MODULE 04
Grid – variables - divers

420-C23-Web 1
Hiver 2026

Patrick Webster

Grid

Grid - introduction

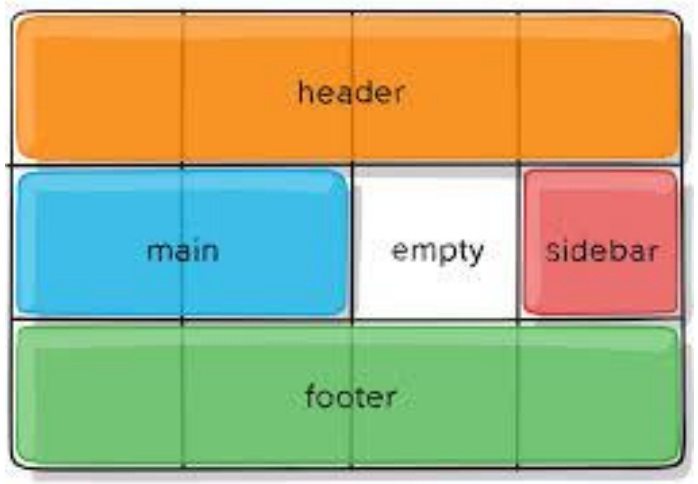
Le module CSS Grid layout (modèle de disposition en grille) permet de faire de la mise en page web en divisant des régions en colonnes et en rangées.

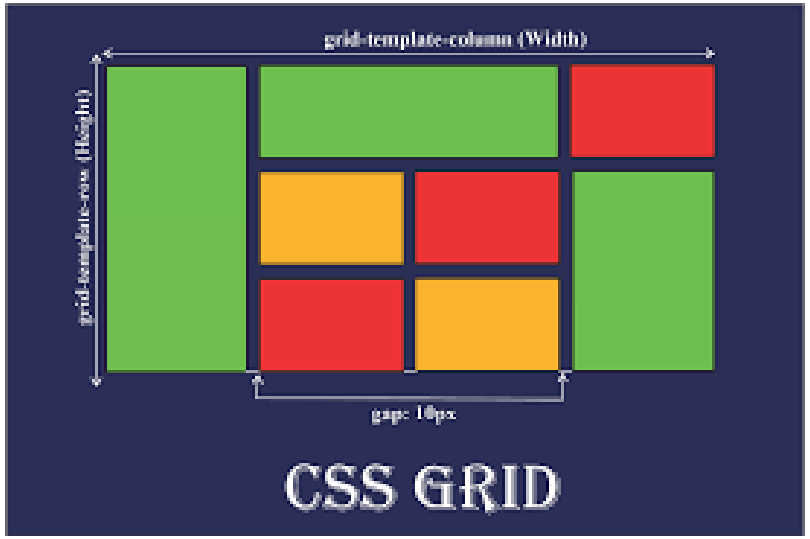
C’est un système qui permet de faire de la mise en page plus facilement et plus efficacement que des méthodes plus anciennes comme les floats et le positionnement absolu et relatif.

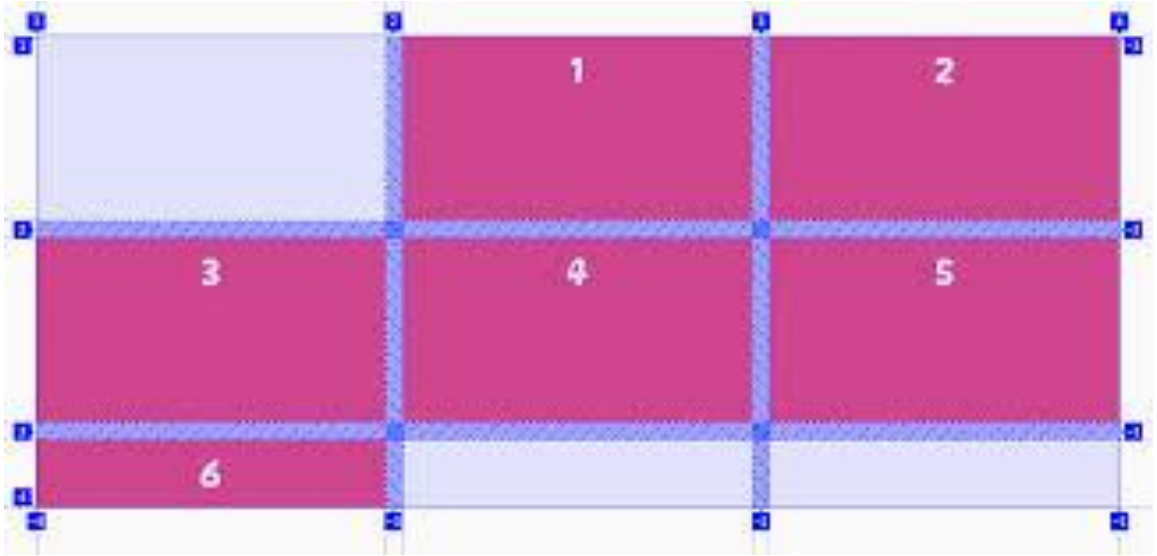
Son rôle et sa facilité d’utilisation sont semblables au FlexBox, mais il fonctionne différemment.

