Le modèle des boîtes flexibles

Rédigé par: Jean-Luc Colson			
Date première rédaction: Janvier 2020			
SUIVI DES MODIFICATIONS A LA FICHE			
Série	Date	Page(s) modifiée(s)	Raison

Contents

Introduction au modèle des boites flexibles ou flexbox 5
Définition du modèle des boites flexibles ou flexbox 5
Conteneur et éléments flexibles 6
Axe principal et axe secondaire des boites flexibles 7
Gérer la direction des éléments flexibles ou flex items 9
L'axe principal et la propriété flex-direction 9
Gérer les dépassements des éléments flexibles du conteneur 11
La notation raccourcie flex-flow 14
L'ordre d'affichage des éléments flexibles dans le conteneur 15
Gérer l'alignement des éléments flexibles 17
Gérer l'alignement des éléments selon l'axe principal avec justify-content 17
Gérer l'alignement des éléments selon l'axe secondaire 22
Gérer l'alignement des éléments dans une ligne ou dans une colonne selon l'axe secondaire avec align-items 22
Les propriétés flex-wrap et align-items 25
Gérer l'alignement des lignes ou des colonnes les unes par rapport aux autres selon l'axe secondaire avec align-content 27
Gérer l'alignement d'un élément en particulier dans son axe secondaire avec align-self 29
Gérer la taille et la flexibilité des éléments flexibles 31
Définir une taille de départ pour nos éléments flexibles avec flex-basis 31
Gérer la capacité des éléments flexibles à grossir avec flex-grow 33
Gérer la capacité des éléments flexibles à rétrécir avec flex-shrink 35
La notation raccourcie flex 36
Cas d'utilisation et limites du flexbox 38
Exemples concrets d'utilisation du flexbox 38
Effectuer un centrage parfait 38
Créer un menu avec le flexbox 39
Grouper des éléments 40
Styliser les formulaires 41
Les limites du flexbox 42

Introduction au modèle des boites flexibles ou flexbox

Dans ce cours, nous avons vu que les éléments HTML pouvaient être affichés de différentes manières et que le type d'affichage conditionnait la disposition par défaut d'un élément :

- Affichage en bloc avec display : block ;
- Affichage en ligne avec display : inline ;
- Affichage sous forme de tableau avec display : table ;
- Etc.

Le but de cette partie est de vous présenter un nouveau mode d'affichage et de disposition très puissant : la disposition selon un modèle de boites flexibles ou flexbox avec display : flex (ou display : inline-flex) et les différentes propriétés CSS liées à ce modèle.

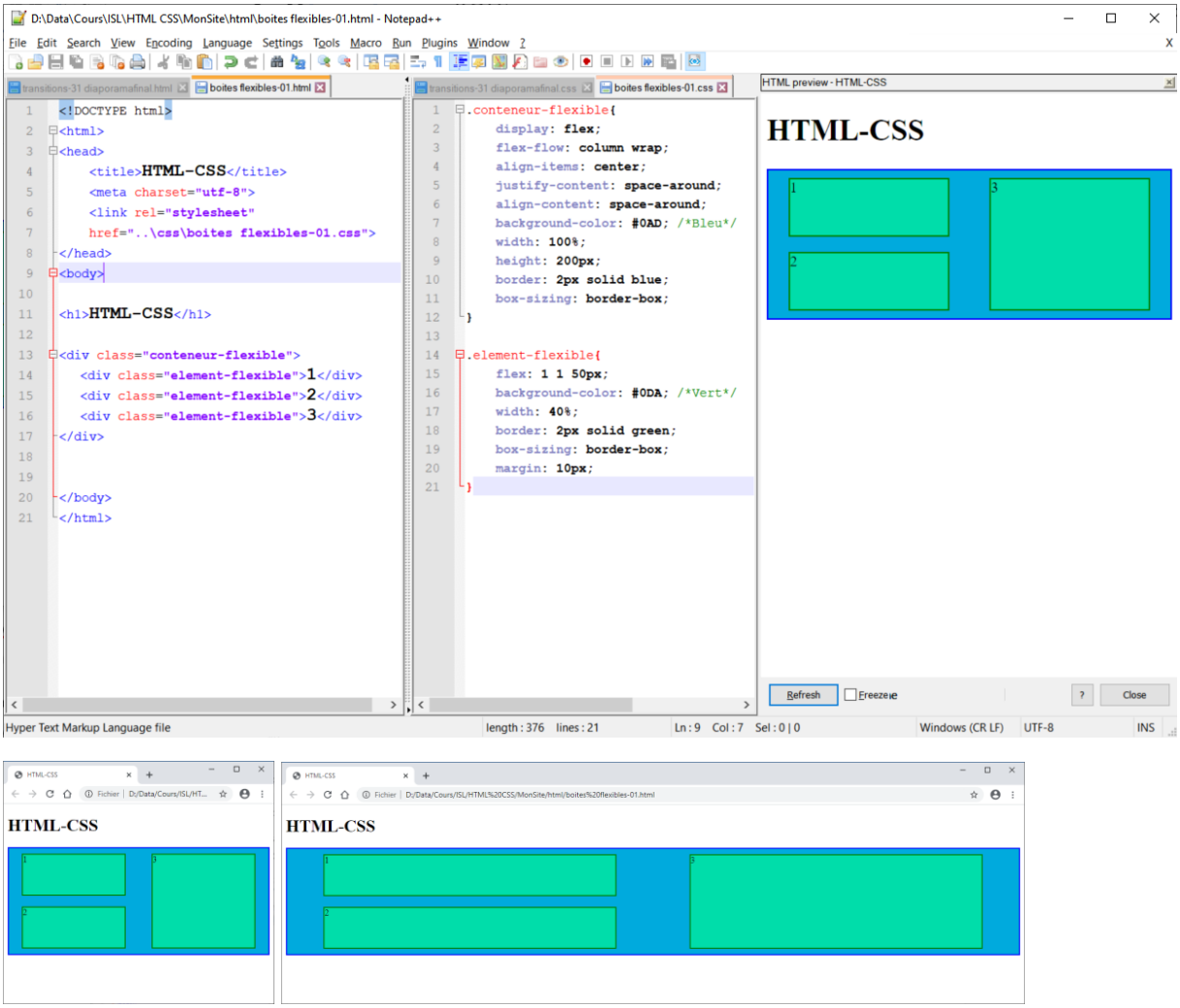
Définition du modèle des boites flexibles ou flexbox

Le flexbox est un modèle de disposition très puissant qui va nous permettre de contrôler facilement et avec précision l'alignement, la direction, l'ordre et la taille de nos éléments (ou plus précisément de nos boites).

Avec le modèle des boites flexibles, nous allons pouvoir définir des conteneurs flexibles. Ces conteneurs sont dits « flexibles » car leurs enfants directs vont être des éléments flexibles qui vont pouvoir se réarranger (se redimensionner, se réaligner, etc.) automatiquement dans leur conteneur lorsque celui-ci change de dimension.

Le flexbox représente ainsi en quelque sorte une alternative beaucoup plus puissante à la propriété float qui nous permettait de positionner nos boites horizontalement puisqu'on va avoir cette fois-ci un contrôle total sur la disposition des éléments.

Voici un premier exemple d'organisation des éléments qu'on peut obtenir avec quelques lignes de code simplement en utilisant le modèle des boites flexibles et ses propriétés :



N'essayez pas pour le moment de comprendre le code ci-dessus : nous allons étudier toutes les spécificités du flexbox dans les leçons à venir !

Conteneur et éléments flexibles

Le modèle des boites flexibles fait la distinction entre deux types de boites auxquelles on va pouvoir appliquer différentes propriétés : des *conteneurs flexibles* d'un côté et des éléments flexibles ou « *flex items* » de l'autre.

Concrètement, nous allons définir un conteneur flexible en attribuant un display : flex à un élément. Tous les éléments directement contenus dans ce conteneur (c'est-à-dire tous les enfants directs) vont alors automatiquement devenir des éléments flexibles.

Nous allons ensuite pouvoir manipuler chaque flex item de manière indépendante grâce à certaines propriétés du modèle des boites flexibles et / ou leur appliquer des propriétés de disposition en tant que groupe en manipulant le conteneur flexible.

Notez également qu'on va pouvoir créer autant de niveau d'imbrication de conteneurs et d'éléments flexibles que souhaité : il va en effet suffire d'ajouter un display : flex à un flex item pour qu'il serve également de conteneur flexible pour ses propres enfants.

Finalement, notez qu'on va pouvoir choisir de définir un conteneur flexible qui va avoir les mêmes propriétés qu'un élément de niveau block avec display : flex ou un conteneur qui va posséder les mêmes propriétés qu'un élément de niveau inline avec display : inline-flex.

Définir un conteneur flexible avec display : flex ou display : inline-flex n'impactera que le comportement du conteneur et pas celui de ses enfants directs qui seront toujours des éléments flexibles.

Notez bien ici que les conteneurs flexibles définissent un nouveau contexte de formatage, ce qui signifie que certaines propriétés qu'on va pouvoir utiliser avec des éléments de type block par exemple ne vont pas se comporter de la même façon avec les éléments flexibles ou ne vont simplement pas pouvoir être utilisées.

En particulier, on notera que les propriétés float, clear et vertical-align ne vont pas pouvoir s'appliquer à des éléments flexibles et que les marges haute et basse d'un conteneur flexible ne vont pas fusionner avec celles de ses enfants.

Axe principal et axe secondaire des boites flexibles

En plus de la distinction conteneur flexible / éléments flexibles, vous devez absolument comprendre la notion d'axe principal et d'axe secondaire des boites flexibles.

En effet, le comportement (et le résultat) de la plupart des propriétés liées au modèle des boites flexibles va être intimement lié à cette notion d'axe.

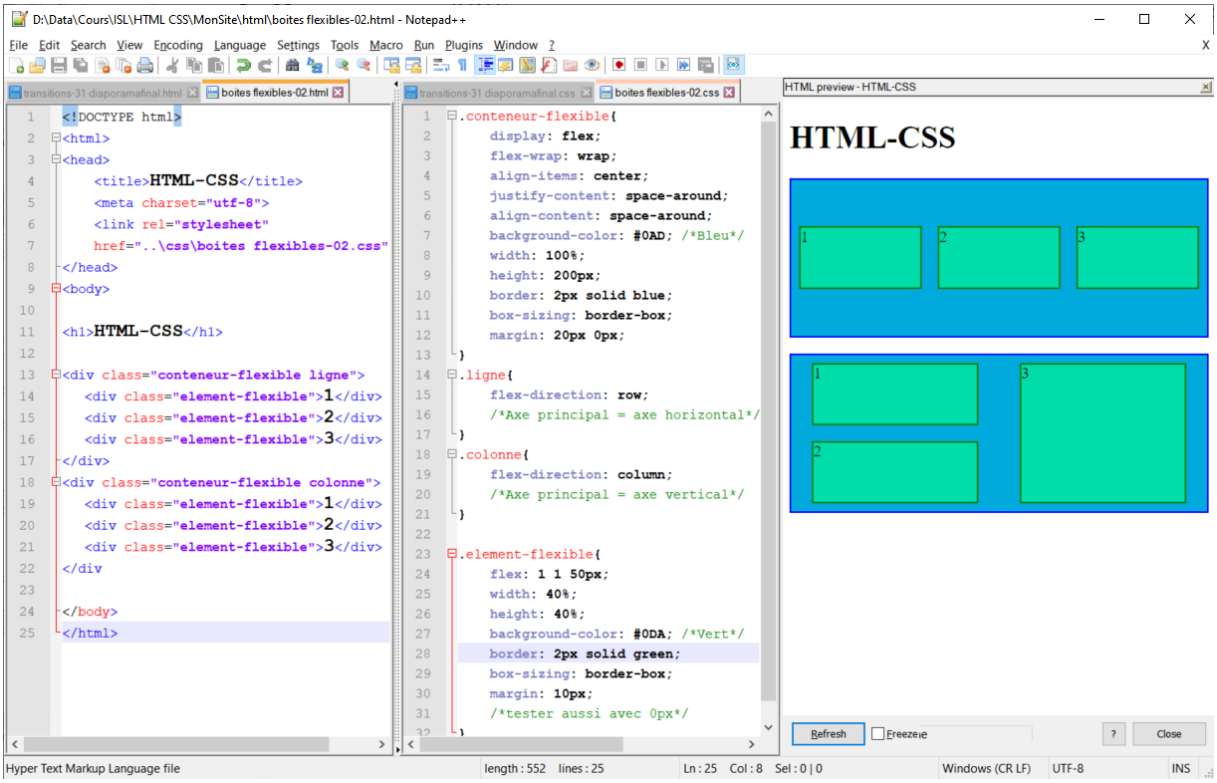
Pour le dire très simplement, deux axes vont intervenir dans le modèle des boites flexibles : l'axe horizontal et l'axe vertical.

La propriété flex-direction va nous permettre de définir quel va être l'axe principal pour un conteneur flexible et ses éléments flexibles ainsi que la direction des éléments le long de cet axe. Le deuxième axe sera ainsi appelé axe secondaire.

Par défaut, l'axe principal est l'axe horizontal (et la direction va être celle de l'écriture) et l'axe secondaire est l'axe vertical (et la direction va être de haut en bas).

Une nouvelle fois, la plupart des propriétés du flexbox vont permettre d'organiser les éléments selon un axe (certaines selon l'axe principal, d'autres selon l'axe secondaire) et selon la direction. Il va donc être essentiel de bien savoir quel est notre axe principal et quel est notre axe secondaire !

Regardez plutôt l'exemple ci-dessous pour vous en convaincre. La seule différence entre les deux conteneurs flexibles est que l'axe principal du premier défini avec flex-direction est l'axe horizontal tandis que l'axe principal du second est l'axe vertical.


D:\Data\Cours\ISL\HTML CSS\MonSite\html\boites flexibles-02.html - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
transitions-31 diaporamafinal.html
boites flexibles-02.html
<!DOCTYPE html>
<html>
<head>
<title>HTML-CSS</title>
<meta charset="utf-8">
<link rel="stylesheet"
href="..\css\boites flexibles-02.css"
</head>
<body>
<h1>HTML-CSS</h1>

1
2
3

1
2
3

</body>
</html>
transitions-31 diaporamafinal.css
boites flexibles-02.css
.conteneur-flexible{
display: flex;
flex-wrap: wrap;
align-items: center;
justify-content: space-around;
align-content: space-around;
background-color: #0AD; /*Bleu*/
width: 100%;
height: 200px;
border: 2px solid blue;
box-sizing: border-box;
margin: 20px 0px;
}
.ligne{
flex-direction: row;
/*Axe principal = axe horizontal*/
}
.colonne{
flex-direction: column;
/*Axe principal = axe vertical*/
}
.element-flexible{
flex: 1 1 50px;
width: 40%;
height: 40%;
background-color: #0DA; /*Vert*/
border: 2px solid green;
box-sizing: border-box;
margin: 10px;
/*tester aussi avec 0px*/
}
HTML preview - HTML-CSS
HTML-CSS
1
2
3
1
3
2
Refresh
Freeze
?
Close
Hyper Text Markup Language file
length : 552 lines : 25
Ln : 25 Col : 8 Sel : 0 | 0
Windows (CR LF)
UTF-8
INS

Gérer la direction des éléments flexibles ou flex items

Le flexbox est un modèle qui va nous permettre de contrôler avec précision la disposition de nos éléments.

Pour disposer nos éléments flexibles, il va déjà falloir définir un axe de direction des éléments principal et un axe secondaire.

Pour définir l'axe principal, nous allons pouvoir utiliser la propriété flex-direction ou la version raccourcie flex-flow en les appliquant au conteneur flexible.

L'objectif de cette leçon est de bien comprendre cette notion d'axe et d'apprendre à modifier la direction des éléments flexibles.

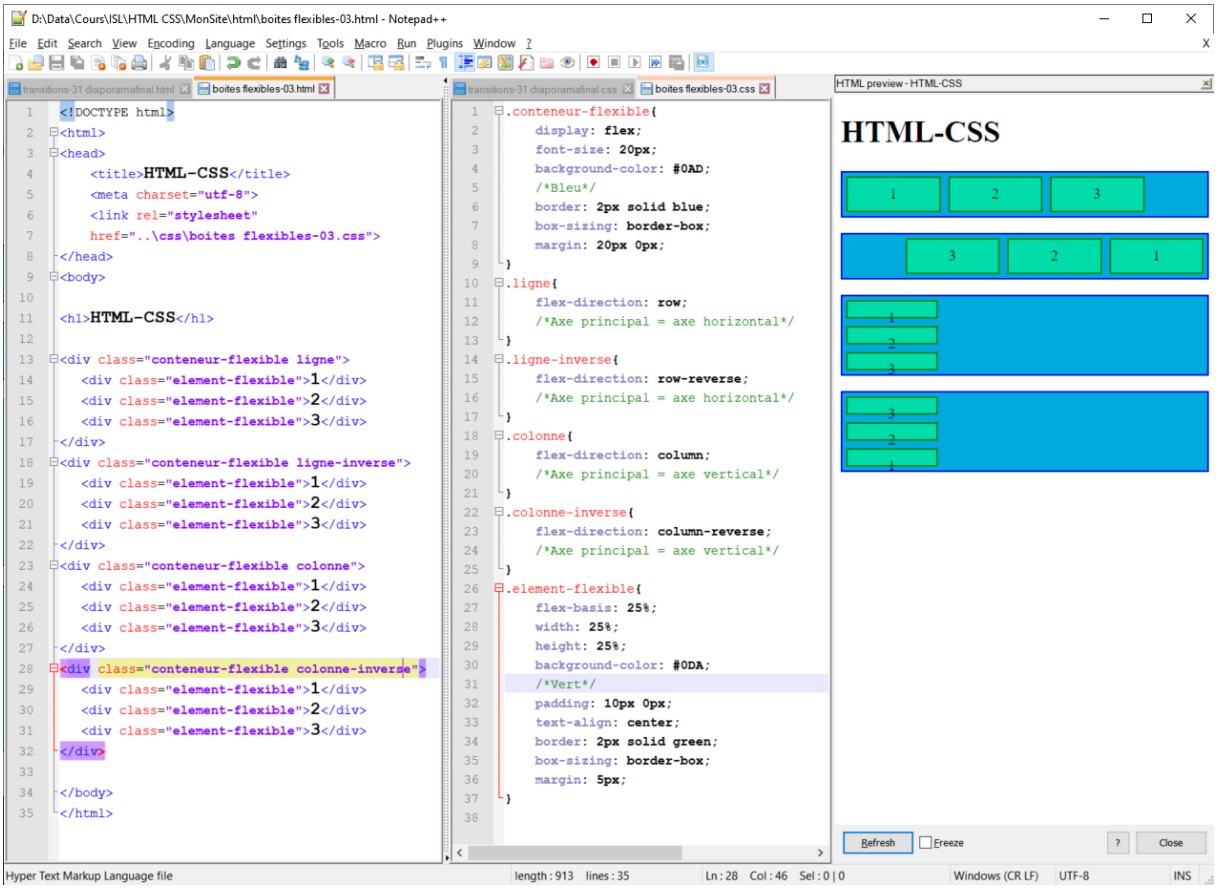
L'axe principal et la propriété flex-direction

La propriété flex-direction nous permet de définir l'axe principal des éléments flexibles. Nous allons utiliser cette propriété sur notre conteneur flexible.

