichier d*ecouverteGrid.css* , afin d'ajouter une bordure noire de 1px, sur tous les éléments de la page.

Normalement vous obtenez :

Élément 1
Lorem ipsum dolor sit amet0 consectetur adipisicing elit. Voluptas amet neque provident quos natus praesentium, corrupti quam cumque ipsa dolorem sit veritatis placeat, velit facere aliquid nostrum quibusdam laudantium dolore.
Élément 3
Élément 4
Élément 5
Élément 6
Élément 7
Élément 8
Élément 9

Les 9 éléments contenus dans le bloc de class *.parent* sont des éléments de type BLOCKS. Ils occupent automatiquement chacun la largeur du bloc parent qui est ici égale à la largeur de la page.

⇨ Donner la valeur grid à la propriété display pour le bloc *.parent* .

⇨ Pour le bloc *.parent* .toujours, donner la valeur indiquée sur la figure ci-contre, à la propriété *grid-template-columns* :

```
*{border:1px solid black;}
.parent{
    display: grid;
    grid-template-columns: 1fr 2fr 1fr;
}
```

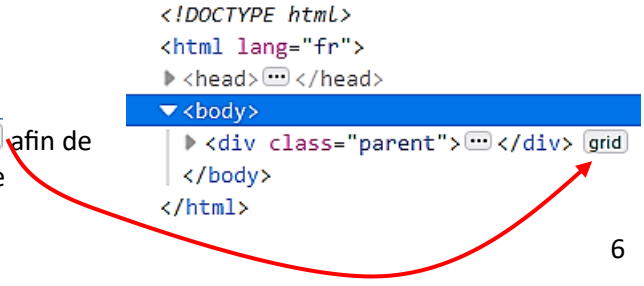
Normalement vous obtenez :

Élément 1	Lorem ipsum dolor sit amet0 consectetur adipisicing elit. Voluptas amet neque provident quos natus praesentium, corrupti quam cumque ipsa dolorem sit veritatis placeat, velit facere aliquid nostrum quibusdam laudantium dolore.	Élément 3
Élément 4	Élément 5	Élément 6
Élément 7	Élément 8	Élément 9

La valeur 1fr 2fr 1fr permet de répartir les éléments enfants dans une sorte de tableau, dans lequel la 2ième colonne est deux fois plus large que les autres.

⇨ Pour le bloc *.parent* .toujours, ajouter une propriété `gap:10px;`.

⇨ Dans les outils du développeur, cliquer sur l'icone grid afin de faire apparaitre sur la page les bordures de cette sorte de « tableau ».