On peut utiliser les FlexBox et les Grids ensemble, et il y a des dispositions qui se prêtent mieux à FlexBox alors que d’autres sont plus facilement conçues avec Grid. Les deux technologies sont en utilisation à l’heure actuelle.

Grid - introduction







Grid - fonctionnement

On définit un conteneur en Grid avec la propriété:

```
display: grid;
```

On définit ensuite les caractéristiques de chacune des colonnes avec :

```
grid-template-columns: ...  ...;
```

Et les caractéristiques de chacune des rangées avec :

```
grid-template-rows: ...  ...;
```

Puis on place simplement les éléments pour qu'ils occupent le nombre de cellules désirées avec:

```
grid-column-start: …;
grid-column-end: …;
grid-row-start: …;
grid-row-end: …;
```

Grid – démo 1

Grid – exemple 1

Télécharger le document C23_D04A-F_Démos Module 04-Étudiants H26.zip (sur Léa)

Extraire le contenu dans un répertoire sur votre X:.

Dans le dossier D04A_Démo-Grid-1, ouvrir le fichier d04a-demo-grid-1.html avec VS Code.

Soit la structure suivante:

```
<div class="contenant">
        <div>Section 1</div>
        <div>Section 2</div>
        <div>Section 3</div>
        <div>Section 4</div>
        <div>Section 5</div>
</div>
```

Grid – exemple 1

On change le type de contant pour grid:

```
.contenant{
        width:960px;
        background-color:burlywood;
        margin: 0px auto;
        display: grid;
}
```

- Tous les div dans le contenant sont maintenant des items de grid.
- Les items sont toujours les uns sous les autres.

Grid – exemple 1

Si on ne spécifie pas les colonnes et rangées, celles-ci sont créées implicitement d'après les éléments enfants.

Dans notre exemple:

- 1 colonne et 5 rangées

Grid – exemple 1

Spécifier les colonnes et rangées explicitement:

Colonnes:

```
grid-template-columns: ...  ...;
```

Rangées:

```
grid-template-rows: ...  ...;
```

Grid – exemple 1

On spécifie 3 colonnes, de taille fixe:

```
.contenant{
        width:960px;
        height: 100px;
        background-color:burlywood;
        margin: 0px auto;
        display: grid;
        grid-template-columns: 300px 200px 200px;
}
```

- On a maintenant une grille de 3 colonnes et 2 rangées.
- Les éléments enfants se sont placés successivement dans chaque cellule.

Grid – exemple 1 - unités

On peut définir la taille des colonnes avec les unités habituelles.

Grid introduit une nouvelle unité: fr.

- fr est une fraction de l'espace disponible dans un contenant grid.