Nous allons pouvoir choisir parmi les valeurs de direction suivantes :

- row : Valeur par défaut. L'axe principal est l'axe horizontal et les éléments flexibles vont se placer en ligne, dans le sens de la lecture (de gauche à droite pour un français) ;
- row-reverse : L'axe principal est l'axe horizontal et les éléments vont se placer en ligne. Cette fois-ci, les éléments se placent dans le sens inverse de la lecture ;
- column : L'axe principal est l'axe vertical et les éléments vont se placer en colonne, en partant du début du conteneur vers la fin (du haut vers le bas par défaut) ;
- column-reverse : L'axe principal est l'axe vertical et les éléments vont se placer en colonne, en partant de la fin du conteneur vers le début (du bas vers le haut par défaut).

Prenons immédiatement un premier exemple pour illustrer cela :



Nous définissons ici 4 conteneurs flexibles. Les conteneurs flexibles sont à priori identiques et contiennent chacun 3 éléments désormais flexibles par définition.

Nous passons ensuite une valeur de flex-direction différente à chaque conteneur flexible pour définir la direction des éléments flexibles et également l'axe principal.

Pour nos deux premiers conteneurs, l'axe principal sera l'axe horizontal. Les éléments flexibles vont ainsi se positionner en ligne, les uns à côté des autres.

Avec la valeur row, les éléments suivent le sens de la lecture : les éléments se positionnent de gauche à droite avec le premier élément flexible tout à gauche, puis le deuxième à sa droite et etc.

Avec la valeur row-reverse, les éléments se positionnent selon le sens inverse à la lecture : le premier élément sera positionné à droite, le deuxième à sa gauche et etc.

Pour nos deux derniers conteneurs flexibles, l'axe principal sera l'axe vertical. Les éléments flexibles vont ainsi se positionner en colonne, les uns au-dessous des autres.

La valeur column va faire se positionner les éléments à partir du début (haut) du conteneur et vers la fin (bas) de celui-ci. Ainsi, le premier élément flexible sera en haut du conteneur, le deuxième sera en dessous du premier et etc.

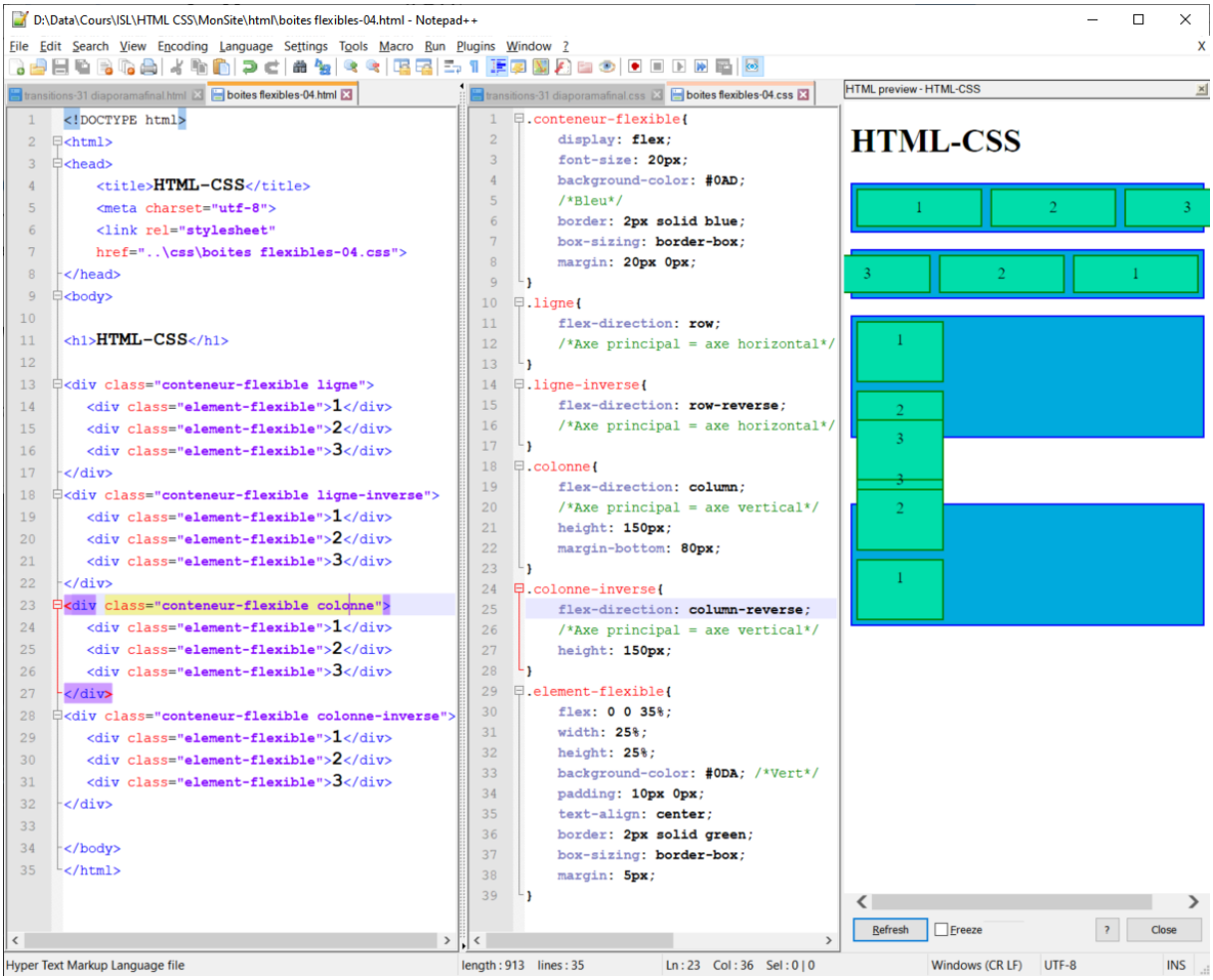
Avec la valeur column-reverse, les éléments flexibles vont continuer de se positionner en colonne mais à partir de la fin du conteneur et vers le début.

Gérer les dépassements des éléments flexibles du conteneur

Dans l'exemple précédent, j'ai choisi les dimensions de mes différents éléments de façon à ce que les éléments flexibles ne dépassent pas de leur conteneur.

Cependant, vous devez savoir que si les éléments flexibles occupent plus de place selon leur axe principal que leur conteneur, alors ils vont par défaut dépasser de celui-ci.

Cela est dû au fait que les éléments flexibles n'ont pas le droit par défaut d'aller à la ligne (si l'axe principal est l'axe horizontal) ou de se placer sur une autre colonne (si l'axe principal est l'axe vertical).



Dans l'exemple précédent, j'ai défini une dimension par défaut pour chacun de mes éléments flexibles égale à 35% de la taille du conteneur avec la propriété flex que nous étudierons plus tard.

Pour comprendre l'exemple ci-dessus, vous devez savoir que le flex : 0 0 35% va définir la largeur des éléments flexibles si l'axe principal est l'axe horizontal ou la hauteur de ces éléments si l'axe principal est l'axe vertical.

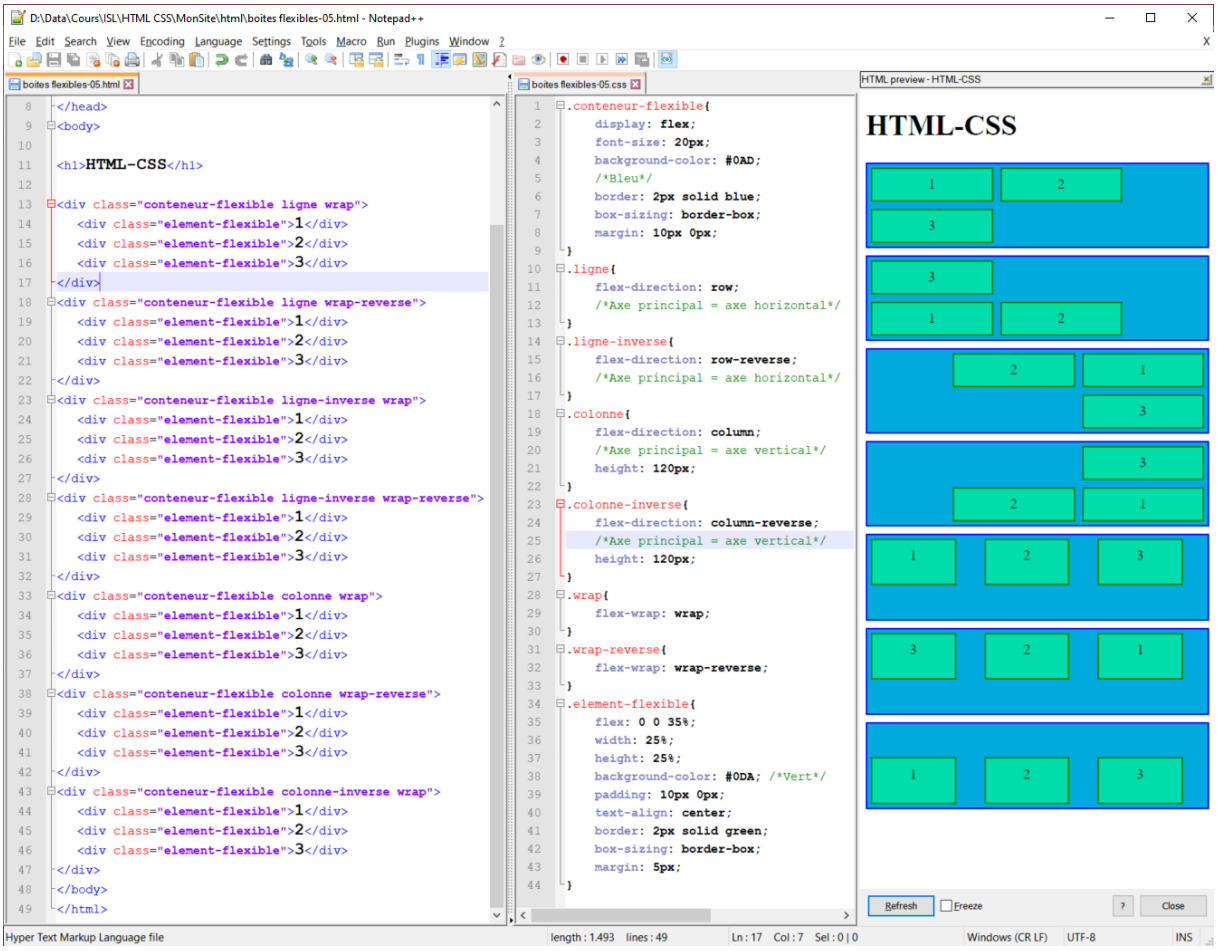
J'ai également ici défini une hauteur fixe pour nos deux conteneurs dont l'axe principal est l'axe vertical car sinon les éléments ne pourraient jamais dépasser. En effet, je vous rappelle que par défaut la hauteur d'un élément va dépendre de celle de ses enfants sauf si une hauteur explicite est précisée.

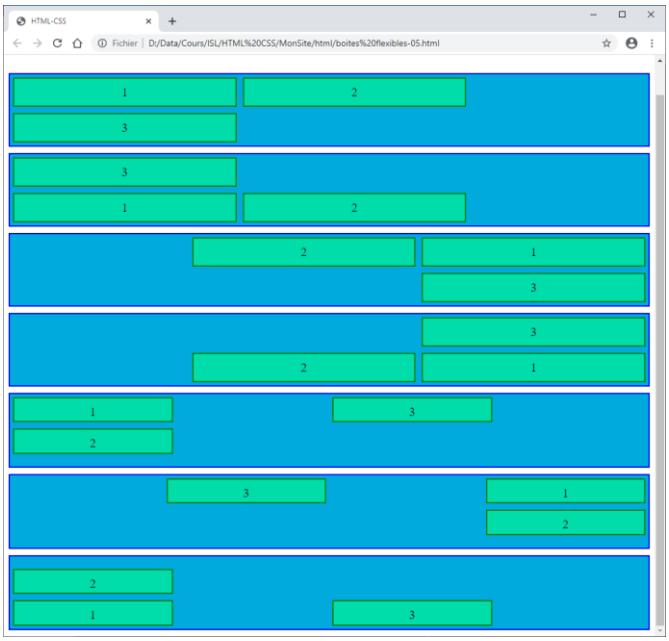
Nous avons donc finalement ici des éléments flexibles qui sont plus grands que leur conteneur (selon leur axe principal) et qui vont donc par défaut dépasser de celui-ci.

Pour éviter ce comportement souvent non souhaité, nous allons pouvoir utiliser la propriété flex-wrap qui va pouvoir donner la possibilité aux éléments flexibles d'aller à la ligne ou en colonne s'ils dépassent de leur conteneur.

Cette propriété va à nouveau s'appliquer au conteneur et accepte les valeurs suivantes :

- nowrap : Valeur par défaut. Les éléments vont rester sur la même ligne ou sur la même colonne (selon leur axe principal) ;
- wrap : Les éléments qui dépassent du conteneur vont avoir la possibilité d'aller à la ligne ou de se positionner sur une nouvelle colonne à partir du début du conteneur et en allant vers la fin de celui-ci (de haut en bas ou de gauche à droite par défaut);
- wrap-reverse : Les éléments qui dépassent du conteneur vont avoir la possibilité d'aller à la ligne ou de se positionner sur une nouvelle colonne à partir cette fois-ci de la fin du conteneur et en allant vers le début (de bas en haut ou de droite à gauche par défaut).





La direction des éléments de mon premier conteneur est définie avec flex-direction : row et on laisse la possibilité aux éléments d'aller à la ligne en partant du début du conteneur vers la fin avec flex-wrap : wrap. *L'élément qui dépasse va donc être placé sous les autres, sur la gauche.*

La direction des éléments du deuxième conteneur est également définie avec flex-direction : row mais cette fois-ci on laisse la possibilité aux éléments d'aller à la ligne en partant de la fin du conteneur vers le début avec flex-wrap : wrap-reverse.

Selon l'axe horizontal, le « début » du conteneur est sa première ligne tandis que la « fin » du conteneur est sa dernière ligne. Les éléments vont donc ici se positionner à partir du bas du conteneur et les éléments qui dépassent vont se positionner au-dessus.

La direction des éléments des troisième et quatrième conteneurs est définie avec flex-direction : row-reverse : les éléments vont se positionner en ligne à partir de la droite et vers la gauche.

On applique un flex-wrap : wrap à notre troisième conteneur, ce qui fait que les éléments vont pouvoir se positionner sur plusieurs lignes en partant du haut du conteneur et en allant vers le bas. Les éléments qui dépassent vont donc être renvoyés sur la ligne du dessous.

On applique un flex-wrap : wrap-reverse à notre troisième conteneur, ce qui fait que les éléments vont pouvoir se positionner sur plusieurs lignes en partant du bas du conteneur. Les éléments qui dépassent vont être renvoyés vers le haut du conteneur, sur une ligne supérieure.

Le même principe va s'appliquer pour les éléments des conteneurs flexibles dont la direction est définie avec column ou column-reverse.

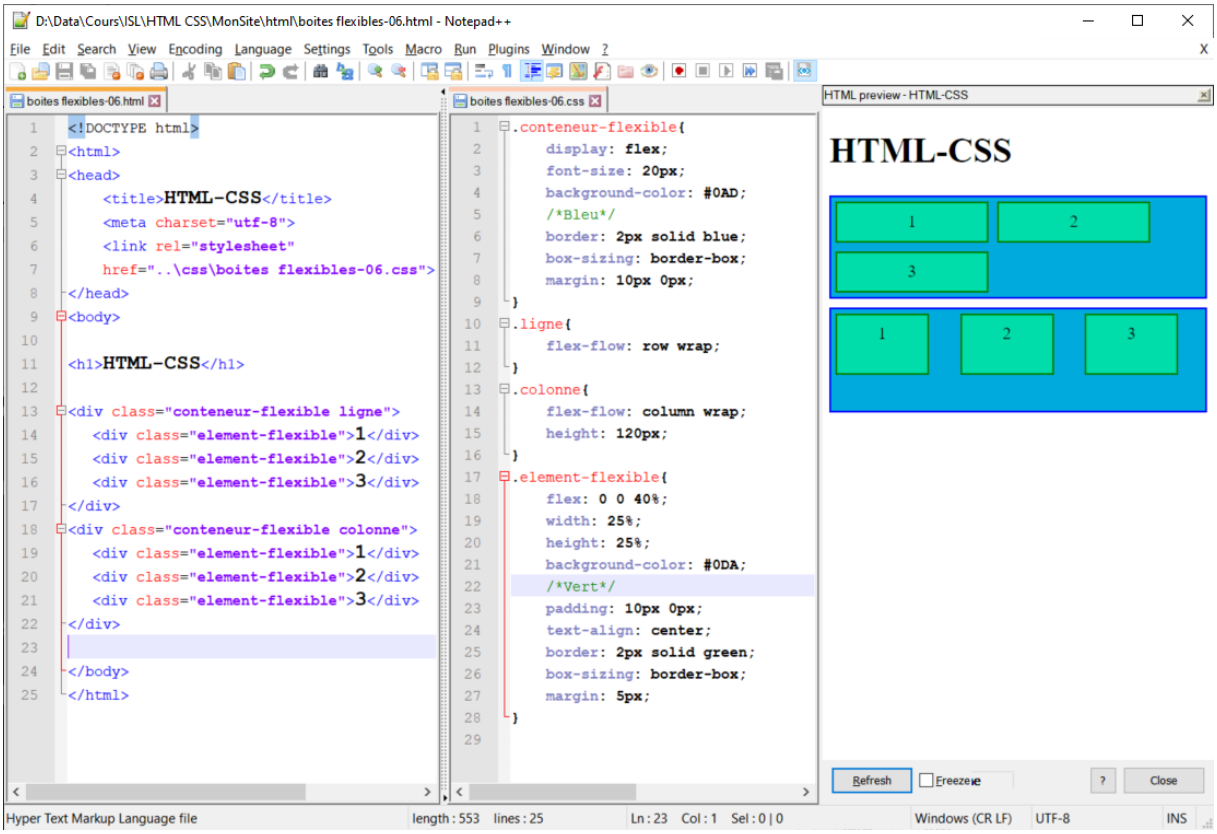
Ici, la valeur wrap de la propriété flex-wrap va permettre aux éléments qui dépassent de se positionner sur une nouvelle colonne, en partant de la gauche et en allant vers la droite. Les éléments qui dépassent vont donc être renvoyés sur d'autres colonnes à droite des premiers éléments.