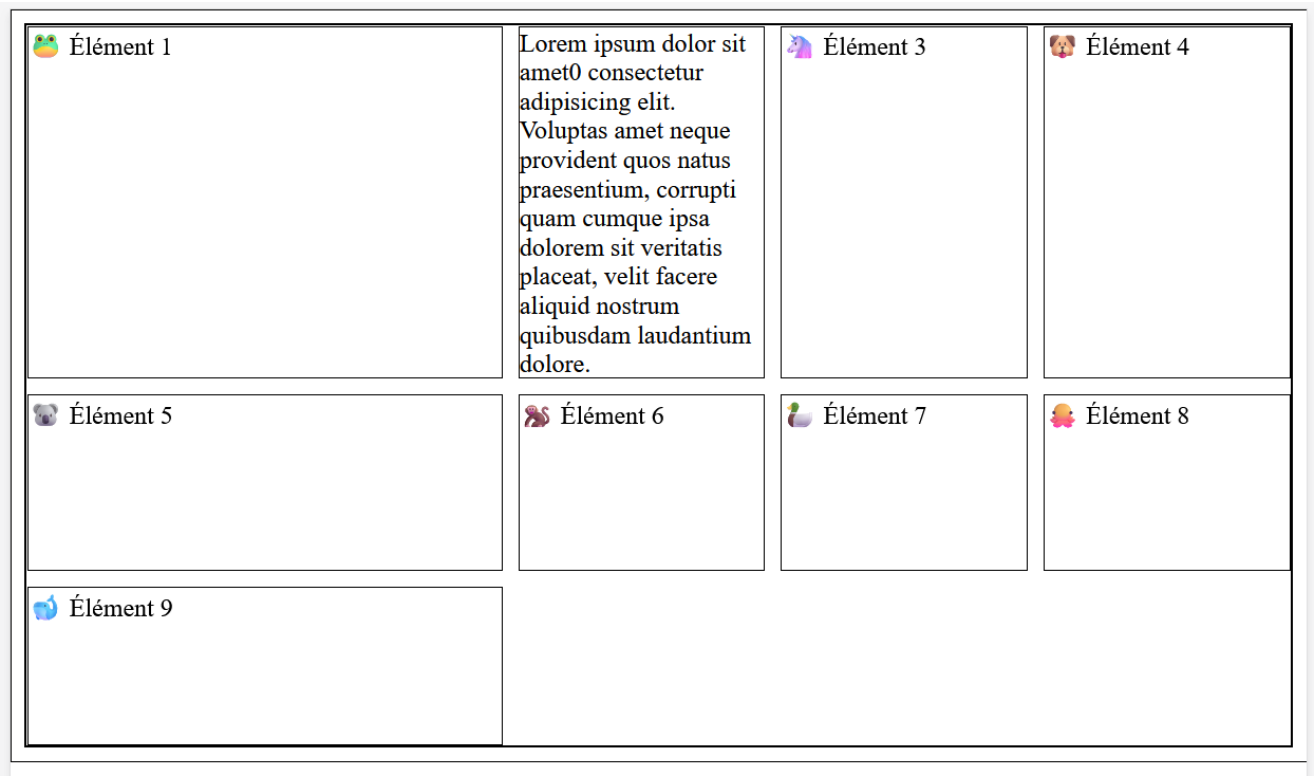
⇨ On peut définir autant de colonnes que l'on souhaite. On peut utiliser la fonction *repeat()* pour éviter les répétitions. Par exemple, tester :

```
.parent{
    display: grid;
    grid-template-columns: 300px repeat(3 , 1fr);
    gap:10px;
}
```

⇨ On peut également définir la hauteur des lignes en appliquant la propriété grid-template-rows , toujours sur le bloc parent. Tester par exemple :

```
.parent{
    display: grid;
    grid-template-columns: 300px repeat(3 , 1fr);
    gap:10px;
    grid-template-rows: 2fr 1fr 100px;
}
```

On obtient normalement les 3 lignes suivantes :



Point Cours :

- Pour gérer le nombre de colonnes et leur largeur, on peut utiliser la propriété `grid-template-columns:` sur le bloc parent.
- Pour gérer la hauteur des lignes, on peut utiliser la propriété `grid-template-rows:`, toujours sur le bloc parent.
- La propriété `gap` appliquée au bloc parent, permet de séparer les éléments d'une distance minimale.

Pour gérer le positionnement horizontal et vertical des contenus des éléments enfants, on utilise à peu près les mêmes propriétés que celles utilisées dans la flexbox vue dans le paragraphe précédent.

Point Cours :

- Pour gérer l'alignement horizontal des éléments enfants :
  - on applique la propriété `justify-items:` sur le bloc parent, pour gérer tous les éléments enfants en même temps,