Quelques exemples:

› `grid-template-columns: 1fr 1fr 1fr;`
  - divise en 3 colonnes égales

› `grid-template-columns: 2fr 1fr 1fr;`
  - divise en 3 colonnes, col 1 = 2 X col 2 et 3

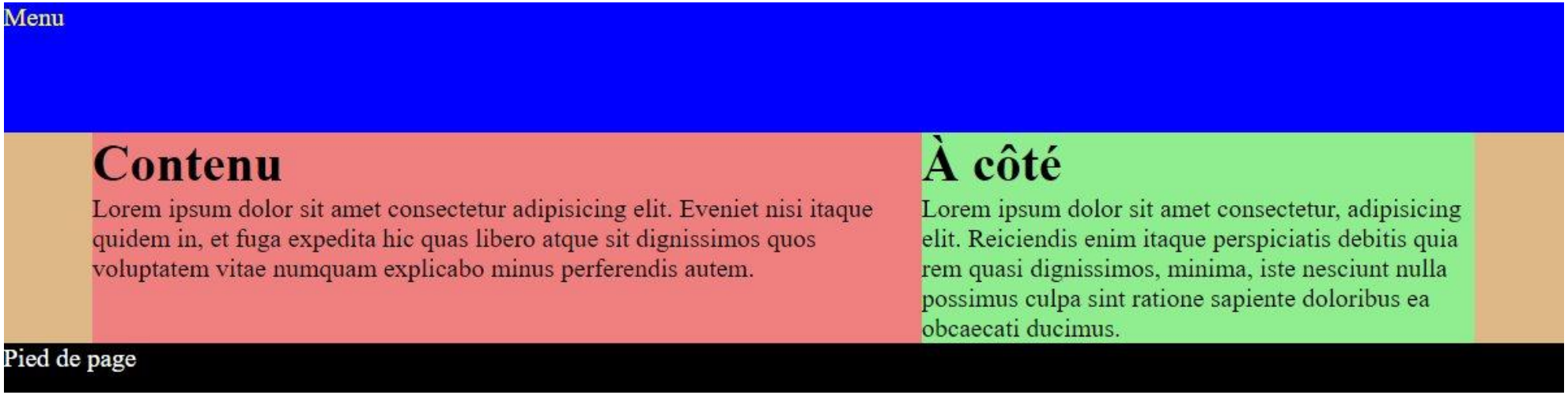
› `grid-template-columns: 100px 1fr 1fr 1fr 50px;`
  - divise en 5 colonnes: col1 = 100px, col5 = 50px, col 2-4, se partagent également l'espace restant (largeur parent - 150px)

Grid – démo 2

Grid – exemple 2

Dans le répertoire D04B_Démo-Grid-2, ouvrez le fichier d04b-demo-grid-2.html dans VS Code.

On veut faire la disposition classique suivante, avec la section contenu qui fait 60% de l'espace disponible.



Grid – exemple 2

On commence avec un display en grid en spécifiant la largeur des colonnes: 5vw pour l'espace à gauche et à droite, et le restant divisé en 60% et 40%.

```
.contenant{
        width:960px;
        background-color:burlywood;
        margin: 0px auto;
        display: grid;
        grid-template-columns: 5vw 3fr 2fr 5vw;
}
```

Pas tout à fait le résultat attendu: chaque élément est placé dans la première cellule disponible.

Grid – exemple 2

Pour qu'un élément occupe plusieurs colonnes:

- grid-column-start et grid-column-end, configuré pour l'élément enfant.
- Start et end sont basés sur les **lignes** de la grille, et non sur les cellules des colonnes: 4 colonnes => 5 lignes

```
.header{
        background-color: blue;
        color: yellow;
        grid-column-start: 1;
        grid-column-end: 5;         }
.contenu{
        background-color: lightcoral;
        grid-column-start: 2;
        grid-column-end: 3;         }
```

```
.acote{
        background-color: lightgreen;
        grid-column-start: 3;
        grid-column-end: 4;         }
.footer{
        background-color:black;
        color:white;
        grid-column-start: 1;
        grid-column-end: 5;         }
```

Grid – exemple 2

Jusqu'à présent, les rangées de la grille ont été définies implicitement, en fonction des éléments et de leur placement.

Pour définir les colonnes explicitement: grid-template-rows.

```
.contenant{
        width:960px;
        background-color:burlywood;
        margin: 0px auto;
        display: grid;
        grid-template-columns: 5vw 3fr 2fr 5vw;
        grid-template-rows: 80px 1fr 30px;
}
```

Grid – exemple 2

Si on définit le contenant comme occupant 100% de hauteur (body et html sont déjà 100vh)

```
.contenant{
        width:960px;
        background-color:burlywood;
        margin: 0px auto;
        height: 100%;
        display: grid;
        grid-template-columns: 5vw 3fr 2fr 5vw;
        grid-template-rows: 80px 1fr 30px;
}
```

Le 1fr de la rangée du milieu s’ajuste automatiquement.