La valeur wrap-reverse va elle nous permettre de renvoyer les éléments qui dépassent sur une nouvelle colonne à gauche des premiers éléments.

La notation raccourcie flex-flow

Les deux propriétés flex-direction et flex-wrap peuvent être abrégées grâce à la propriété raccourcie flex-flow.

Si une valeur est omise, alors la valeur par défaut de la propriété correspondante sera utilisée à savoir row pour flex-direction et nowrap pour flex-wrap.



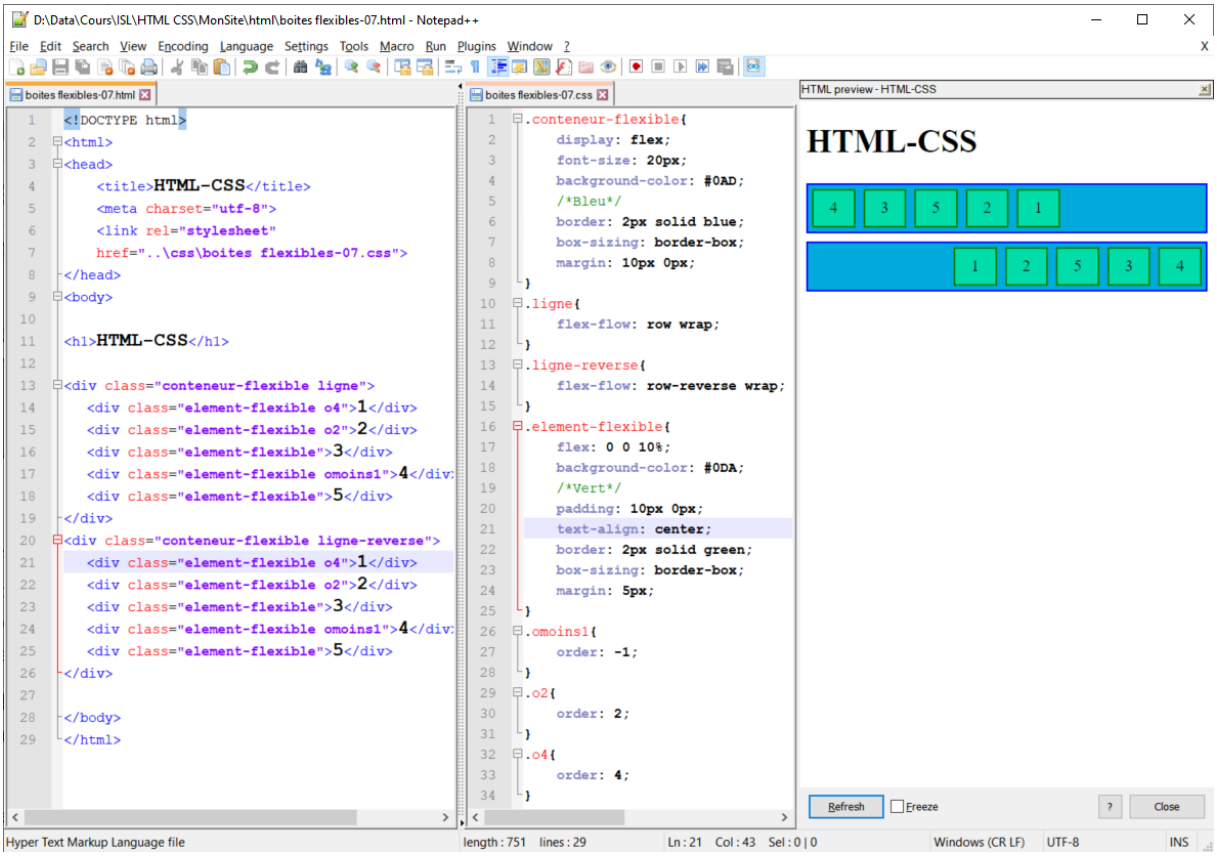
L'ordre d'affichage des éléments flexibles dans le conteneur

En plus de définir une direction pour les éléments de notre conteneur flexible, nous allons également pouvoir choisir leur ordre d'affichage élément par élément grâce à la propriété order.

Nous allons appliquer order aux éléments flexibles et non au conteneur et allons pouvoir lui passer un nombre.

La valeur de order d'un élément va être comparée à celle des autres et l'élément possédant la plus petite valeur sera affiché en premier, celui possédant la deuxième plus petite valeur en deuxième et etc.

La valeur par défaut de order est 0 et si plusieurs éléments possèdent la même valeur pour leur propriété order ils seront affichés selon l'ordre d'écriture du code.



Ici, nous avons deux conteneurs flexibles identiques possédant chacun 5 éléments. La seule différence entre nos deux conteneurs va être la direction de leur axe principal : l'axe principal de notre premier conteneur est horizontal selon une direction qui suit le sens de l'écriture (gauche à droite pour moi) tandis que l'axe principal de notre deuxième conteneur est horizontal selon une direction contraire au sens de l'écriture.

A chaque fois, on attribue un order : 4 au premier élément des conteneurs, un order : 2 au deuxième élément et un order : -1 au quatrième élément.

On ne définit pas de valeur pour la propriété order pour nos troisième et cinquième éléments. La valeur par défaut, 0, sera donc utilisée.

L'élément qui possède la valeur la plus petite pour sa propriété order est donc le quatrième élément qui sera affiché en premier puis vont suivre dans l'ordre d'écriture dans le code les troisième et cinquième éléments puis le deuxième et enfin le premier.

Notez bien ici que l'ordre des éléments va dépendre de la direction de l'axe du conteneur flexible : pour notre deuxième conteneur, le début du conteneur se situe à droite et la fin à gauche. Le premier élément va donc s'afficher à droite, le deuxième à sa gauche et ainsi de suite vers la gauche.

Gérer l'alignement des éléments flexibles

Dans la leçon précédente, nous avons vu comment définir l'axe principal d'un conteneur flexible et avons appris à gérer la direction des éléments dans ce conteneur.

Le modèle des boites flexibles va nous permettre d'aller plus loin et de définir l'alignement des éléments à l'intérieur du conteneur selon l'axe principal ou l'axe secondaire.

Dans cette nouvelle leçon, nous allons apprendre à aligner les éléments d'un conteneur flexible par rapport à leur axe principal ou secondaire en nous servant des propriétés suivantes :

- justify-content ;
- align-items ;
- align-content ;
- align-self.

Attention : cette partie n'est pas simple. Il faudra bien comprendre les concepts d'axe principal et d'axe secondaire et bien se représenter mentalement l'espace que prend une ligne ou une colonne et l'espace que prennent les éléments dans chaque ligne ou colonne. J'ai essayé d'illustrer au maximum pour que vous puissiez bien vous représenter tout cela. N'hésitez pas à faire des pauses, à lire la leçon plusieurs fois et à expérimenter !

Gérer l'alignement des éléments selon l'axe principal avec justify-content

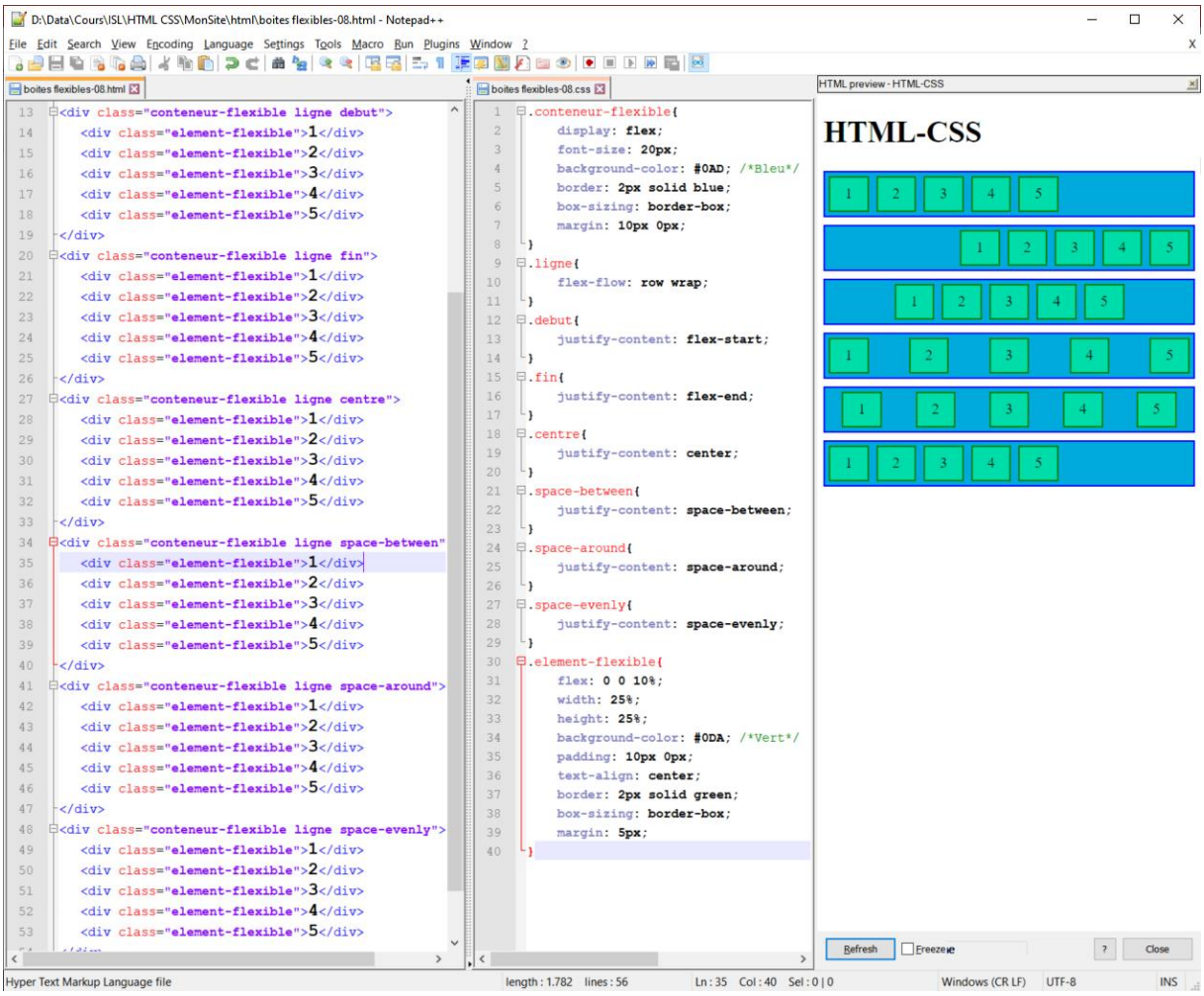
La propriété justify-content nous permet d'aligner les éléments d'un conteneur flexible selon l'axe principal.

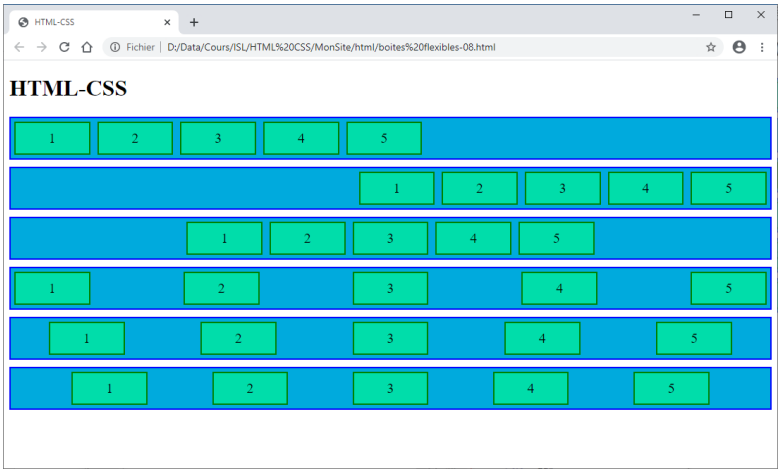
Plus précisément, cette propriété va nous permettre de *définir de quelle manière doit être distribué l'espace restant dans le conteneur*. On va l'appliquer au conteneur et pouvoir lui passer l'une des valeurs suivantes :

- flex-start : Valeur par défaut. Les éléments vont être concentrés au début du conteneur (selon leur axe principal) ;
- flex-end : Les éléments vont être concentrés à la fin du conteneur (selon leur axe principal) ;
- center : Les éléments vont être centrés dans le conteneur (selon leur axe principal) ;
- space-between : Les éléments vont être régulièrement distribués dans le conteneur. Les éléments se trouvant contre un bord du conteneur vont être collés au bord ;
- space-around : Les éléments vont être régulièrement distribués dans le conteneur. Chaque élément va posséder le même espace et les espaces vont être cumulatifs entre deux éléments, ce qui fait que 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 ;

- space-evenly Les éléments vont être régulièrement distribués dans le conteneur. L'espace entre le bord du conteneur et un élément sera le même que celui entre deux éléments.

Prenons immédiatement un exemple pour illustrer le fonctionnement de cette propriété :



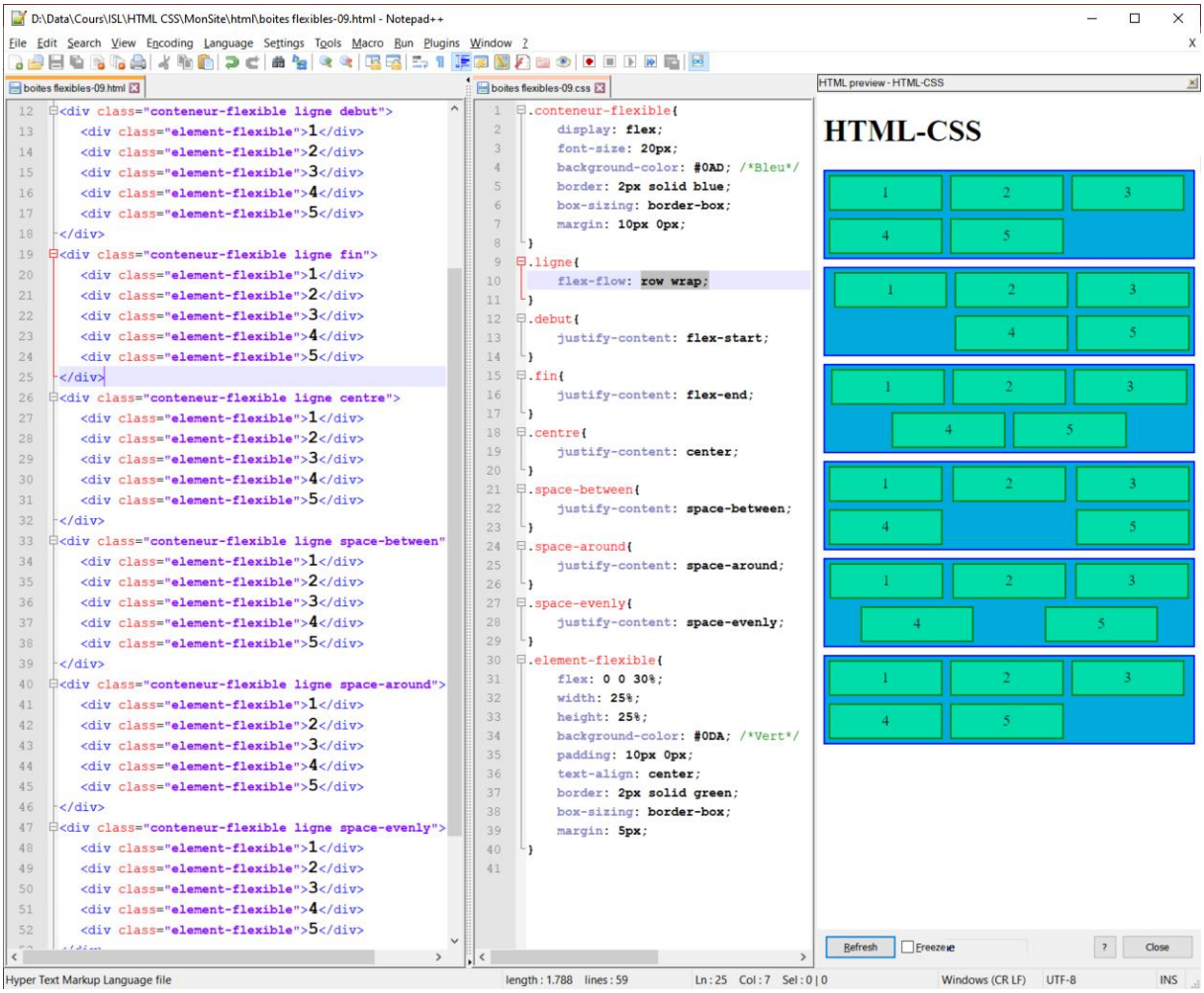


Dans cette première série d'exemples, notre axe principal est l'axe horizontal et nos éléments flexibles n'occupent qu'une seule ligne dans leur conteneur. La direction d'affichage des éléments est celle du sens de la lecture (de gauche à droite).