  OU

  - on applique la propriété `justify-self:` sur un élément enfant particulier

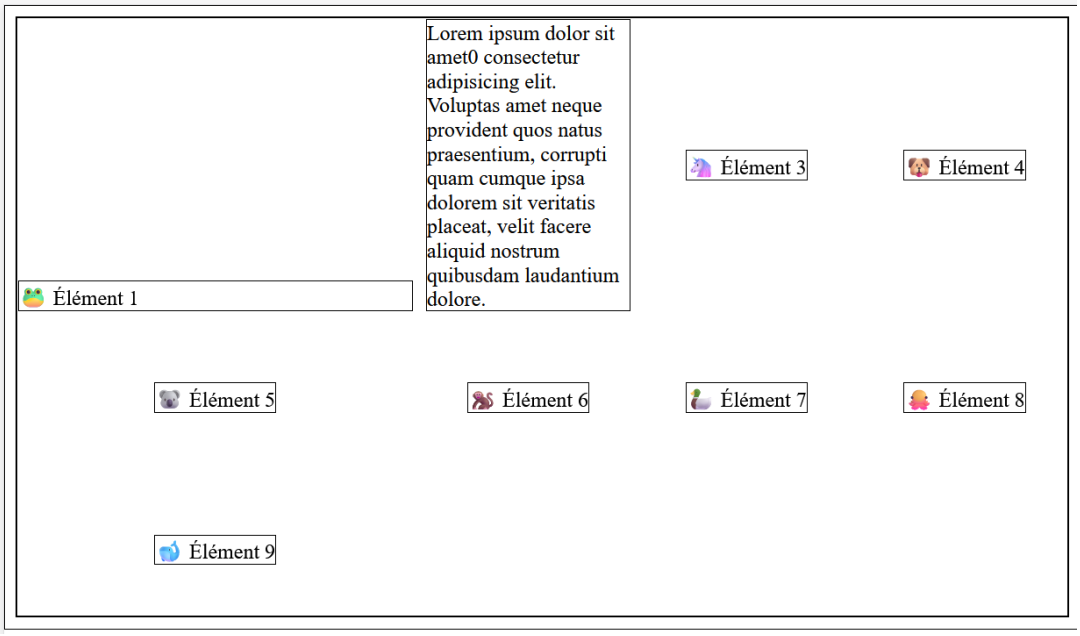
- Pour gérer l'alignement vertical des éléments enfants :
  - on applique la propriété `align-items:` sur le bloc parent, pour gérer tous les éléments enfants en même temps,

  OU

  - on applique la propriété `align-self:` sur un élément enfant particulier

⇨ Compléter le fichier css ci-contre afin d'obtenir la page ci-dessous :

```css
*{border:1px solid black;}
.parent{
    display: grid;
    grid-template-columns: 300px repeat(3 , 1fr);
    gap:10px;
    grid-template-rows: 2fr 1fr 100px;
    justify-items:
    align-items:
}
.un{
    justify-self:
    align-self:
}
```



Pour la suite, on revient au css suivant :

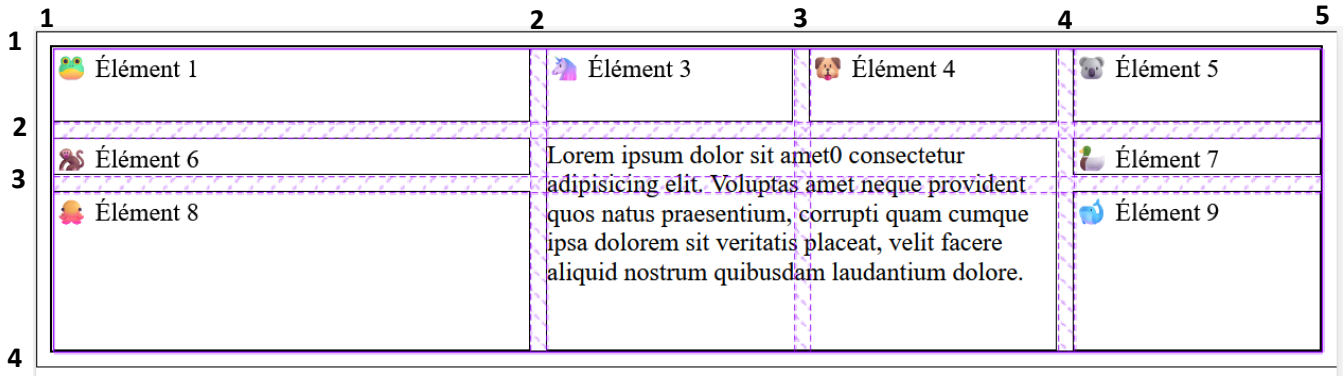
```css
*{border:1px solid ■black;}
.parent{
    display: grid;
    grid-template-columns: 300px repeat(3 , 1fr);
    gap:10px;
    grid-template-rows: 2fr 1fr 100px;
    justify-items: stretch;
    align-items: stretch;
}
```