Grid – exemple 2

On veut ajouter une rangée juste avant le footer, 1/4 de la hauteur entre le header et le footer:

```
.contenant{
        width:960px;
        background-color:burlywood;
        margin: 0px auto;
        height: 100%;
        display: grid;
        grid-template-columns: 5vw 3fr 2fr 5vw;
        grid-template-rows: 80px 3fr 1fr 30px;
}
```

Le 1fr de la rangée du milieu s'ajuste automatiquement.

Mais le *footer* est en avant dernière rangée, on veut plutôt insérer une rangée avant le *footer*.

Grid – exemple 2

Pour spécifier explicitement les rangées occupées par un élément:

grid-row-start et grid-row-end, configuré pour l'élément enfant.

Start et end sont basés sur les lignes de la grille, et non sur les cellules des colonnes:
4 rangées => 5 lignes

```
.footer{
        background-color:black;
        color:white;
        /* height: 30px; */
        grid-column-start: 1;
        grid-column-end: 5;
        grid-row-start: 4;
        grid-row-end:5;
}
```

Une rangée peut ne pas avoir d'éléments, tout comme une colonne.

Grid – espacement entre les cellules

Pour les espaces sur une rangée:

› column-gap: 1vw; /* autres unités valides */

Pour les espaces sur une colonne:

› row-gap: 10px; ; /* autres unités valides */

Forme raccourcie: (row-gap column-gap)

gap: 5px 2% ; /* autres unités valides */

ou

gap: 5px; /* pour row-gap ET column-gap */

Grid – espacement

Si on veut de l'espacement entre le contenu et le aside:

```
.contenant{
            width:960px;
            background-color:burlywood;
            margin: 0px auto;
            height: 100%;
            display: grid;
            grid-template-columns: 5vw 3fr 2fr 5vw;
            grid-template-rows: 80px 3fr 1fr 30px;
            gap: 10px;
        }
```

Grid – espacement

Si on ne veut pas d'espacement vertical:

```
.contenant{
            width:960px;
            background-color:burlywood;
            margin: 0px auto;
            height: 100%;
            display: grid;
            grid-template-columns: 5vw 3fr 2fr 5vw;
            grid-template-rows: 80px 3fr 1fr 30px;
            gap: 0px 10px;
        }
```

Grid – exemple 2

Les éléments peuvent occuper un nombre de colonnes et cellules définies, sans lien avec les autres éléments!

```
.acote{
	background-color: lightgreen;
	grid-column-start: 3;
	grid-column-end: 4;
	grid-row-start: 2;
	grid-row-end: 5;
}
```

Grid – exemple 2

Les éléments peuvent même se chevaucher dans la grille…

```
.contenu{
	background-color: lightcoral;
	grid-column-start: 2;
	grid-column-end: 5;
}
.acote{
	background-color: lightgreen;
	grid-column-start: 3;
	grid-column-end: 4;
	grid-row-start: 1;
	grid-row-end: 5;
}
```

Avec ces changements seulement, le résultat n'est pas celui attendu. Certaines coordonnées n'ont pas été définies explicitement et le navigateur décide par lui-même de la priorité du placement.

Grid – exemple 2

On définit explicitement les colonnes et rangées pour tous les éléments de la grille:

```
.contenu{
        background-color: lightcoral;
        grid-column-start: 2;
        grid-column-end: 5;
        grid-row-start: 2;
        grid-row-end: 3;          }
```

```
.header{
        background-color: blue;
        color: yellow;
        grid-column-start: 1;
        grid-column-end: 5;
        grid-row-start:1;
        grid-row-end:2;      }
```

Voilà!

Grid – exemple 2

On peut définir quel élément apparait devant les autres avec z-index.

Plus le z-index est grand, plus il est « par-dessus » ou « en avant ».

La valeur par défaut du z-index (tous les éléments par défaut) est 0.

```
.contenu{
        background-color: lightcoral;
        grid-column-start: 2;
        grid-column-end: 5;
        grid-row-start: 2;
        grid-row-end: 3;
        z-index: -1;          }
```

```
.acote{
        background-color:
lightgreen;
        grid-column-start: 3;
        grid-column-end: 4;
        grid-row-start: 1;
        grid-row-end: 5;
        z-index: 1;     }
```

z-index fonctionne aussi avec Flex…

Grid – forme raccourcie

La forme raccourcie permet d'inclure toutes les coordonnées d'un élément grid sur une ligne, séparées par un /.