Ici, la valeur flex-start va aligner les éléments au début du conteneur c'est-à-dire dans le cas présent à gauche de celui-ci tandis que flex-end va les aligner sur la droite et que flex-center va nous permettre de centrer horizontalement les éléments dans le conteneur.

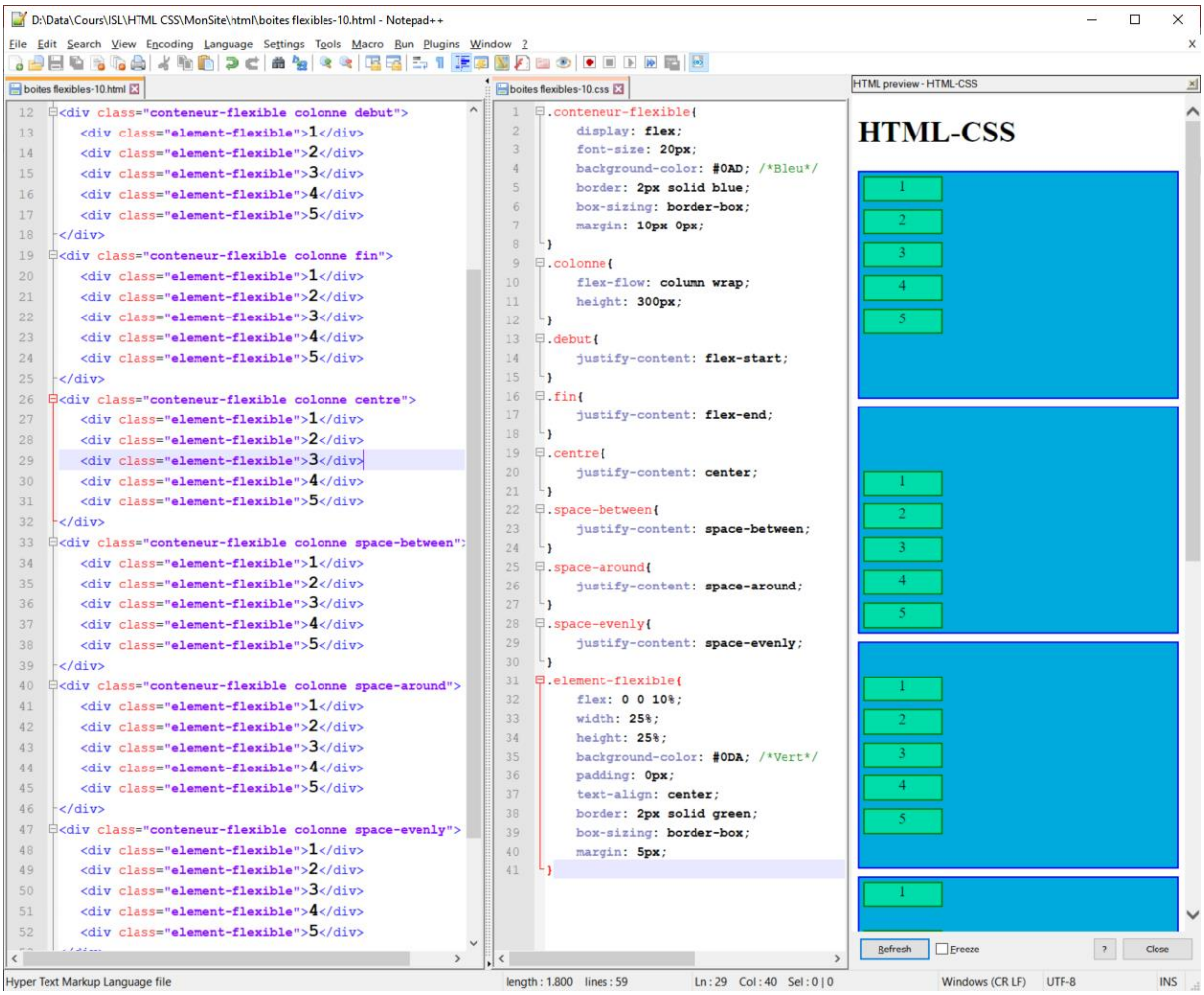
Comme vous pouvez le voir, l'espace vide est régulièrement distribué dans le conteneur avec les valeurs space-between, space-around et space-evenly. La seule différence entre ces valeurs va être l'espace qui va être attribué entre les bords du conteneur et les éléments collés contre les bords.

La propriété justify-content va se comporter exactement de la même façon dans le cas où les éléments occupent plusieurs lignes dans le conteneur :



Regardons de plus près les valeurs space-between, space-around et space-evenly : vous pouvez remarquer que l'espace entre le premier élément et le dernier élément de chaque ligne et le bord du conteneur va être le même pour chaque ligne d'éléments. Ce comportement est tout à fait normal et voulu.

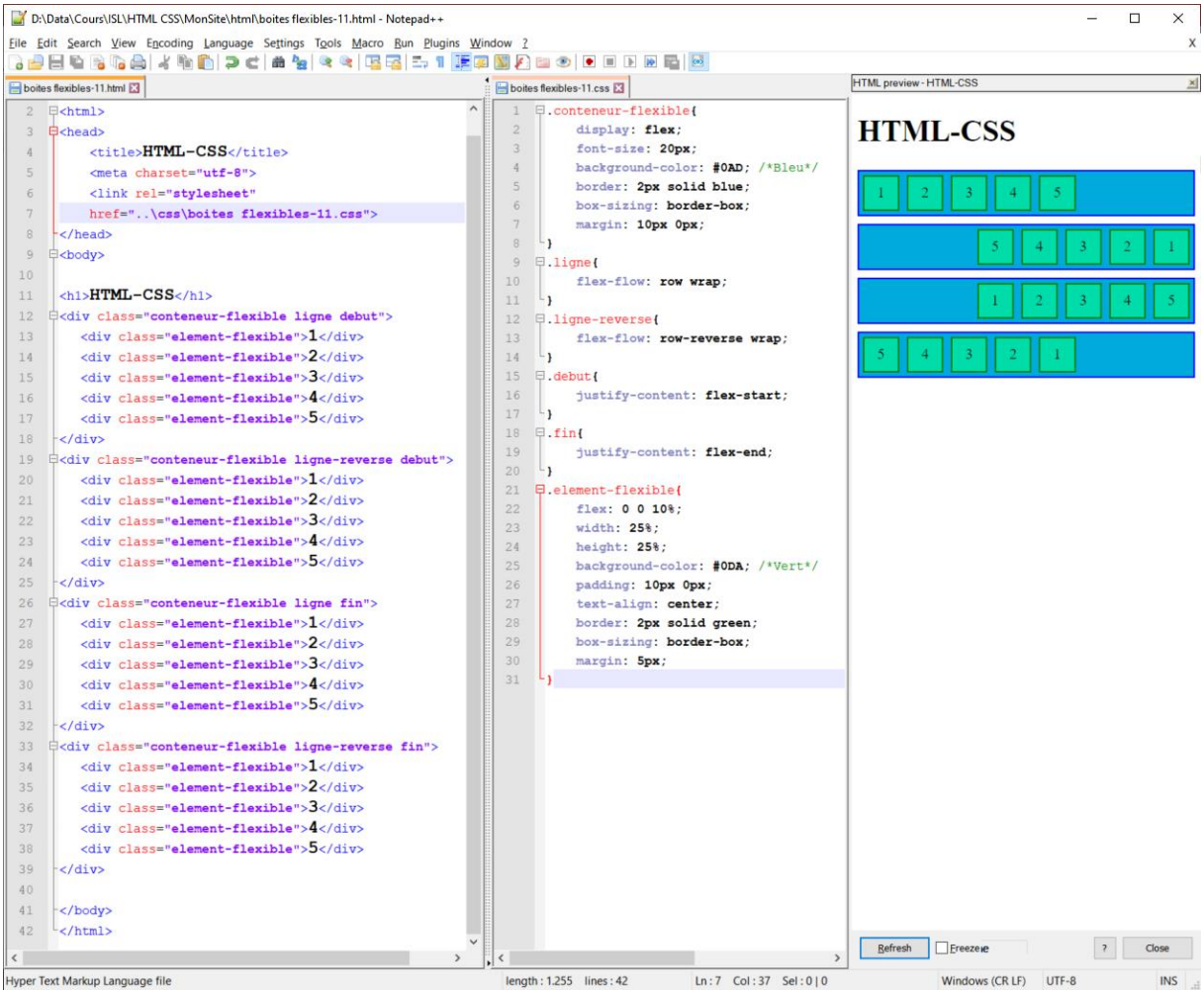
Enfin, la propriété justify-content va à nouveau avoir le même effet si l'axe principal est l'axe vertical.



Cette fois-ci, la valeur flex-start va nous permettre d'aligner les éléments en haut du conteneur tandis que flex-end va nous permettre de les aligner en bas de celui-ci et que center va les centrer verticalement dans le conteneur.

De la même manière c'est l'espace vide dans le sens de la hauteur qui sera distribué régulièrement avec les valeurs space-between, space-around et space-evenly.

Ici, faites bien attention à ne pas confondre les effets des propriétés justify-content et flex-direction : la propriété justify-content va gérer la distribution de l'espace vide dans le conteneur tandis que flex-direction va permettre d'indiquer un axe et ou une direction pour nos éléments flexibles et va donc justement permettre de définir où est le « début » et la « fin » du conteneur.




```html
<html>
<head>
    <title>HTML-CSS</title>
    <meta charset="utf-8">
    <link rel="stylesheet"
    href="..\css\boites flexibles-11.css">
</head>
<body>

<h1>HTML-CSS</h1>
<div class="conteneur-flexible ligne debut">
    <div class="element-flexible">1</div>
    <div class="element-flexible">2</div>
    <div class="element-flexible">3</div>
    <div class="element-flexible">4</div>
    <div class="element-flexible">5</div>
</div>
<div class="conteneur-flexible ligne-reverse debut">
    <div class="element-flexible">1</div>
    <div class="element-flexible">2</div>
    <div class="element-flexible">3</div>
    <div class="element-flexible">4</div>
    <div class="element-flexible">5</div>
</div>
<div class="conteneur-flexible ligne fin">
    <div class="element-flexible">1</div>
    <div class="element-flexible">2</div>
    <div class="element-flexible">3</div>
    <div class="element-flexible">4</div>
    <div class="element-flexible">5</div>
</div>
<div class="conteneur-flexible ligne-reverse fin">
    <div class="element-flexible">1</div>
    <div class="element-flexible">2</div>
    <div class="element-flexible">3</div>
    <div class="element-flexible">4</div>
    <div class="element-flexible">5</div>
</div>

</body>
</html>
```

```css
.conteneur-flexible{
    display: flex;
    font-size: 20px;
    background-color: #0AD; /*Bleu*/
    border: 2px solid blue;
    box-sizing: border-box;
    margin: 10px 0px;
}
.ligne{
    flex-flow: row wrap;
}
.ligne-reverse{
    flex-flow: row-reverse wrap;
}
.debut{
    justify-content: flex-start;
}
.fin{
    justify-content: flex-end;
}
.element-flexible{
    flex: 0 0 10%;
    width: 25%;
    height: 25%;
    background-color: #0DA; /*Vert*/
    padding: 10px 0px;
    text-align: center;
    border: 2px solid green;
    box-sizing: border-box;
    margin: 5px;
}
```

Ce que vous devez absolument comprendre ici est que dans le cas où la valeur de flex-direction est row, l'axe principal sera horizontal et le début du conteneur sera son bord gauche.

En revanche, dans le cas où flex-direction vaut row-reverse, alors le début du conteneur sera son bord droit.

La propriété justify-content va ensuite aligner les éléments du début du conteneur. Ainsi, lorsqu'on applique un justify-content : flex-start sur un conteneur dont la direction est flex-direction : row-reverse, l'effet va être de concentrer les éléments sur la droite du conteneur puisque le début du conteneur correspond à son bord droit.

Note : Tous ces concepts de direction sont dépendants de la direction de l'écriture de votre document. Si votre document est défini dans une langue dont l'écriture se fait de droite à gauche alors le début d'un conteneur flexible dont la direction a été définie avec flex-direction : row sera son bord droit.

Gérer l'alignement des éléments selon l'axe secondaire

La gestion de l'alignement des éléments flexibles selon l'axe secondaire va pouvoir se faire de différentes façons et avec plusieurs propriétés :

- La propriété align-items va nous permettre de gérer l'alignement des éléments au sein d'une ligne (ou d'une colonne selon les axes principal et secondaires choisis) dans l'axe secondaire ;
- La propriété align-content va nous permettre de gérer l'alignement des lignes (ou des colonnes selon l'axe) les unes par rapport aux autres dans l'axe secondaire et ne va donc avoir un effet que si nous avons plusieurs lignes (ou colonnes) d'éléments flexibles ;
- La propriété align-self va nous permettre de gérer l'alignement d'un élément flexible en particulier dans l'axe secondaire.

A partir d'ici, nous allons avoir besoin d'avoir des éléments flexibles de différentes tailles pour bien voir l'effet des différentes propriétés. Pour le moment, je vais me contenter de modifier la hauteur des éléments en attribuant à certains éléments un padding haut et bas en sachant que dans les exemples suivants l'axe principal sera l'axe horizontal sauf mention contraire.

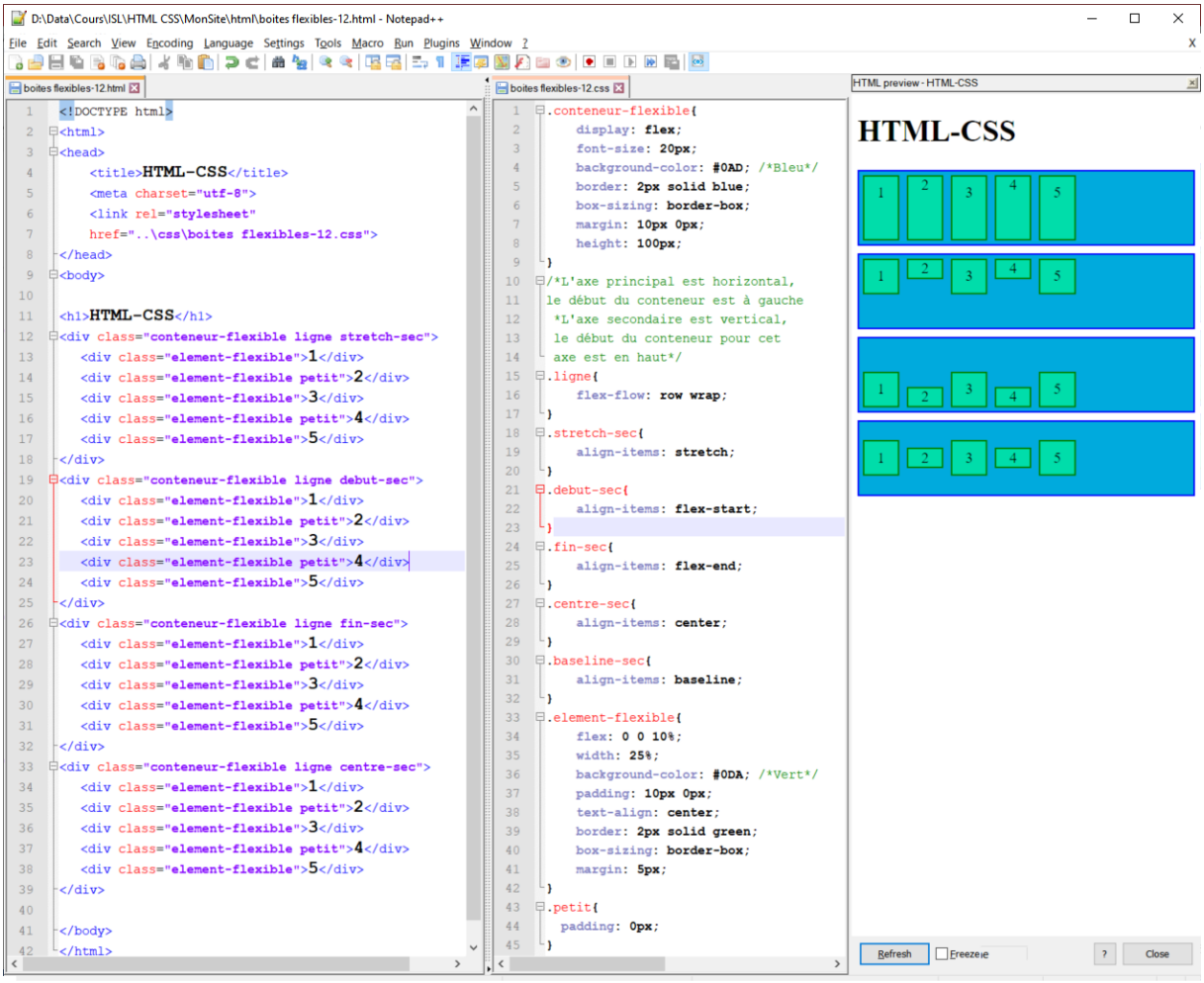
Gérer l'alignement des éléments dans une ligne ou dans une colonne selon l'axe secondaire avec align-items

La propriété align-items va nous permettre de définir l'alignement des éléments flexibles au sein d'une ligne (ou d'une colonne) selon l'axe secondaire.

Cette propriété est l'équivalent de la propriété justify-content mais va permettre l'alignement des éléments selon leur axe secondaire et non pas principal cette fois-ci.

Nous allons une nouvelle fois devoir définir align-items pour le conteneur flexible et allons pouvoir lui passer les valeurs suivantes :

- stretch : Valeur par défaut. Les éléments vont s'étirer dans leur axe secondaire jusqu'à remplir tout l'espace disponible ;
- flex-start : Les éléments vont être placés au début de leur conteneur en fonction de l'axe secondaire ;
- flex-end : Les éléments vont être placés à la fin de leur conteneur en fonction de l'axe secondaire ;
- center : Les éléments vont être placés au milieu de leur conteneur en fonction de l'axe secondaire ;
- 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.



Dans les exemples ci-dessus, l'axe principal de mon conteneur flexible est l'axe horizontal et l'axe secondaire est donc l'axe vertical.

La propriété align-items va donc impacter l'alignement des éléments selon l'axe vertical qui est ici l'axe secondaire.