Jusqu'à présent, chaque cellule de cette sorte de tableau que l'on appelle plutôt « *grille* » accueille 1 élément enfant. Il est possible de spécifier que certains éléments enfants doivent s'afficher sur plusieurs cellules voisines de cette grille. Il existe plusieurs façons de procéder pour obtenir ce résultat. On décrit ici la première technique que l'on apprend pour réaliser cette opération :

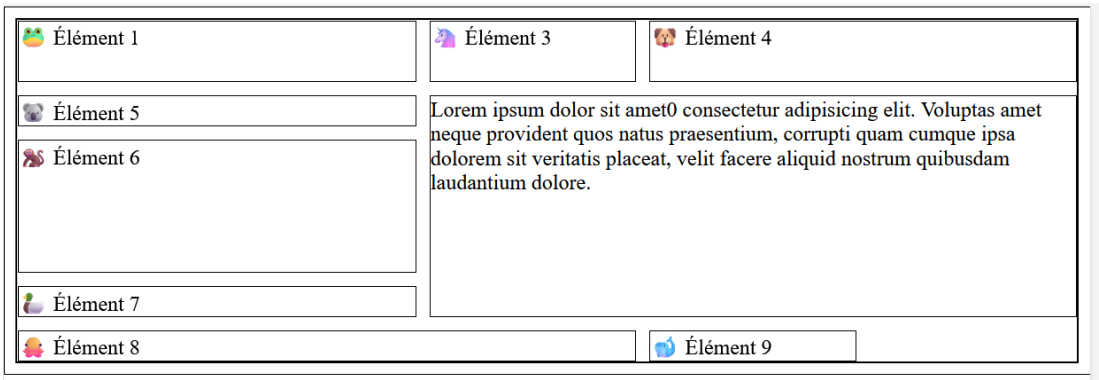
⇨ Pour l'élément enfant de class ''deux'' , on ajoute des propriétés *grid-row* et *grid-colums* en leur donnant les valeurs de la figure ci-contre :

```css
.deux{
    grid-row: 2/4;
    grid-column:2/4;
}
```

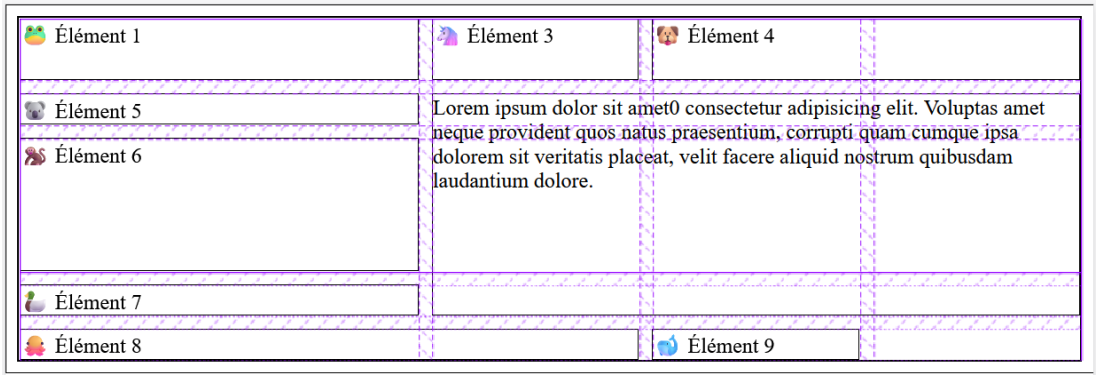
….. observer … les lignes et colonnes de la grille sont numérotées ainsi :