Pour la grille au complet:

```
grid: 80px 3fr 1fr 30px / 5vw 3fr 2fr 5vw ;
```

Identique à:

```
grid-template-columns: 5vw 3fr 2fr 5vw;
grid-template-rows: 80px 3fr 1fr 30px;
```

Grid – forme raccourcie – grid-area

Pour les éléments:

› grid-area: grid-row-start / grid-column-start / grid-row-end / grid-column-end;

grid-area: 3 / 1 /4 / 2;

Identique à

```
› grid-column-start: 1;
› grid-column-end: 2;
› grid-row-start: 3;
› grid-row-end: 4;
```

Propriétés communes avec FlexBox

Les conteneurs et des items GridBox supportent plusieurs des propriétés communes avec FlexBox comme:

- `justify-content`
- `align-items`
- `. . .`

Et plusieurs autres que vous êtes invités à explorer lors du projet.

Grid vs FlexBox

La mise en page d'une page web au complet ne correspond pas nécessairement à une grille.

On peut utiliser grid ET Flex dans une même page.

Par exemple:

- La page en flex
- Un header en flex
- Un footer en flex
- La partie centrale en grid (pas toute la page en contenant grid, seulement une partie)

Grid est plus approprié lorsque des sections d'une page entrent bien dans une disposition quadrillée.

Grid - requêtes média

Grid est réactif de par sa nature.

L'unité fr améliore encore plus ce comportement.

Grid fonctionne aussi avec des requêtes média.

On redéfinit alors les rangées, colonnes et la position des éléments d'une taille d'écran à l'autre.

Grid - autres

Il y a beaucoup plus à explorer avec grid. Voici quelques exemples:

- Utiliser repeat pour définir les colonnes/rangées
  `repeat(3, 1fr)` => 3 colonnes/rangées de 1 fr chacune

- Utiliser minmax() pour donner un minimum et maximum à une valeur:
  `minmax(100px, auto)` => minimum 10px, pas de maximum

- Utiliser column-gap/row-gap pour définir des marges entres les éléments de la grille:

```
column-gap: 10px;
row-gap: 1em;
```

- Utiliser subgrid pour créer des grilles dans des grilles

Et plus…

Planification du projet final

Sujet à remettre et élaborer au prochain cours

GRID

Laboratoire 04A

Réalisez le laboratoire L04A-Grids.

Fonction calc()

Fonction calc()

CSS inclus la fonction calc() qui permet de faire les opérations mathématiques suivantes: + - * /

On peut l'utiliser dans différents contextes.

Exemples d'utilisation pour calcul d'unités:

- `width: calc(100px + 50px); /* Largeur de 150px */`
- `min-width: calc(100% / 7); /* 1 septième de la largeur du parent */`
- `height: calc(100vw – 10vw); /* pour pleine hauteur moins padding 5vw en haut et en bas) */`
- `Width: calc(2 * 50px * 3.14159265358979) /* Une largeur assez grande pour un cercle de 50px de rayon */`

Fonction calc() – différentes unités

Une des caractéristiques de calc() les plus utiles est le mélange d'unités différentes dans un même calcul.

Exemples:

- `width: calc(50% - 20px); /* moitié du conteneur parent moins un paddidng fixe */`
- `width: calc(25% - 1.5vw);  /* 25% du conteneur parent – moins du padding réactif */`
- `font-size: calc(10px + 1vw); /* police avec une taille minimale */`

Fonction calc() – caractéristiques et limites

- Permet le mélange d'unités
- Peut être imbriqué (calc dans un calc)
- Division par 0 génère une erreur
- Seuls les navigateurs les plus récents supportent calc dans des requêtes média – https://caniuse.com/mdn-css_at-rules_media_calc
- Il DOIT y avoir des espace autours du + et du –
- L'espace n'est pas obligatoire autour du * et /, mais suggéré par meilleure pratiques

Fonction calc() – caractéristiques et limites

Dans le répertoire D04CDE_Démos-Calc, ouvrez les fichiers:

- d04c-demo-calc-1.html
- d04d-demo-calc-2.html
- d04e-demo-calc-3.html

dans VS Code.