Par défaut, un conteneur va posséder la même hauteur que le plus grand de ses éléments. Ici, la hauteur de mes éléments dépend de leur contenu et des propriétés padding et border.

Afin que l'effet des propriétés liées à l'alignement que l'on va étudier soit bien visible, je donne une hauteur fixe plus grande que celle de mes éléments à mon conteneur pour créer de l'espace vide.

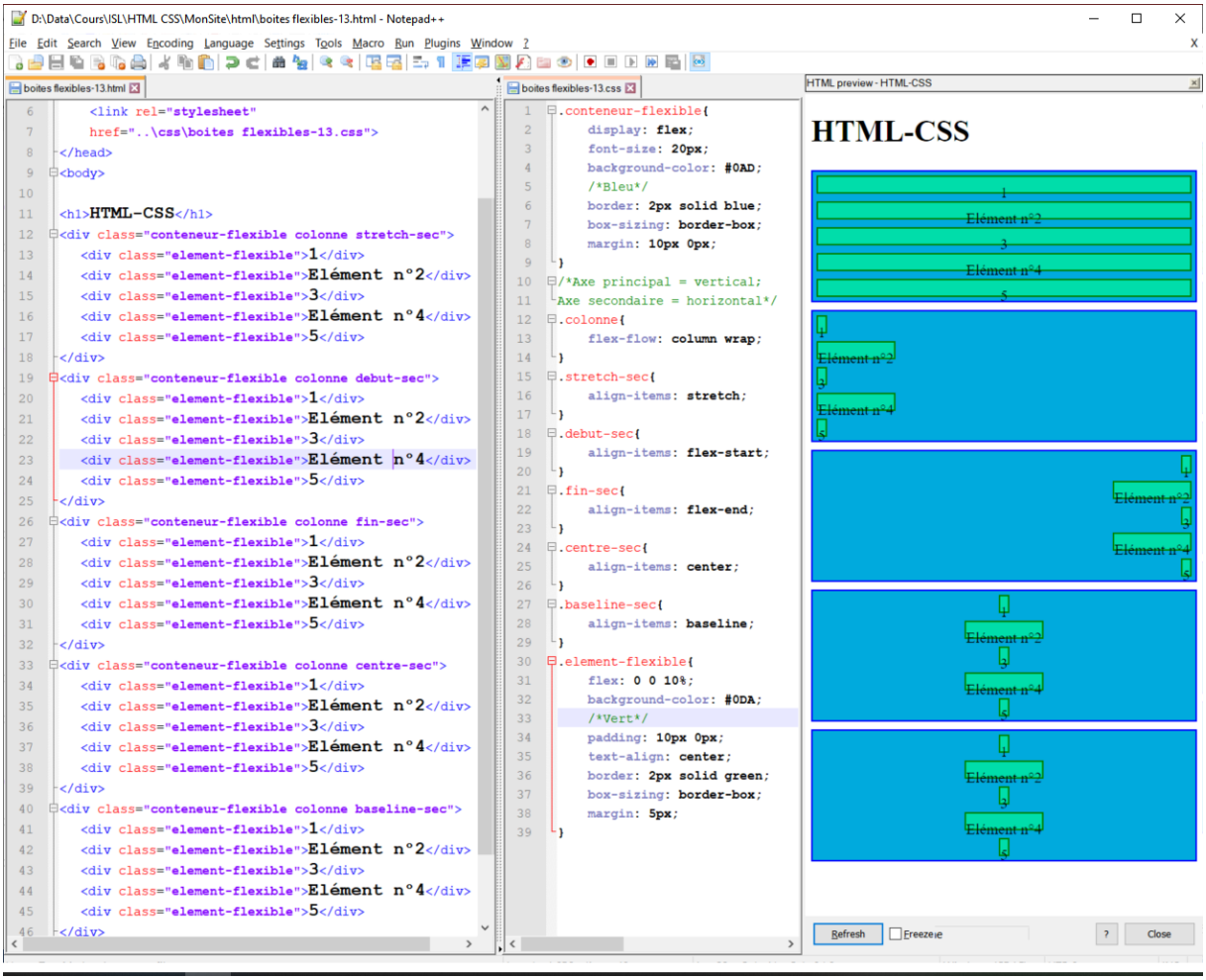
Comme vous pouvez le constater, la valeur stretch va ici étirer les éléments de façon à ce qu'ils occupent tout l'espace disponible dans leur conteneur selon leur axe secondaire qui est ici l'axe vertical.

La valeur flex-start va ici aligner les éléments en haut du conteneur ce qui correspond au début du conteneur dans l'axe vertical tandis que la valeur flex-end va avoir l'effet contraire.

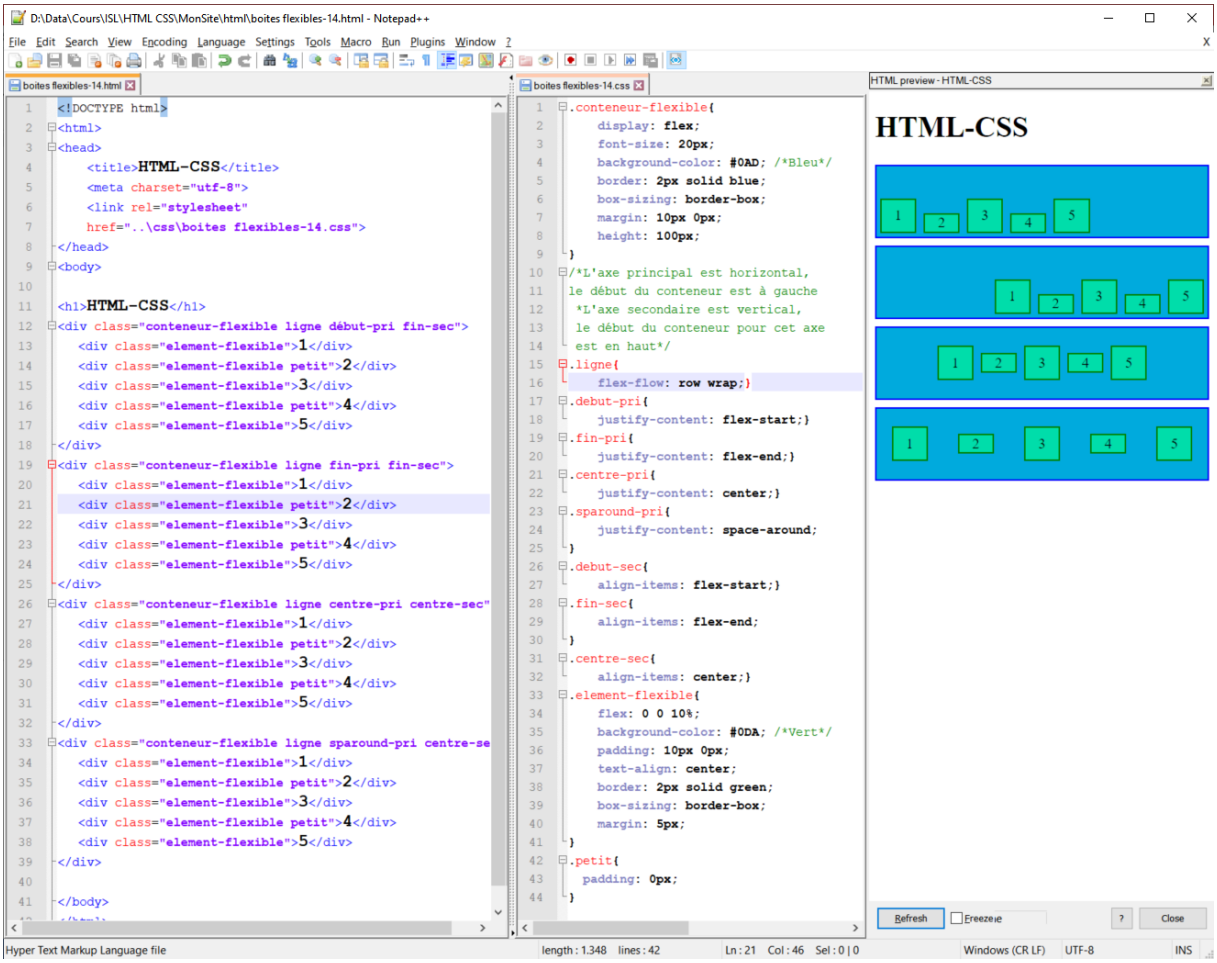
La valeur center nous permet ici très simplement de centrer verticalement les éléments dans leur conteneur.

Finalement, la valeur baseline permet d'aligner les éléments selon leur ligne de base qui est la ligne sur laquelle sont écrits les textes.

De manière similaire, si l'axe vertical est défini comme axe principal pour un conteneur flexible, alors la propriété align-items alignera les éléments selon leur axe horizontal.



Bien évidemment, nous allons pouvoir utiliser les propriétés justify-content et align-items ensemble pour aligner les éléments à la fois selon leur axe principal et selon leur axe secondaire.



Ici, les éléments de notre premier conteneur se placent au début de celui-ci selon l'axe principal qui est l'axe horizontal c'est-à-dire à gauche et à la fin selon l'axe secondaire (axe vertical) c'est-à-dire en bas.

Les éléments du deuxième conteneur s'alignent à la fin de celui-ci à la fois selon l'axe principal et selon l'axe secondaire c'est-à-dire à droite et en bas.

Les éléments de notre troisième conteneur sont centrés à la fois dans l'axe principal et dans l'axe secondaire, ils vont ainsi être centrés à la fois horizontalement et verticalement dans le conteneur. On va ainsi très simplement pouvoir obtenir un centrage parfait en utilisant le flexbox.

Les éléments de notre quatrième conteneur ont un alignement de type space-around (distribution régulière) selon leur axe principal (horizontal) et sont centrés selon leur axe secondaire c'est-à-dire centrés verticalement.

Les propriétés flex-wrap et align-items

Il convient ici de ne pas confondre les effets des propriétés flex-wrap et align-items. La propriété flex-wrap va définir la possibilité pour les éléments qui dépassent du conteneur de se placer sur une nouvelle ligne ou une nouvelle colonne.

Lorsqu'on autorise les éléments à se placer sur une nouvelle ligne ou une nouvelle colonne, c'est-à-dire lorsqu'on passe les valeurs wrap ou wrap-reverse à la propriété flex-wrap, alors cette propriété va également définir la direction de l'axe secondaire.

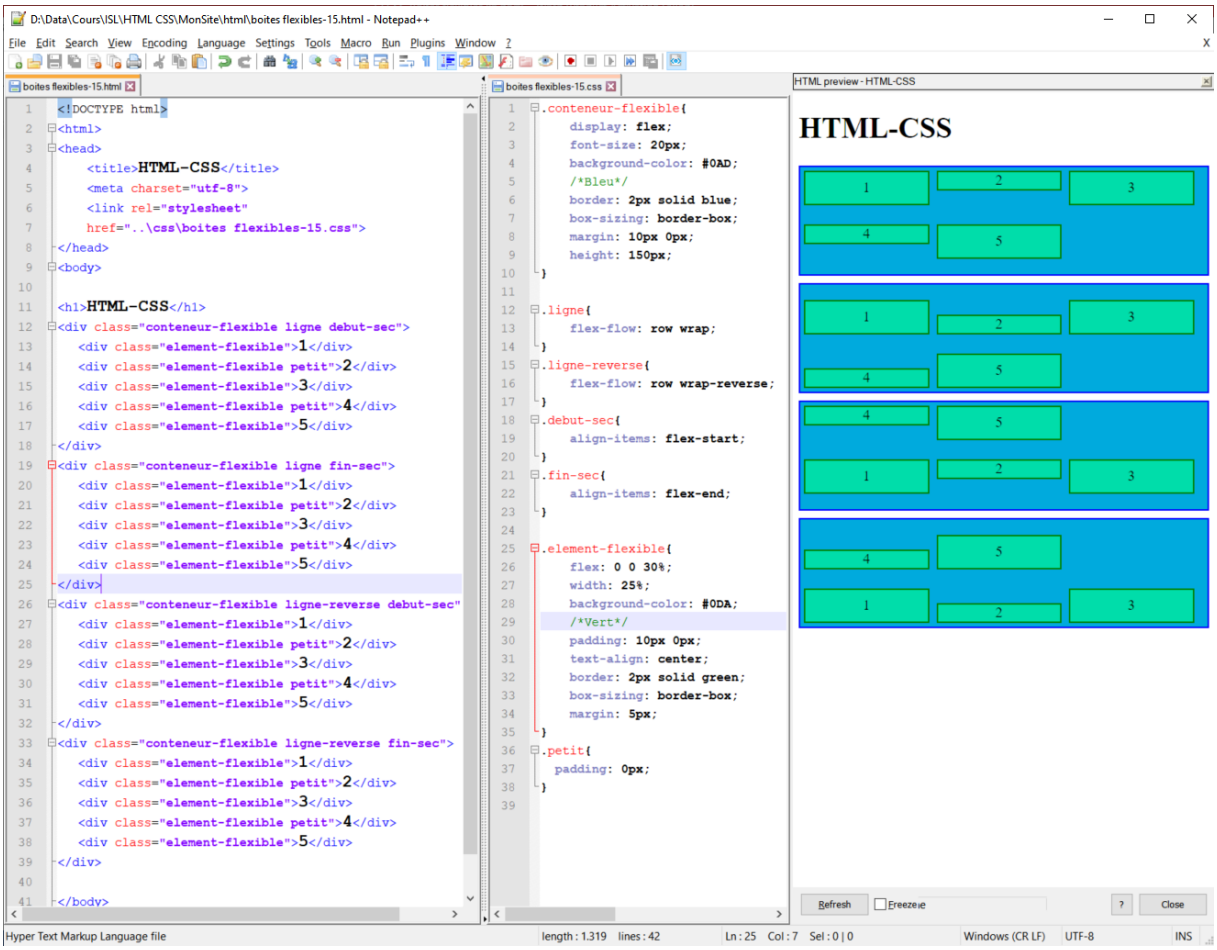
Par défaut, la direction liée à la valeur wrap va être de gauche à droite si l'axe secondaire est l'axe horizontal et de haut en bas si l'axe secondaire est l'axe vertical.

Au contraire, si la valeur de flex-wrap est wrap-reverse, alors la direction de l'axe secondaire sera de droite à gauche si c'est l'axe horizontal et de bas en haut si l'axe secondaire est l'axe vertical.

La propriété align-items va elle définir l'alignement des éléments dans chaque ligne (ou colonne) selon leur axe secondaire. Le résultat lié aux valeurs de align-items va être intimement lié à la direction définie pour l'axe secondaire.

Un flex-flow : row wrap-reverse par exemple va définir que l'axe principal est l'axe horizontal et que sa direction pour nous va être de gauche à droite et que la direction de l'axe vertical sera inversée c'est-à-dire partir du bas (début) vers le haut (fin) pour nous.

Ainsi, si on applique également un align-items : flex-end au conteneur flexible, les éléments se placeront à la fin de leur ligne dans le conteneur selon l'axe secondaire c'est-à-dire… en haut de leur ligne !



Gérer l'alignement des lignes ou des colonnes les unes par rapport aux autres selon l'axe secondaire avec align-content

La propriété align-content ne va pas nous permettre de gérer l'alignement des éléments au sein d'une ligne ou d'une colonne mais plutôt de gérer l'alignement des différentes lignes (ou colonnes) les unes par rapport aux autres.

Cette propriété n'aura d'effet que si le conteneur flexible possède plusieurs lignes (si l'axe principal est horizontal) ou plusieurs colonnes (si l'axe principal est vertical).

On va à nouveau définir cette propriété au niveau de notre conteneur et allons pouvoir lui passer une des valeurs suivantes :

- stretch : Valeur par défaut. Les lignes (ou les colonnes) vont essayer de s'étendre pour prendre tout l'espace disponible dans le conteneur ;
- flex-start : Les lignes (ou les colonnes) vont être concentrées au début du conteneur ;
- flex-end : Les lignes (ou les colonnes) vont être concentrées à la fin du conteneur;
- center : Les lignes (ou les colonnes) vont être concentrées au milieu du conteneur;
- space-between : Les lignes (ou les colonnes) vont être régulièrement distribuées dans le conteneur. La première ligne (ou colonne) va être collée contre le début du conteneur et la dernière sera collée contre la fin du conteneur ;
- space-around : Les lignes (ou les colonnes) vont être régulièrement distribuées dans le conteneur. L'espace autour de chaque ligne (colonne) va être le même et cumulatif entre deux lignes (colonnes).

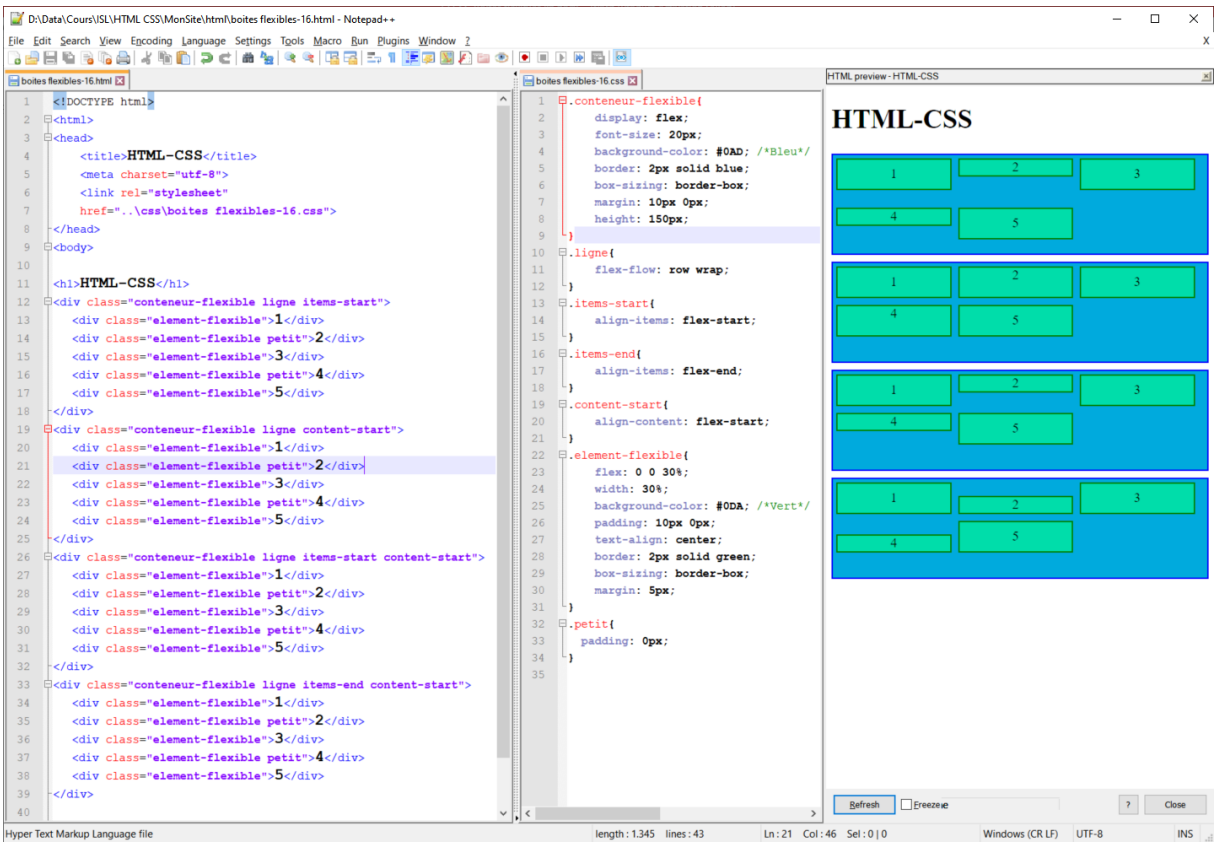
Il convient de ne pas confondre align-items et align-content. En effet, ces deux propriétés agissent sur des choses différentes.