⇨ Modifier le fichier css pour obtenir :



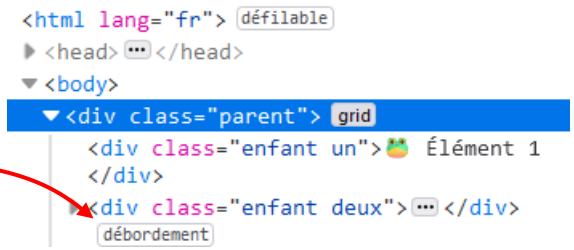
Avec la grille on a :



Point Cours :

Si l'on souhaite qu'un élément enfant se positionne sur 2 cellules voisines on utilise la propriété `grid-column:` si elles sont sur la même ligne et `grid-row:` si elles sont sur la même colonne.

⇨ En utilisant l'outil *Vue adaptative*  modifier la largeur de la fenêtre. … on constate que la structure de la grille est conservée. Ici, à partir de 450px de largeur, un débordement apparait :



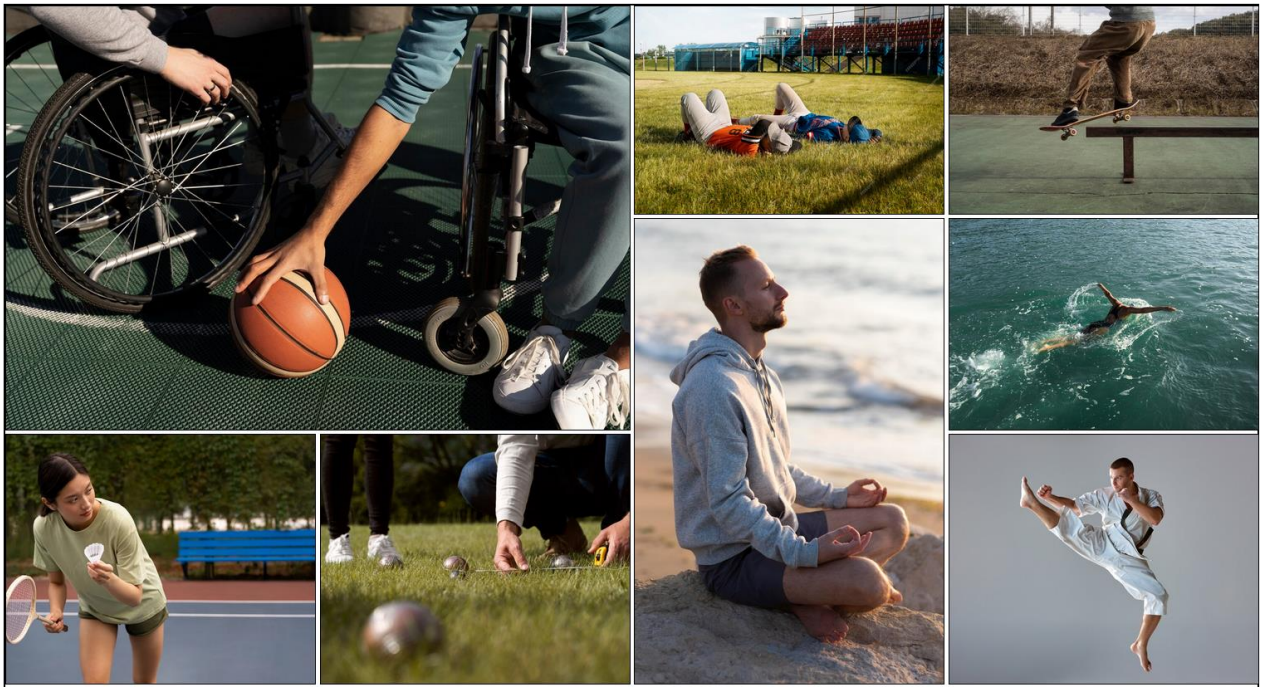
Pour que cette grille s'adapte à tous les écrans, il existe une procédure automatique qui modifie le nombre de colonnes lorsque des débordements apparaissent. Quand les structures sont complexes, autant piloter cela manuellement en ajoutant dans le css des media queries. Par exemple, ici on peut rajouter :