Faire les démonstrations.

CSS - Variables

CSS - variables

CSS permet l'utilisation (limitée) de variables.

Déclarer une variable se fait avec --, à l'intérieur d'un style:

```
--nom_variable
--nom_variable: valeur_variable;
```

Utiliser une variable se fait avec la fonction var():

```
var(--nom_variable)
```

CSS – variables exemples

Déclaration de variables:

```
--bleu-nuit: #191970; /* Une couleur */
--marge-page: 20px; /* en pixels */
--padding-section: 2vw; /* en vw */
--largeur-menu: 80%; /* en % */
--nombre-sections: 5; /* nombre sans unités */
```

Utilisation de variables

```
color: var(--bleu-nuit);
width: var(--marge-page);
padding: var(--padding-section);
width: var(--largeur-menu);
```

CSS – variables – avec calc()

On ne peut pas faire des opérations mathématiques directement avec les variables en css, mais on peut les manipuler avec calc().

```
--rayon: 20;
--pi: 3.14159265358979; /* un nombre fixe, sans unité */

width: calc(2 * var(--pi) * var(--rayon) * 1px)
```

Produit une largeur de la taille d'un cercle avec un rayon de 20 px.

Le * 1 px sert à transformer une valeur sans unité en unité de px, requis par width.

CSS – variables - domaine

Les variables ont un domaine.

Si déclarées dans un sélecteur, accessible dans ce sélecteur et en dessous (conteneurs enfants).

CSS – variables – domaine - exemple

Soit le css:

```
.parent{
    --bleunuit: #191970;
    background-color: var(--peche);
    color:var(--bleunuit) }
.parent div{
    --peche: #ecbd90; }
```

… et le HTML

```
<div class="parent">
    <div>Enfant 1</div>
    <div>Enfant 2</div>
</div>
```

Le texte sera bleu nuit, parce que déclaré dans un conteneur parent.

Le fond ne sera pas pêche parce la variable existe dans un conteneur enfant et ne peut être utilisée dans un conteneur parent.

Si on veut des variables **globales**, on les déclare dans *****, ou **:root** ou **html**.

CSS – variables - démonstration

Dans le répertoire D04F_Démo-Var-1, ouvrez le fichier d04f-demo-var-1.html dans VS Code.

Faire la démonstration.

GRID

Laboratoire 04B

Réalisez le laboratoire L04B_Calc-Var.

Propriété position

Position: syntaxe

La propriété position permet de disposer des éléments dans la page par rapport à d'autres facteurs que le flux régulier.

Les valeurs possibles pour position sont:

- static
- relative
- absolute
- fixed
- sticky

La valeur static est le comportement par défaut des éléments dans le flux de la page, identique à e pas avoir de propriété position.

Propriétés utilisées avec position

top: décalage de l'élément par rapport au haut d'un autre élément

bottom: décalage de l'élément par rapport au bas d'un autre élément

left: décalage de l'élément par rapport à la gauche d'un autre élément

right: décalage de l'élément par rapport à la droite d'un autre élément

z-index: emplacement de l'élément en profondeur (devant/derrière)

position: relative

```
position: relative
```

L'élément fait partie du flux normal

L'élément est positionné par rapport à lui-même (sa disposition normale)

Son emplacement est d'abord calculé selon sa position normale puis décalé selon les valeurs de top, bottom, left et right.

Les autres éléments de la page interagissent avec cet élément selon sa disposition avant le décalage.

L'élément apparait devant les autres éléments.

position: absolue

`position: absolute`

L'élément est sorti du flux normal.

L'élément est positionné par rapport à son conteneur parent, décalé selon les valeurs de top, bottom, left et right.

Les autres éléments de la page n'interagissent pas avec cet élément, ils ignorent cet élément.

L'élément apparait devant les autres éléments.

L'élément peut avoir des marges, celles-ci n'interagissent avec les autres éléments du flux.

position: fixed

`position: fixed`

L'élément est sorti du flux normal.

L'élément est positionné par rapport au viewport, décalé selon les valeurs de top, bottom, left et right.

Un item fixed reste au même endroit même quand on défile la page.