Imaginez un conteneur flexible défini avec flex-flow : row wrap et contenant deux lignes d'éléments. En appliquant un align-items : flex-start, les éléments du conteneur vont être placés au début de celui-ci selon leur axe secondaire c'est-à-dire en haut de celui-ci ou plus exactement les différents éléments vont être placés au début de leur ligne.

En effet, chaque ligne (ou chaque colonne) d'un conteneur flexible agit indépendamment des autres et peut être considérée comme un conteneur à part. Les conteneurs flexibles ont été créés comme cela justement pour nous donner un contrôle sur les éléments dans les lignes (colonnes) et sur les lignes (colonnes) en soi.

La propriété align-content va-elle donc nous permettre ensuite d'aligner les lignes ou colonnes du conteneur.

Regardez l'exemple ci-dessous pour bien comprendre :



L'axe secondaire pour chacun de nos conteneurs flexibles est l'axe vertical. On définit un align-items : flex-start pour notre premier conteneur mais pas de align-content. La valeur par défaut stretch sera donc utilisée. Les lignes vont donc prendre chacune le maximum d'espace dans le conteneur c'est-à-dire 50% chacune et les éléments vont se placer au début de chaque ligne grâce à align-items : flex-start.

Ici, vous devez bien comprendre que le fait qu'une ligne s'étende au maximum dans le conteneur ne signifie pas que les éléments qu'elle contient vont également s'étendre.

Pour notre deuxième conteneur, on demande cette fois-ci aux lignes de se positionner au début du conteneur. Les lignes vont donc se concentrer en haut du conteneur et avoir par défaut une hauteur égale à celle du plus grand des éléments de la ligne.

On ne définit cette fois-ci pas de valeur pour align-items. La valeur par défaut, stretch va donc être utilisée et les éléments vont s'étendre un maximum dans la ligne.

On définit pour notre troisième conteneur un align-items : flex-start et un align-content : flex-start. Les lignes vont donc se concentrer au début du conteneur et les éléments vont se concentrer au début de la ligne.

Finalement, on définit un align-items : flex-end et un align-content : flex-start pour notre quatrième conteneur. Les lignes vont don à nouveau se concentrer en début de conteneur mais les éléments vont cette fois-ci se concentrer à la fin de leur ligne.

Gérer l'alignement d'un élément en particulier dans son axe secondaire avec align-self

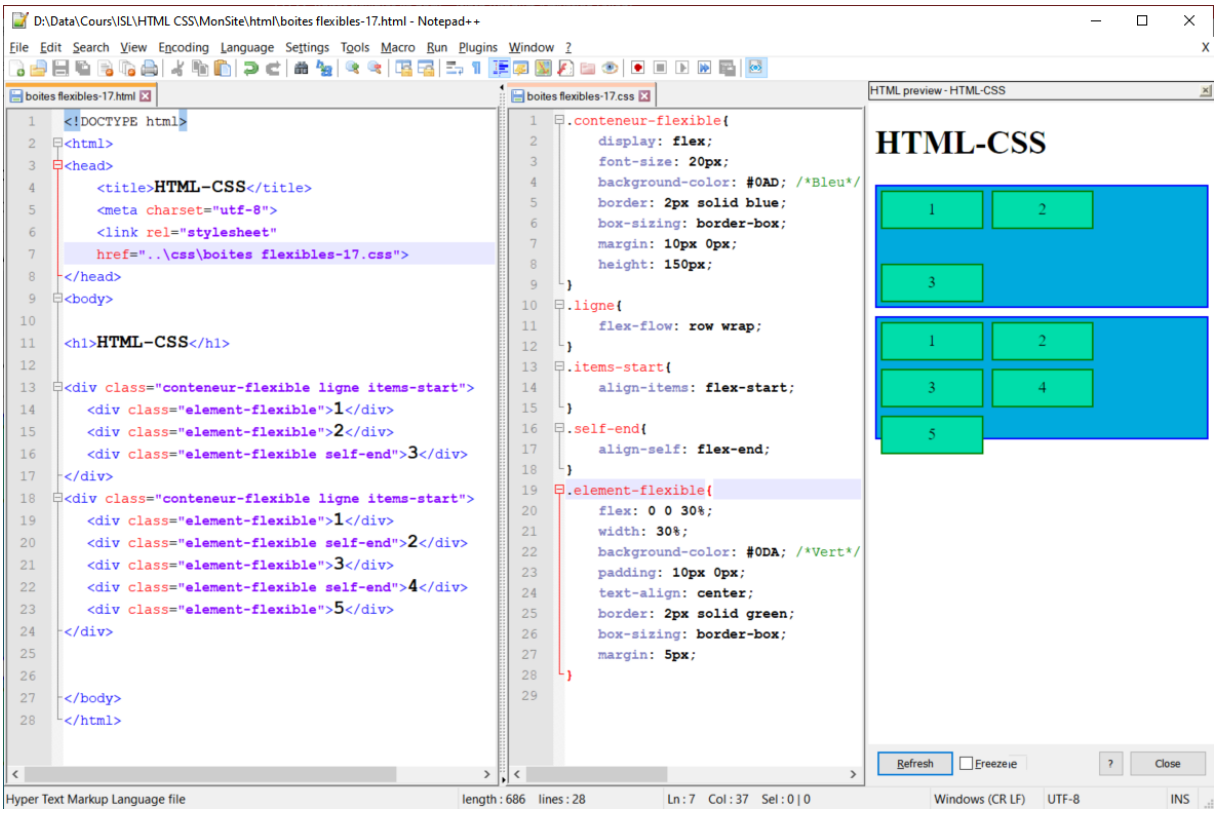
La propriété align-self va finalement nous permettre de gérer l'alignement d'un élément en particulier dans une ligne (ou une colonne) selon l'axe secondaire.

Nous allons cette fois-ci non pas appliquer la propriété au conteneur mais bien aux éléments flexibles et allons pouvoir lui passer les mêmes valeurs qu'à align-items à savoir :

- stretch ;
- flex-start ;
- flex-end ;
- center ;
- baseline ;

Cette propriété va *surcharger* la valeur de align-items pour les éléments pour lesquels elle a été définie. Cela signifie en d'autres termes que ce sera la valeur de align-self qui sera retenue pour l'alignement par-dessus la valeur de align-items.

De manière similaire à align-items, cette propriété n'aura un effet visible que si la ligne (ou la colonne) est plus grande que les éléments qu'elle contient, c'est-à-dire s'il existe des espaces vides.



Dans cet exemple, l'axe secondaire de nos conteneurs est à nouveau l'axe vertical. Notre premier conteneur ne possède qu'une ligne. La hauteur de la ligne va ainsi être égale à celle du conteneur. On

définit ici un align-items : flex-start ce qui place les éléments en haut du conteneur et on surcharge ce comportement pour un élément en particulier qu'on place en bas avec align-self : flex-end.

Notre deuxième conteneur possède deux lignes. Aucune valeur n'est définie pour align-content et c'est donc sa valeur par défaut stretch qui va s'appliquer. On place ensuite les éléments en haut de chaque ligne avec align-items et on surcharge ce comportement pour deux éléments flexibles avec align-self : flex-end ce qui va les envoyer en bas de leur ligne.

Gérer la taille et la flexibilité des éléments flexibles

Dans les leçons précédentes, nous avons vu comment définir la direction des éléments flexibles, leur ordre et avons appris à les aligner sur leurs axes principal et secondaire.

Le modèle des boites flexibles nous offre une dernière série de propriété qui vont nous permettre de gérer la taille des éléments et de définir leur capacité à grossir et à se rétracter en cas de besoin dans le conteneur, c'est-à-dire définir leur capacité à être « flexible ».

Nous allons dans cette nouvelle leçon nous intéresser aux propriétés flex-grow, flex-shrink et flex-basis ainsi qu'à la propriété raccourcie flex.

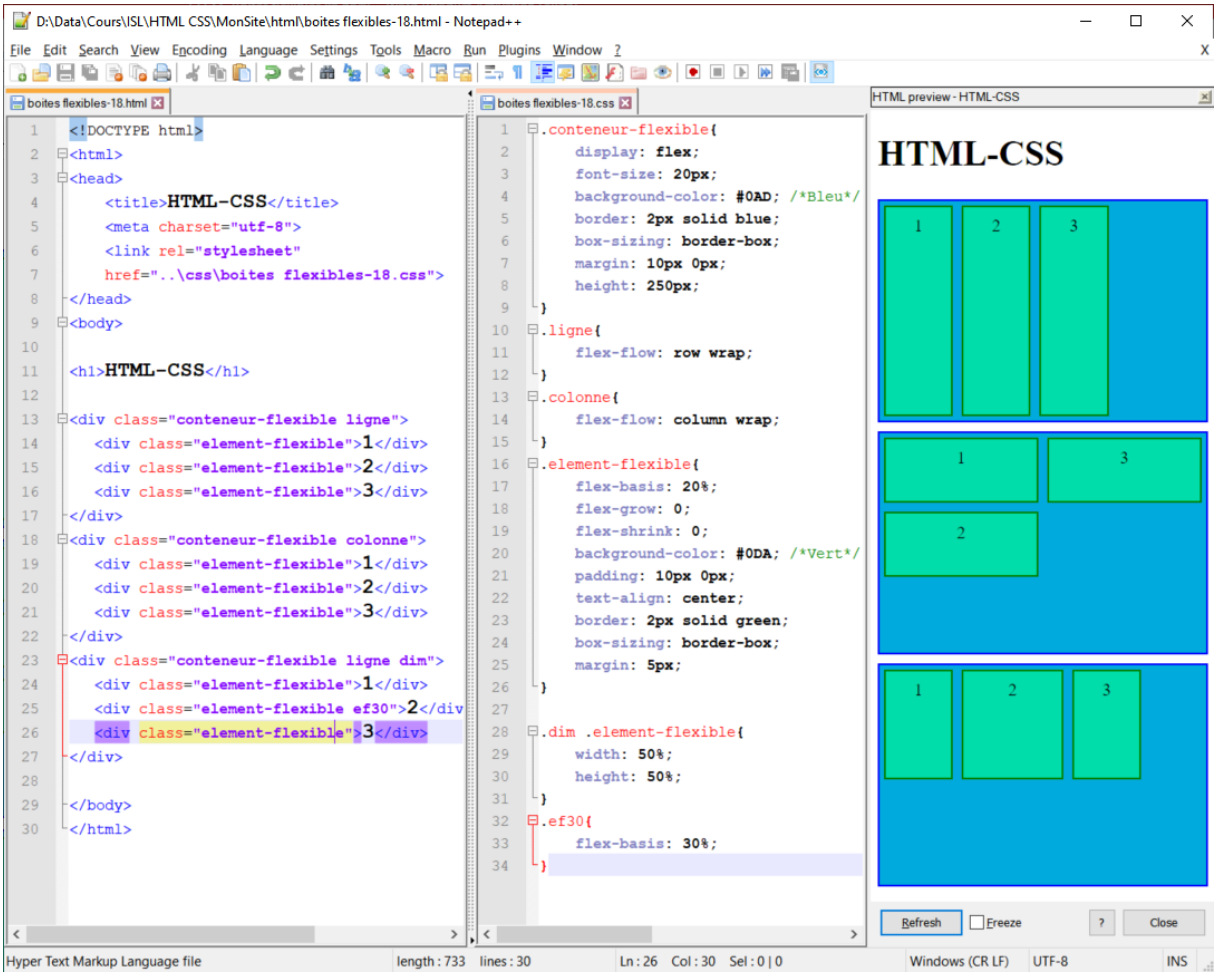
Définir une taille de départ pour nos éléments flexibles avec flex-basis

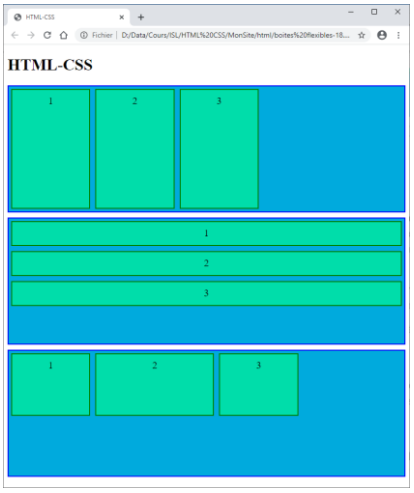
La propriété flex-basis va nous permettre de définir une taille de départ pour les différents éléments d'un conteneur flexible avant distribution éventuelle de l'espace vide du conteneur ou au contraire rétrécissement des éléments.

Nous allons appliquer cette propriété directement aux éléments flexibles et allons pouvoir lui passer une valeur, que nous exprimerons généralement en pourcentage de la taille du conteneur.

Notez ici que la propriété flex-basis ne va définir que la dimension de base d'un élément liée à son axe principal, c'est-à-dire sa largeur si l'axe principal est horizontal et sa hauteur si l'axe principal est vertical.

Cette propriété va par ailleurs surcharger la valeur éventuellement donnée à width ou à height selon que l'axe principal soit horizontal ou vertical.





Ici, nous définissons trois conteneurs flexibles qui contiennent trois éléments flexibles chacun. L'axe principal du premier et du troisième conteneur est horizontal et l'axe principal du deuxième conteneur est vertical. J'ai précisé une hauteur explicite pour nos conteneurs afin de bien voir l'effet des propriétés que nous allons étudier.

Je définis ensuite un flex-basis : 20% pour chacun des éléments flexibles ainsi qu'un flex-grow : 0 et un flex-shrink : 0 que nous allons étudier ensuite pour interdire à nos éléments de grossir ou rétrécir.

Les éléments de notre premier conteneur vont ainsi avoir une largeur de départ égale à 20% de la largeur du conteneur tandis que les éléments de notre deuxième conteneur auront une hauteur également à 20% de la hauteur du conteneur.

Dans notre troisième conteneur, nous définissons un flex-basis différent pour un élément en particulier. L'élément en question aura donc ici une largeur de départ égale à 30% de celle de son conteneur.

Notez que nous passons également une largeur et une hauteur explicites à chacun des éléments flexibles de notre troisième conteneur. L'axe principal de ce conteneur est l'axe horizontal : la propriété flex-basis va donc nous permettre de définir la largeur de départ des éléments flexibles et la propriété width va être surchargée. La propriété height va en revanche ici bien s'appliquer.

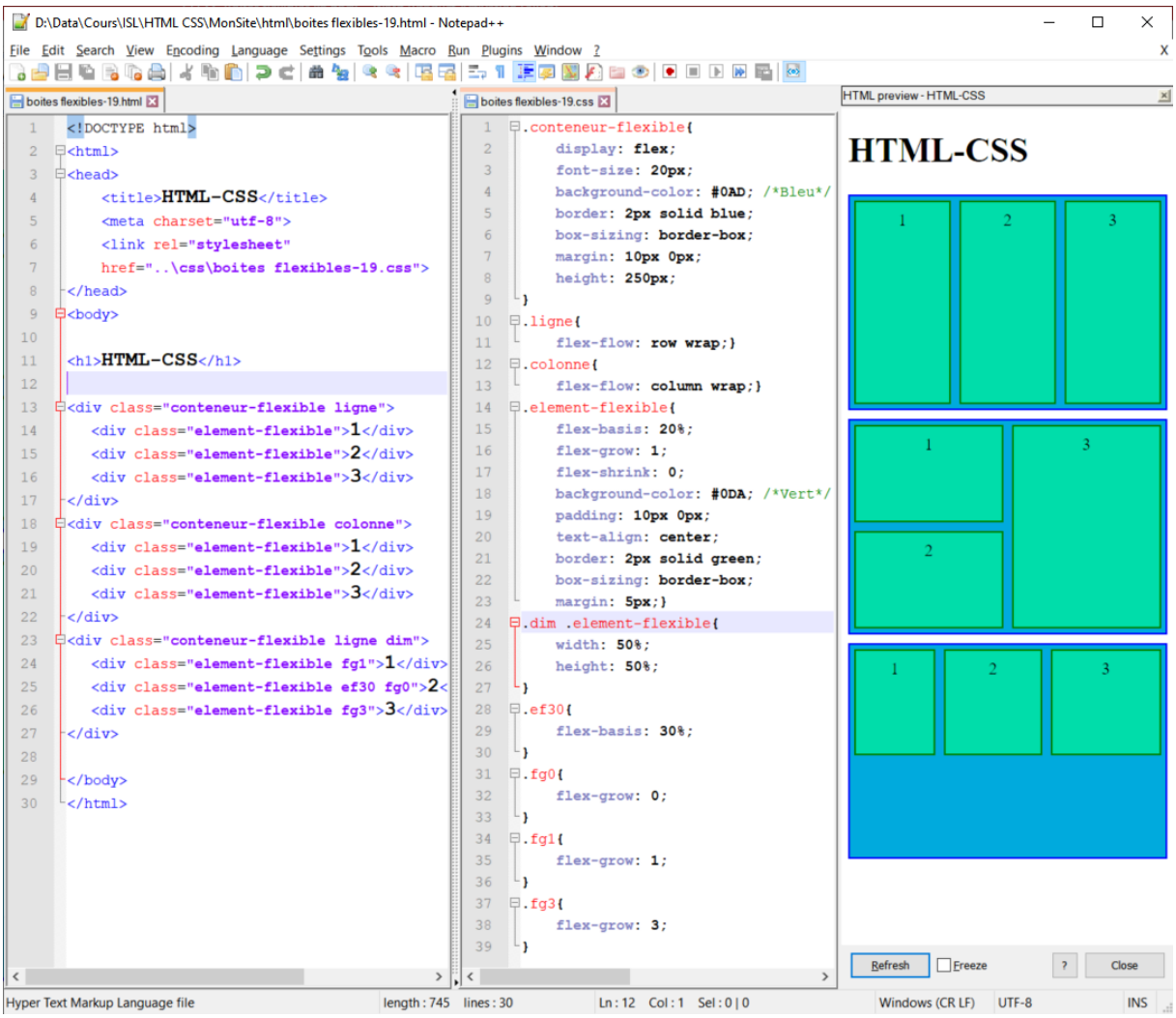
Gérer la capacité des éléments flexibles à grossir avec flex-grow

La propriété flex-grow va nous permettre de définir la capacité des éléments à s'étirer dans leur conteneur pour remplir l'espace vide. Nous allons à nouveau l'appliquer aux éléments flexibles.