⇨ Compléter le css en rajoutant .. et tester.

```css
@media screen and (max-width: 600px) {
    .parent{
        grid-template-columns: repeat(3, 1fr);
    }
}
@media screen and (max-width: 350px) {
    .parent{
        grid-template-columns: repeat(2, 1fr);
    }
    .deux{
        grid-row: 1/3   ;
        grid-column:1/3;
    }
    .quatre{
        grid-column:unset;
    }
    .huit{
        grid-column:unset;
    }
}
@media screen and (max-width: 250px) {
    .parent{
        grid-template-columns: 1fr;
        grid-template-rows: unset;
    }
    .deux{
        grid-row: unset   ;
        grid-column:unset;
    }
}
```

3- MOSAIQUE DE PHOTOS AVEC « GRID » :

On se propose de réaliser la mosaïque de photos ci-dessous, en utilisant la méthode « ***grid*** ».



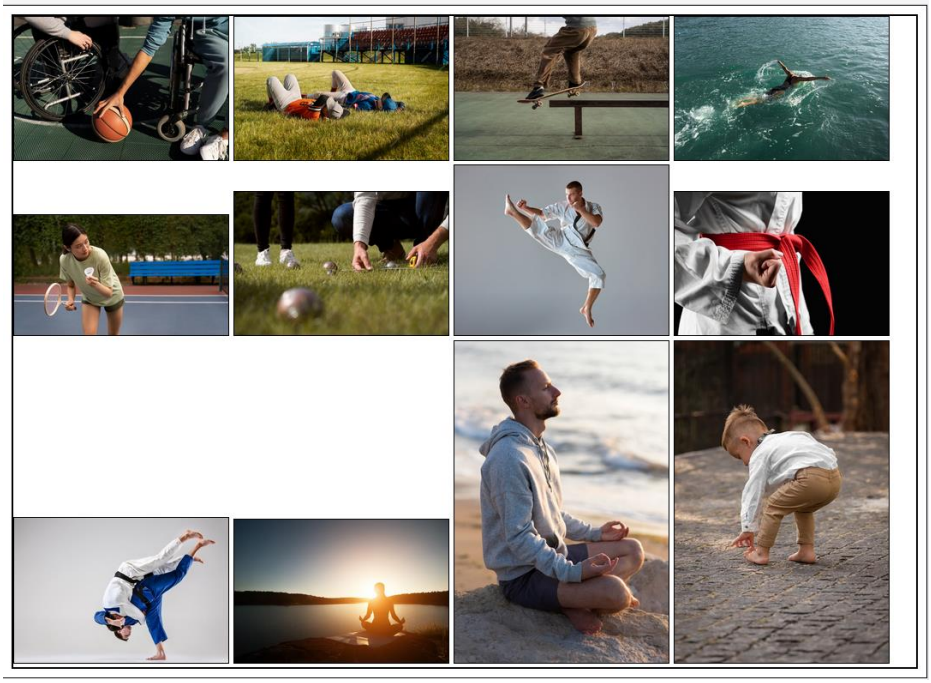
⇨ Sur nsibranly.fr , télécharger le fichier mosaique.zip (• Tp - Découverte *flexbox* et *grid* : Enoncé Mosaique zip) et l'extraire dans un dossier de travail, dans votre espace sur U:/

⇨ Ouvrir les fichiers *mosaique.html* et *mosaiqueGrid.css* sur *VisualStudioCode*. Afficher la page *mosaique.html* dans le navigateur

….. les 12 images sont de tailles importantes, elles s'alignent pour l'instant l'une sous l'autre.