Les autres éléments de la page n'interagissent pas avec cet élément, ils ignorent cet élément.

L'élément apparait devant les autres éléments.

L'élément peut avoir des marges, celles-ci n'interagissent avec les autres éléments du flux.

position: sticky

`position: sticky`

L'élément fait d'abord partie du flux normal, puis il est décalé selon son ancêtre de défilement immédiat ET son bloc englobant.

Un item fixed reste au même endroit même quand on défile la page.

L'élément apparait devant les autres éléments.

Interagit avec overflow (qui définit un ancêtre de défilement).

Un élément sticky restera toujours visible à l'intérieur de son bloc de défilement, à mesure qu'on défile ce bloc.

Position - démonstration

Dans le répertoire D04G_Démo-position-1, ouvrez le fichier d04g-demo-position-1.html dans VS Code.

Faire la démonstration.

Décommenter successivement les commentaires selon leur numéro et constatez le résultat produit.

Exemples de cas d'utilisations

- Éléments superposés : placer des éléments devant ou derrière d'autres éléments (z-index)
- Modal (message)
- Modal (Chatbox)
- Sticky menu (toujours présent)
- Déplacements d'items à travers l'écran (comme un jeu, combiné avec du javascript)
- *Drop-down* menus
- Faire défiler des éléments dans un conteneurs
- Tooltips (bulle d'info accrochée à un conteneur, sur hover)

Ref d'exemples : https://medium.com/@jatindertech/absolute-and-relative-css-positions-with-these-10-use-cases-bb806a8195d4

Désavantages

L’utilisation de la propriété position doit être réservée à des utilisations spécifiques qui ne peuvent pas être résolues par des propriétés css plus communes.

Des comportements inattendus peuvent survenir entre les éléments de la page et plusieurs désavantages sont à considérer:

1. Peut cacher des éléments de la page
2. Comportements inattendus en
   - changement de taille de navigateur
   - autre taille d’écran
   - zoom
   - accessibilité

GRID

Laboratoire 04C

Réalisez le laboratoire L04C_Position.

Sélecteurs d’attributs

Sélecteurs d’attributs

Il existe une gamme de sélecteurs utilisant spécifiquement les attributs d’éléments HTML. Ils utilisent les "[...]" avec plusieurs variantes pour effectuer des sélections flexibles selon l’attribut attribut :

- `balise[attribut]` sélectionne les balises identifiées ayant défini l’attribut indiqué
- `balise[attribut="valeur"]` balise dont la valeur de l’attribut est égale à valeur
- `balise[attribut~="valeur"]` contient valeur en mot entier, avec ou sans espaces
- `balise[attribut|="valeur"]` valeur d’attribut est valeur ou bien valeur et un -
- `balise[attribut^="valeur"]` valeur d’attribut débutant par valeur
- `balise[attribut$="valeur"]` valeur d’attribut se terminant par valeur
- `balise[attribut*="valeur"]` valeur d’attribut contient valeur

Le nom de la balise n’est pas nécessaire.

Le comportement s’appliquera à toutes les balises dont les attributs correspondent.

Sélecteurs d’attributs - utilité

Tous les hyperliens qui pointent vers un lien dans le même document sont de couleur argentée:

```
a[href^="#"]{
    color: silver;  }
```

Tous les hyperliens qui incluent « cvm » sont en rouge:

```
a[href*="cvm"]{
    color: red;  }
```

Sélecteurs d'attributs – utilité (suite)

Tous les hyperliens qui ouvrent dans un nouvel onglet sont en bleu:

```
a[target="_blank"]{
    color: blue;  }
```

Tous les éléments dont le nom de classe est menu ou commence avec menu- ont un arrière-plan noir:

```
a[class|="menu"]{
    background-color: black;  }
```

Sélecteurs d'attributs – utilité (suite)

Tous les hyperliens qui ouvrent dans un nouvel onglet sont en bleu:

```
a[target="_blank"]{
    color: blue;  }
```

Tous les éléments dont le nom de classe est menu ou commence avec menu- ont un arrière-plan noir:

```
a[class|="menu"]{
    background-color: black;  }
```

Préparation au projet final

Sujet et sous-sujets, choix des pages, choix des sections

Travail Pratique 4 (TP4)

mini-quiz à faire, les autres parties (B et C) sont intégrées au Projet Final