On va passer un nombre positif à flex-grow qui va définir la quantité d'espace disponible qu'un élément doit récupérer par rapport aux autres.

La valeur par défaut de flew-grow est 0 ce qui signifie que les éléments n'ont pas le droit de grossir dans leur conteneur.

Reprenons l'exemple précédent et appliquons cette fois-ci un flex-grow : 1 aux éléments flexibles.



Ici, on définit les dimensions de départ de nos éléments ave la propriété flex-basis. Nos éléments auront donc une largeur ou hauteur selon l'axe principal égale à 20% de leur conteneur sauf pour l'élément possédant la classe .ef30 dont la largeur de départ va être égale à 30% du conteneur.

Vous devez bien comprendre ici que flex-basis sert à définir les dimensions de base des éléments flexibles avant un éventuel étirement ou rétrécissement.

On va également laisser la possibilité à nos éléments flexibles de s'étirer pour occuper l'espace vide dans le conteneur avec flew-grow : 1.

L'espace disponible dans le conteneur va donc être équitablement réparti entre les différents éléments de nos deux premiers conteneurs.

Pour notre troisième conteneur, on définit trois comportements de flex-grow différents pour nos trois éléments : notre premier élément va avoir le droit de s'étirer, le deuxième n'aura pas le droit, et le troisième aura le droit et va essayer de récupérer une quantité de l'espace restant dans le conteneur 3 fois plus importante que le premier élément.

Si vous êtes familiers des mathématiques, vous pouvez considérer que les nombres passés à flex-grow représentent une pondération. Sinon, retenez simplement que les nombres passés à flex-grow vont définir combien d'espace libre un élément peut avaler par rapport aux autres éléments.

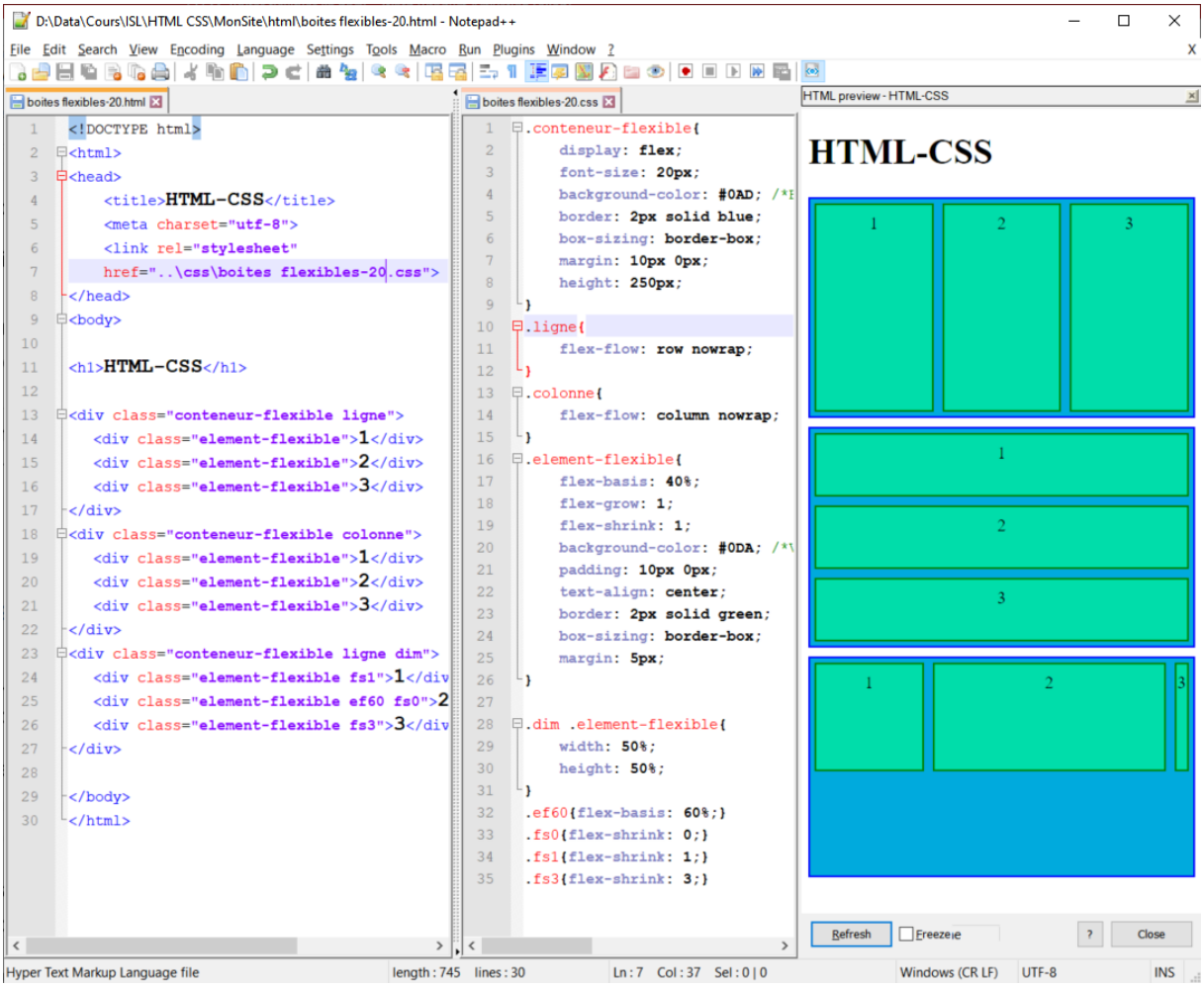
Gérer la capacité des éléments flexibles à rétrécir avec flex-shrink

En plus de pouvoir gérer la capacité des éléments à grossir pour absorber l'espace vide dans un conteneur, nous allons également pouvoir gérer leur capacité à rétrécir pour éviter que des éléments ne dépassent du conteneur avec la propriété flex-shrink.

Cette propriété va s'utiliser de manière similaire à la précédente : nous allons à nouveau pouvoir passer un nombre positif à flex-shrink qui va indiquer la capacité des éléments à rétrécir ainsi que l'importance de leur rétrécissement les uns par rapport aux autres.

Cependant, à la différence de flex-grow, la valeur par défaut de flex-shrink est 1 ce qui correspond à laisser la capacité aux éléments de rétrécir.

Notez par ailleurs qu'un élément ne va pouvoir rétrécir que jusqu'au point où son contenu est prêt à dépasser de l'élément.



Cette fois-ci, on définit une taille de base de 40% pour chacun de nos éléments avec flex-basis : 40% et de 60% pour un élément en particulier dans le troisième conteneur.

On interdit également aux éléments de se placer sur plusieurs lignes ou sur plusieurs colonnes en passant la valeur nowrap pour la propriété flex-wrap aux différents conteneurs afin que les éléments dépassent bien par défaut de leur conteneur.

Finalement, on laisse la possibilité aux éléments de rétrécir afin qu'ils essaient de ne pas dépasser du conteneur avec flex-shrink : 1.

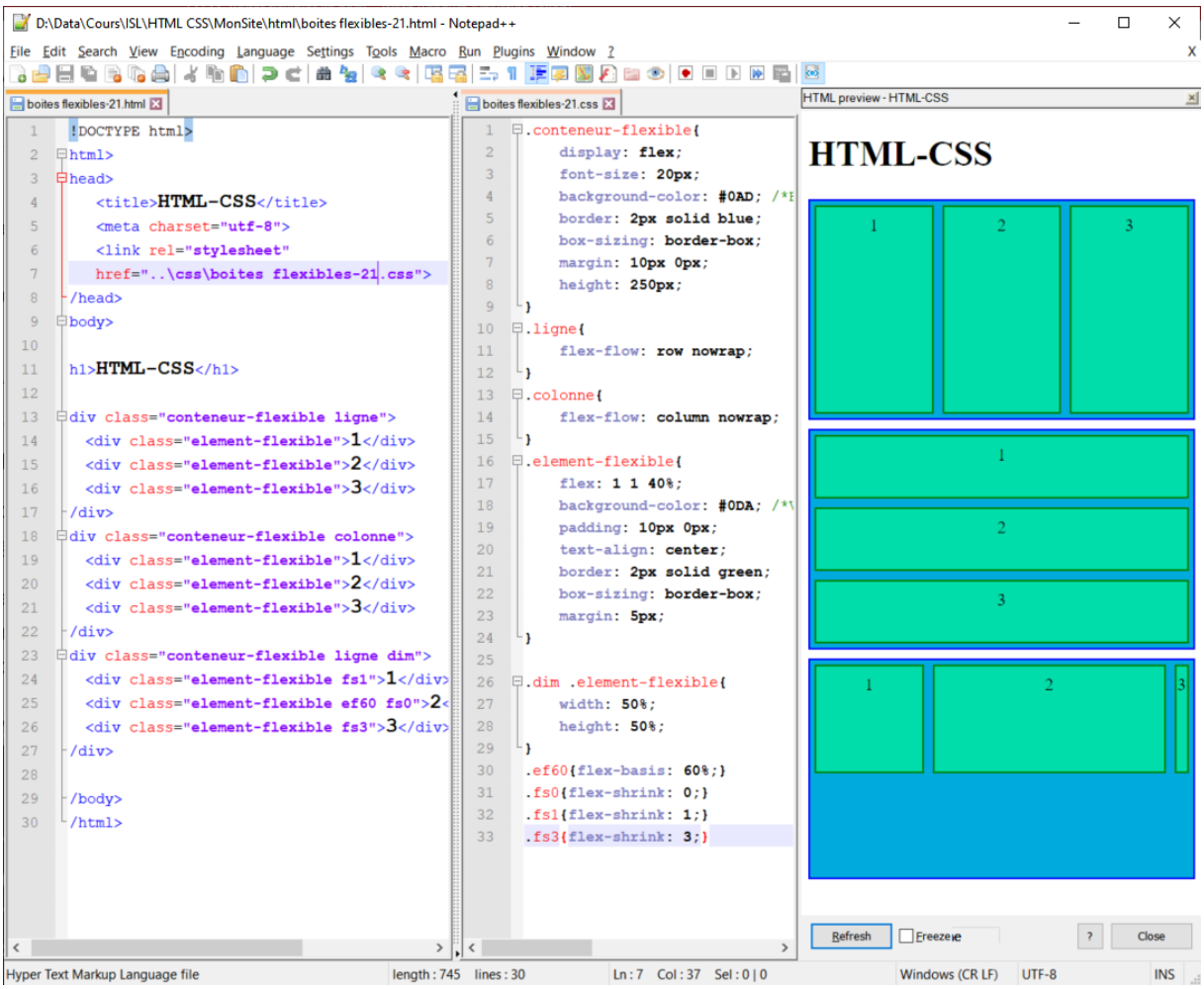
Dans notre troisième conteneur, nous définissons des comportements de rétrécissement différents pour chacun de nos éléments flexibles : le premier élément va pouvoir rétrécir selon un flex-shrink : 1, le deuxième ne pourra pas rétrécir et le troisième va également pouvoir rétrécir selon un flex-shrink : 3, ce qui signifie qu'il va essayer de rétrécir 3 fois plus que le premier élément flexible du conteneur.

La notation raccourcie flex

Nous allons pouvoir définir les trois propriétés flex-grow, flex-shrink et flex-basis d'un coup en utilisant la propriété raccourcie flex.

Pour que la propriété fonctionne correctement, il faudra lui passer *d'abord* la valeur liée à flex-grow, puis celle de flex-shrink et finalement celle de flex-basis.

Faites bien attention une nouvelle fois avec l'utilisation des propriétés raccourcies : si l'une des valeurs de la propriété raccourcie a été définie avant alors elle sera écrasée par la valeur de la propriété raccourcie.


D:\Data\Cours\ISL\HTML CSS\MonSite\html\boites flexibles-21.html - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
boites flexibles-21.html
boites flexibles-21.css
HTML preview - HTML-CSS
HTML-CSS
Refresh
Freeze
Close
Hyper Text Markup Language file
length : 745
lines : 30
Ln : 7
Col : 37
Sel : 0 | 0
Windows (CR LF)
UTF-8
INS

Cas d'utilisation et limites du flexbox

Le modèle des boites flexibles ou flexbox est un modèle de disposition très puissant qui permet de créer un conteneur et des éléments flexibles, c'est-à-dire des éléments qui vont pouvoir se réorganiser lorsque le conteneur change de dimension mais également les uns par rapport aux autres.

Nous allons pouvoir utiliser les propriétés suivantes pour déclarer un conteneur flexible et disposer nos différents éléments :

- display : flex nous permet de créer un conteneur flexible ;
- flex-direction nous permet de choisir l'axe principal et la direction des éléments flexibles selon cet axe ;
- flex-wrap nous permet de laisser la possibilité aux éléments flexibles qui dépassent du conteneur d'aller à la ligne ou sur une nouvelle colonne et de définir la direction des éléments selon l'axe secondaire ;
- order permet de choisir l'ordre d'affichage des éléments ;
- justify-content permet de définir l'alignement des éléments dans leur axe principal ;
- align-items permet de définir l'alignement des éléments dans leur axe secondaire ;
- align-self permet de définir l'alignement d'un élément en particulier dans son axe secondaire;
- align-content permet de définir l'alignement des lignes ou des colonnes selon l'axe secondaire ;
- flex-grow permet de laisser la possibilité aux éléments flexibles de grossir pour absorber l'espace vide dans le conteneur ;
- flex-shrink permet de laisser la possibilité aux éléments flexibles de rétrécir dans le conteneur ;
- flex-basis permet de définir une taille de base pour les éléments flexibles.

En pratique, nous allons pouvoir recourir au modèle des boites flexibles pour résoudre simplement de nombreuses questions d'affichage et de positionnement qui pouvaient poser problème par le passé.

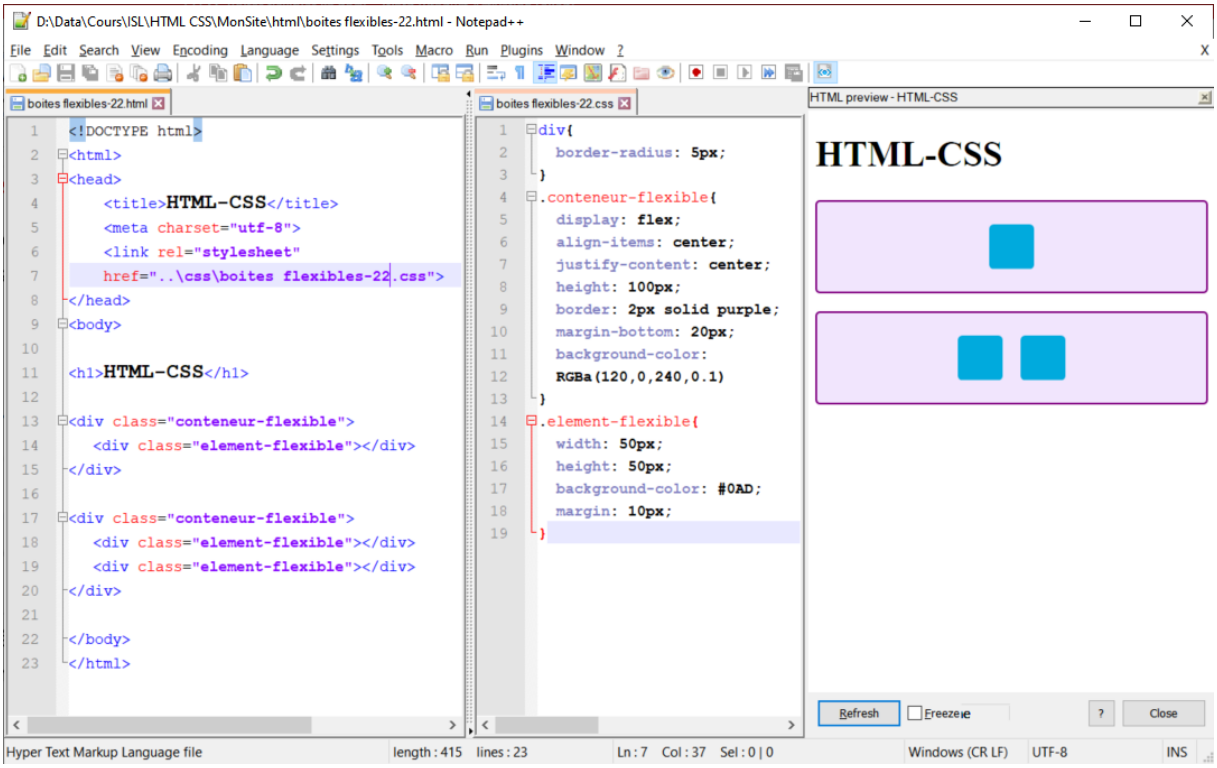
Exemples concrets d'utilisation du flexbox

Voici quelques exemples :

Effectuer un centrage parfait

Pendant très longtemps, il a été très difficile d'obtenir un centrage parfait en toute circonstance pour un élément ou un ensemble d'éléments et notamment un centrage vertical.