⇨ On limite la largeur des images à 200px par exemple, pour mieux voir ce qui se passe :

```
*{border:1px solid black;}
img{
    width:200px;
}
```



… on constate que les images ont bien prix la même largeur de 200px, par contre les hauteurs ne sont pas les mêmes.

⇨ On impose une hauteur de 200px aussi afin de pouvoir facilement imbriquer les photos de la mosaïque :

…

on

```
img{
    width:200px;
    height:200px;
}
```

constate que le navigateur exécute bien cette dernière commande, mais en déformant les photos, car elles n'ont pas ce même format au départ.

Il existe une propriété en css qui permet de remédier à ce problème `object-fit:`. Cette propriété permet de redimensionner l'image en maintenant son ratio d'affichage. Ainsi une partie de l'image est généralement rognée pour qu'elle puisse s'adapter aux dimensions imposées.

⇨ ajouter cette propriété aux images en donnant la valeur *cover* à la propriété *object-fit* :

```css
img{
    object-fit: cover;
    width: 200px;
    height: 200px;
}
```

…. on obtient finalement une mosaïque d'images carrées de 200px par 200px. Certaines images ont été rognées, mais aucune n'a été déformée.

On se propose à présent d'utiliser la méthode css « grid ». Elle permettra en plus, de modifier la taille d'affichage pour certaines photos choisies, notamment pour celles en mode portrait.

⇨ Modifier et compléter votre css afin qu'il soit le même que celui de la figure ci-contre :

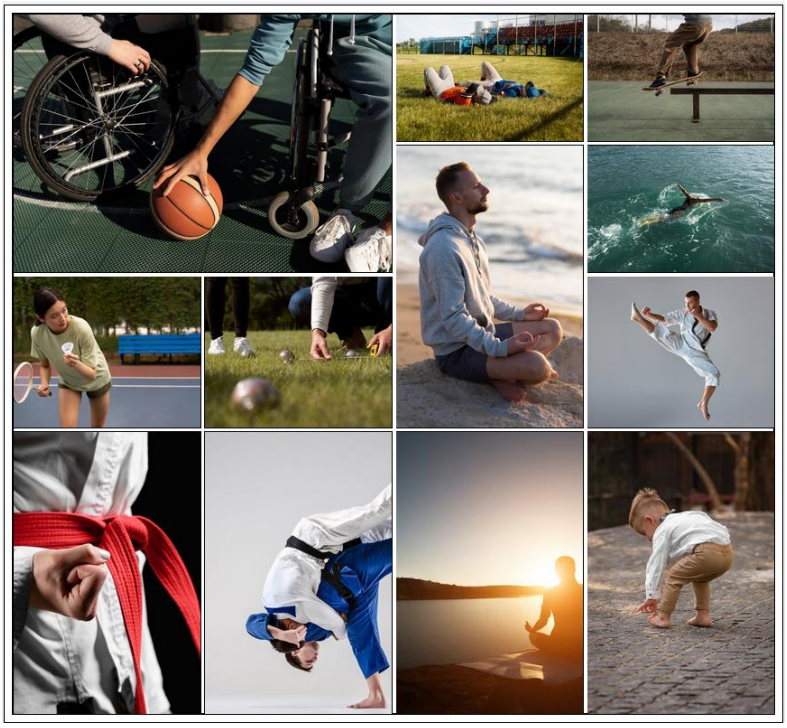
Comme l'image s'adaptera à la taille de la grille, on précise que la largeur et la hauteur des images seront égales à 100 % de celles de la cellule qui les contiendra.