Cela devient très facile avec le flexbox puisqu'on va pouvoir centrer les éléments dans leur axe principal et dans leur axe secondaire.



Ici, on centre le ou les éléments par rapport à leur axe principal dans leur conteneur avec justify-content : center et on les centre également dans leur axe secondaire avec align-items : center.

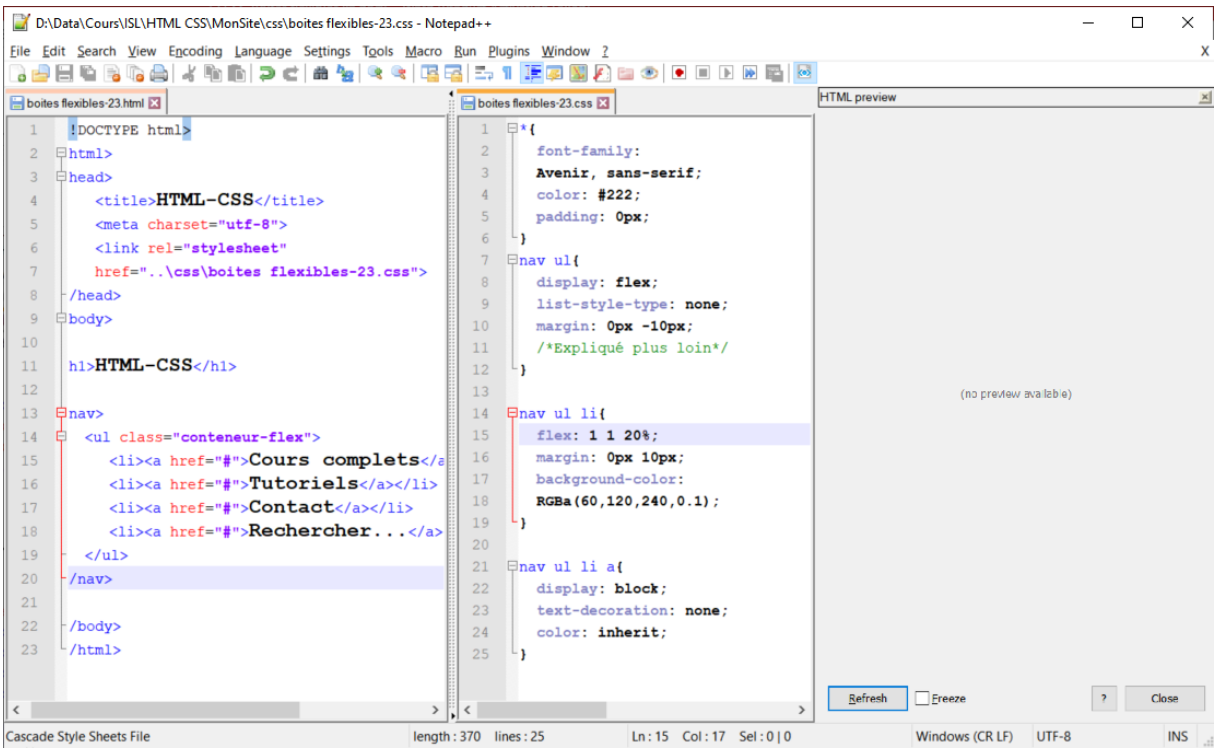
Créer un menu avec le flexbox

Dans les parties précédentes, je vous ai proposé de créer un menu horizontal simple puis un menu déroulant.

Pour que notre menu principal s'affiche de manière horizontale, nous avons utilisé la propriété float.

Ensuite, pour que les éléments du menu s'affichent correctement, nous leur avons attribué une largeur égale à 100% divisée par le nombre d'éléments.

Cette méthode n'était pas parfaite, loin de là. En effet, imaginons qu'on modifie le nombre d'éléments dans le menu : il faudra alors modifier la largeur donnée à chaque élément de menu. De même, selon la taille de certains écrans, on aimerait pouvoir distribuer l'espace dans le conteneur d'une façon ou d'une autre. Nous allons pouvoir gérer tout cela facilement avec le flexbox.



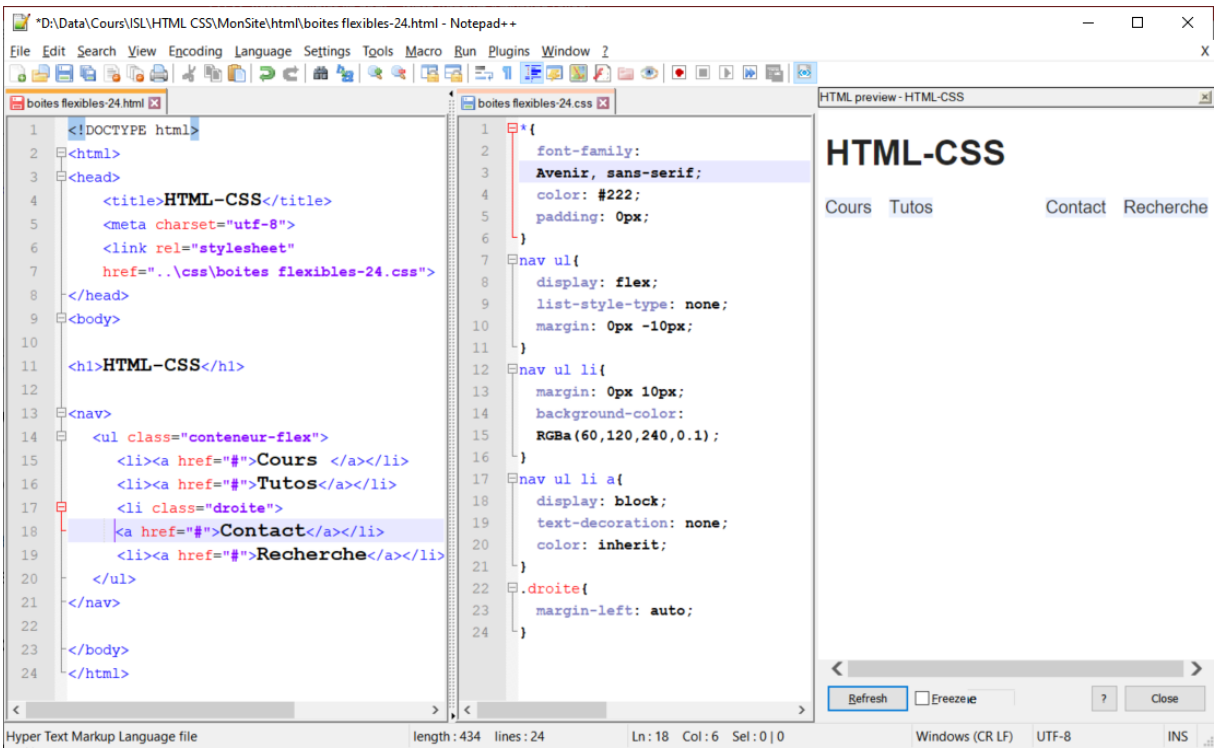

```
!DOCTYPE html>
html>
head>
    <title>HTML-CSS</title>
    <meta charset="utf-8">
    <link rel="stylesheet"
    href="..\css\boites flexibles-23.css">
/head>
body>

h1>HTML-CSS</h1>

nav>
  <ul class="conteneur-flex">
      <li><a href="#">Cours complets</a
      <li><a href="#">Tutoriels</a></li>
      <li><a href="#">Contact</a></li>
      <li><a href="#">Rechercher...</a>
  </ul>
/nav>

/body>
/html>
```

```
*{
  font-family:
  Avenir, sans-serif;
  color: #222;
  padding: 0px;
}
nav ul{
  display: flex;
  list-style-type: none;
  margin: 0px -10px;
  /*Expliqué plus loin*/
}

nav ul li{
  flex: 1 1 20%;
  margin: 0px 10px;
  background-color:
  RGBa(60,120,240,0.1);
}

nav ul li a{
  display: block;
  text-decoration: none;
  color: inherit;
}
```

Ici, on attribue un flex : 1 1 0 à nos éléments de menu. Cela signifie que nos différents éléments de menu vont tous avoir la même taille de base avec flex-basis : 0 et vont pouvoir grandir ou rétrécir pour absorber l'espace vide du menu ou pour faire de la place à de nouveaux éléments.

Grouper des éléments

Le flexbox va également nous permettre simplement de créer des groupes d'éléments et de les aligner d'un côté ou de l'autre du conteneur. Pour cela, nous allons utiliser la propriété margin et sa valeur auto qui va faire que la marge va consommer tout l'espace vide dans le conteneur.



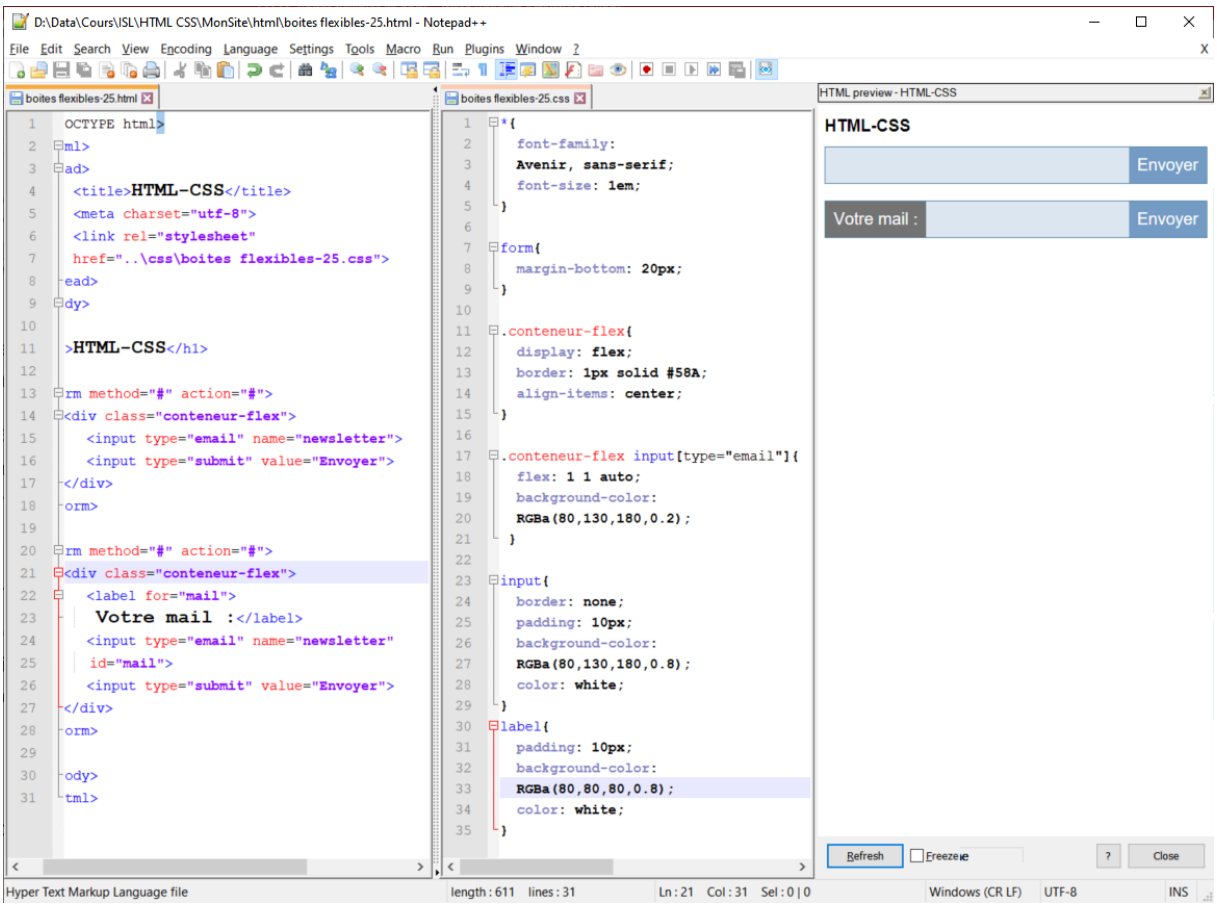
Ici, on attribue un margin-left : auto au troisième élément de notre menu. Cela fait que sa marge gauche va consommer tout l'espace vide du conteneur et va donc repousser l'élément et les éléments suivants sur la droite. Cela va être particulièrement utile pour la création de menus.

Styliser les formulaires

Imaginons que l'on souhaite créer un formulaire d'inscription à une newsletter. La seule information qu'on va demander aux utilisateurs va être leur adresse mail.

Ici, plutôt que d'afficher notre unique input sur une ligne puis le bouton d'envoi du formulaire en dessous, on aimerait tout avoir sur la même ligne pour des questions d'ergonomie.

Cependant, pour que nos éléments de formulaire alignés s'affichent toujours bien, il va falloir qu'ils se redimensionnent les uns par rapport aux autres en fonction de l'espace disponible. On va pouvoir faire cela très simplement avec le flexbox.



Ici, on applique flex : 1 1 auto à notre input de type email afin que celui-ci se redimensionne en fonction des éléments autour et pour qu'il occupe toujours tout l'espace vide dans le conteneur.

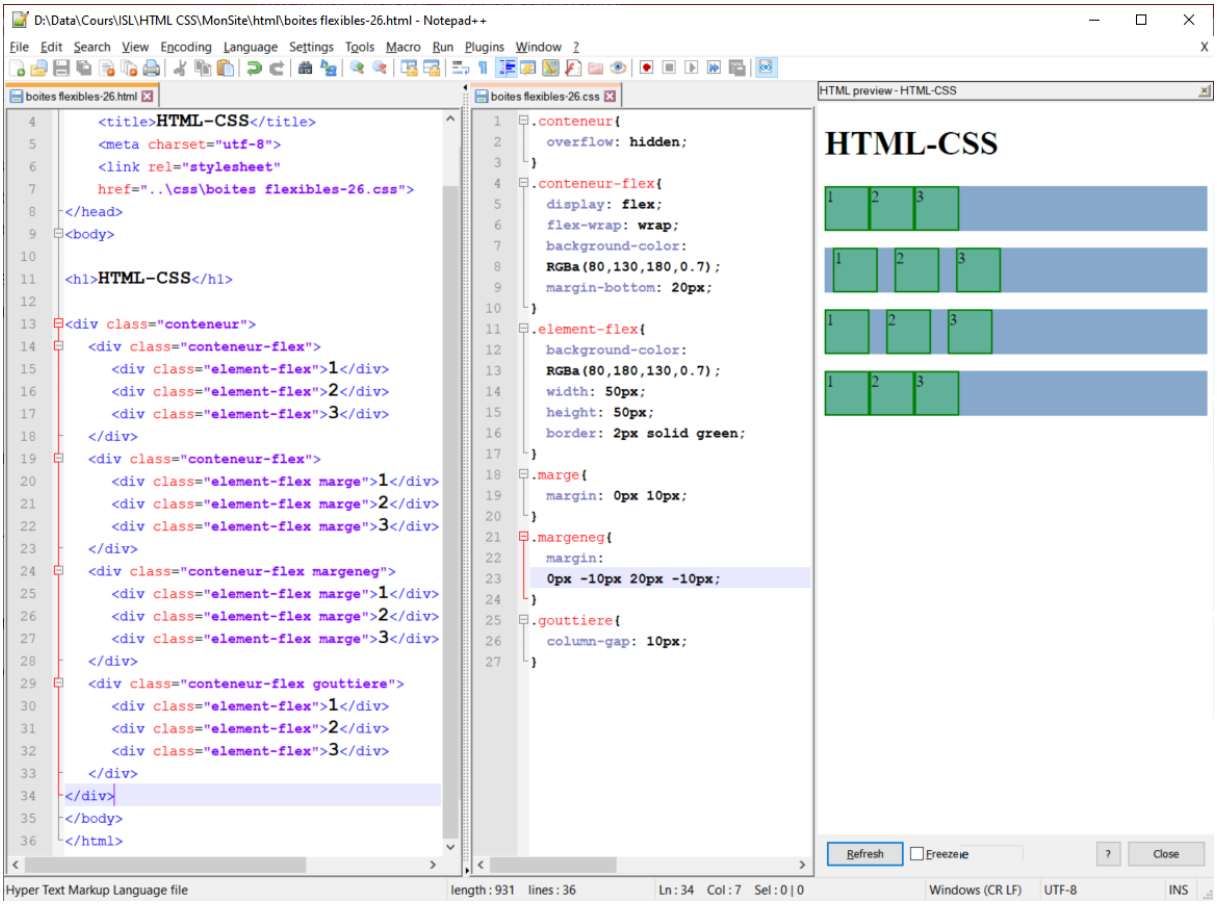
Les limites du flexbox

Le modèle des boites flexibles est à l'heure actuelle l'un des modèles de disposition les plus puissants que l'on puisse utiliser. Cependant, comme ce modèle est relativement récent, l'implémentation n'est pas encore parfaite et certaines propriétés manquent pour le moment.

Cela va être notamment le cas pour la gestion des gouttières, c'est-à-dire de l'espace entre les éléments flexibles. En effet, par défaut, nos différents éléments flexibles vont venir se coller les uns aux autres.

Pour créer des gouttières, c'est-à-dire pour créer un espace entre ces éléments, j'ai jusqu'à présent utilisé la propriété margin. Le problème ici est que cela va également créer un espace entre les éléments et le bord du conteneur, ce qui n'est pas toujours le comportement voulu.

Pour annuler cet espace, la solution actuellement admise est d'attribuer des marges négatives au conteneur qui vont venir contrebalancer les marges attribuées aux éléments flexibles.

Notez qu'on ne sera bientôt plus obligé d'utiliser cette technique puisque deux nouvelles propriétés qui vont nous permettre justement de créer des gouttières sont en train d'être ajoutées au modèle

des boites : les propriétés row-gap et column-gap. Ces propriétés ne sont néanmoins pour le moment pas supportées par la plupart des navigateurs.