On commence par une mosaïque avec 4 colonne

```css
*{border:1px solid black;}

img{
    object-fit: cover;
    width: 100%;
    height: 100%;
}

.parent{
    display: grid;
    grid-template-columns: repeat(4 ,1fr)  ;
    gap:5px;

}
```

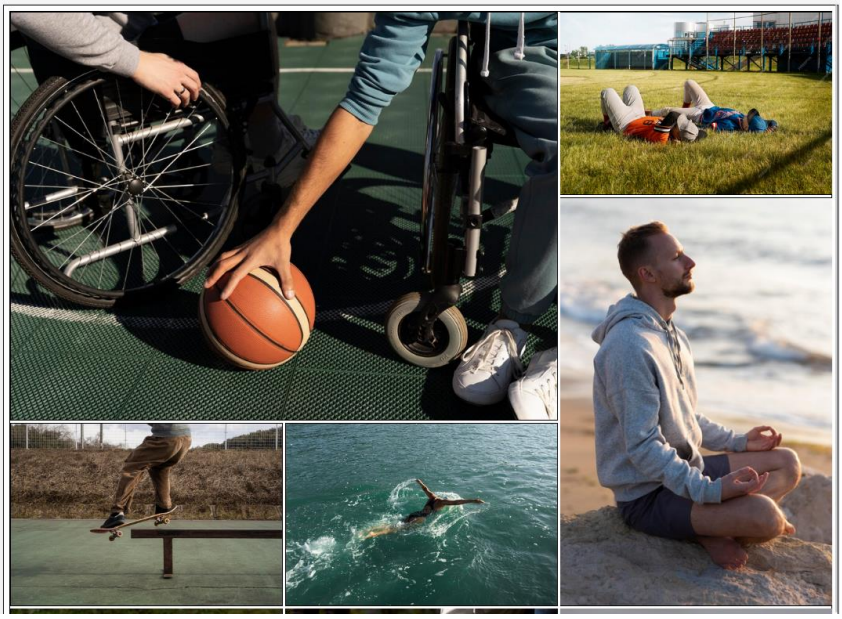
⇨ Compléter encore votre css afin d'agrandir par exemple la première image, en l'affichant sur 2 colonnes et 2 lignes :

```css
.e1{
    grid-column: 1/3;
    grid-row: 1/3;
}
```

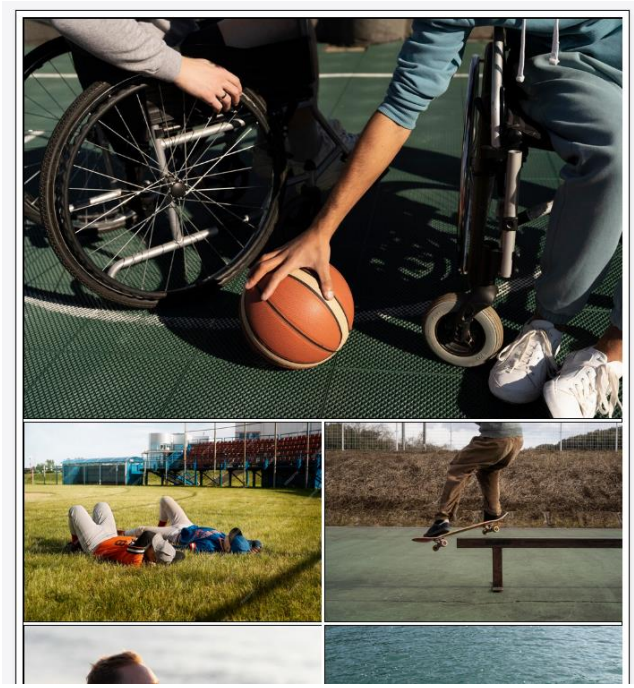


⇨ Compléter le css afin d'obtenir, la mosaïque ci-contre.

⇨ Ajouter un média queries afin d'obtenir, la mosaïque ci-contre pour une taille d'écran inférieure à 1250 px :



⇨ Ajouter un média queries afin d'obtenir, la mosaïque ci-contre pour une taille d'écran inférieure à 850 px :



⇨ Ajouter un média queries afin d'obtenir, la mosaïque ci-contre pour une taille d'écran inférieure à 400 px.