Position et affichage des éléments

Rédigé par: Jean-Luc Colson
Date première rédaction: Janvier 2020

SUIVI DES MODIFICATIONS A LA FICHE			
Série	Date	Page(s) modifiée(s)	Raison

Contents

Gérer l'affichage avec la propriété CSS display

La propriété CSS display est une propriété très puissante puisqu'elle va nous permettre de modifier la façon dont un élément va s'afficher dans la page : en ligne, sous forme de bloc, etc. et donc la façon dont il va se comporter avec ses voisins.

Nous avons déjà eu l'occasion de parler de l'affichage des éléments dans la leçon expliquant les différences entre les éléments de niveau block et de niveau inline.

En CSS3, la propriété display accepte de nombreuses valeurs différentes ce qui va nous permettre de choisir précisément comment chaque élément HTML doit être affiché dans la page.

Dans cette nouvelle leçon, nous allons expliquer comment fonctionne la propriété display en détail et constater son impact sur l'affichage des éléments en utilisant ses différentes valeurs.

Rappels sur la définition du type d'affichage d'un élément par défaut

Le type d'affichage d'un élément va toujours être défini en CSS via la propriété display. Si cette propriété n'est pas explicitement renseignée pour un élément, la valeur par défaut de qui est display: inline sera appliquée à l'élément.

Cependant, vous devez bien comprendre que lorsque vous ouvrez une page dans un navigateur, le navigateur va appliquer des styles par défaut à chaque élément HTML afin d'améliorer le rendu et pour servir de solution de secours si des styles n'ont pas été appliqués par les développeurs de la page.

Parmi ces styles par défaut appliqués par n'importe quel navigateur (et dépendant de chaque navigateur, attention !) se trouve la définition du type d'affichage ou du display pour chaque élément.

Pour savoir quel type de display appliquer par défaut à chaque élément, la plupart des navigateurs sérieux et connus se basent sur les recommandations du W3C (ou du WHATWG, car il est souvent difficile de savoir lequel de ces deux groupes est à l'origine d'une recommandation)

Attention cependant : encore une fois, ce ne sont que des recommandations et chaque navigateur est libre de ne pas en tenir compte et de définir une autre valeur de display pour chaque élément, ce qui peut en pratique arriver pour certains éléments particuliers ou dans certaines situations peu courantes.

Ceci étant dit, nous allons maintenant apprendre à modifier nous-mêmes le type d'affichage d'un élément en utilisant la propriété display pour ne pas avoir à dépendre de l'implémentation par les différents navigateurs.

Inner display et outer display

Plus haut, j'ai dit que la propriété CSS display affectait la façon dont un élément s'affichait dans une page.

Plus précisément, cette propriété affecte la façon dont l'élément va générer les boites le composant. Je vous rappelle qu'en HTML5 et CSS3, tout élément HTML doit être vu comme un empilement de boites :

- Boite n°1 (la plus interne) : contenu de l'élément ;
- Boite n°2 : boite n°1 + marges internes ;
- Boite n°3 : boite n°2 + bordures ;
- Boite n°4 : boite n°3 + marges externes.

La propriété display va ainsi impacter la génération de l'outer display qui correspond au comportement de l'élément globalement (qu'on peut représenter via la boite globale ou boite n°4) dans le flux de la page et donc par rapport aux autres éléments mais également de l'*inner display* de l'élément qui correspond à la façon dont l'élément va créer ses boites internes (excepté pour les éléments remplacés qui sont en dehors de la portée du CSS).

Ainsi, la propriété display va toujours implicitement recevoir deux valeurs : une première pour définir l'*outer display* et une seconde pour définir l'*inner display*.

Cependant, en pratique, nous ne mentionnerons explicitement qu'une valeur cet laisserons le CSS définir la deuxième valeur par défaut. Retenez toutefois bien que la propriété display définira quand même toujours un *outer display* et un *inner display*.

Ainsi, lorsqu'on mentionne display : block par exemple, la « vraie » valeur complète de display va être display : block flow.

Dans le cas où nous voudrions une deuxième valeur autre que la valeur par défaut, nous n'aurons pas non plus besoin de préciser deux valeurs en pratique puisque le CSS a implémenté des mots clefs composés comme inline-block qui sont là pour répondre à ces cas.

Pour information, voici les versions « raccourcies » des valeurs données à display que vous devriez toujours utiliser et les valeurs complètes pour référence. Par d'inquiétude, nous allons indiquer par la suite à quoi correspond chaque valeur.

Valeur raccourcie	**Valeur complète**	**Comportement**
none	—	L'élément ne s'affiche pas et ne génère aucune boite
contents	—	L'élément ne génère aucune boite à l'affichage mais ses enfants oui

Valeur raccourcie	Valeur complète	Comportement
block	block flow	Elément de niveau block (block-level) et boites internes de type block (block container)
flow-root	block flow-root	Elément de niveau block (block-level) et boites internes de type block (block container) établissant un nouveau contexte de formatage
inline	inline flow	Elément de niveau inline (inline-level) et boites internes de type inline
inline-block	inline flow-root	Elément de niveau inline (inline-level) et boites internes de type block (block container)
run-in	run-in flow	Elément de type run-in (inline avec des règles spéciales)
list-item	block flow list-item	Elément block-level avec block container de type block qui crée également une boite contenant un marqueur
inline list-item	inline flow list-item	Elément inline-level qui crée également une boite contenant un marqueur
flex	block flex	Elément block-level avec boites internes flexibles (flex container)
inline-flex	inline flex	Elément inline-level avec boites internes flexibles (flex container)
table	block table	Elément block-level avec boites internes de type table
inline-table	inline table	Elément inline-level avec boites internes de type table
grid	block grid	Elément block-level avec boites internes de type grille (grid)

Valeur raccourcie	Valeur complète	Comportement
inline-grid	inline grid	Elément inline-level avec boites internes de type grille (grid)

Les valeurs de display-outside de display

Nous allons déjà pouvoir commencer par définir l'outer display d'un élément avec display, c'est-à-dire le type d'affichage de l'élément en soi et son comportement visuel par rapport aux autres.

Si on exclut les valeurs particulières display : none et display : contents, nous ne pouvons définir l'outer display d'un élément qu'avec trois valeurs différentes :

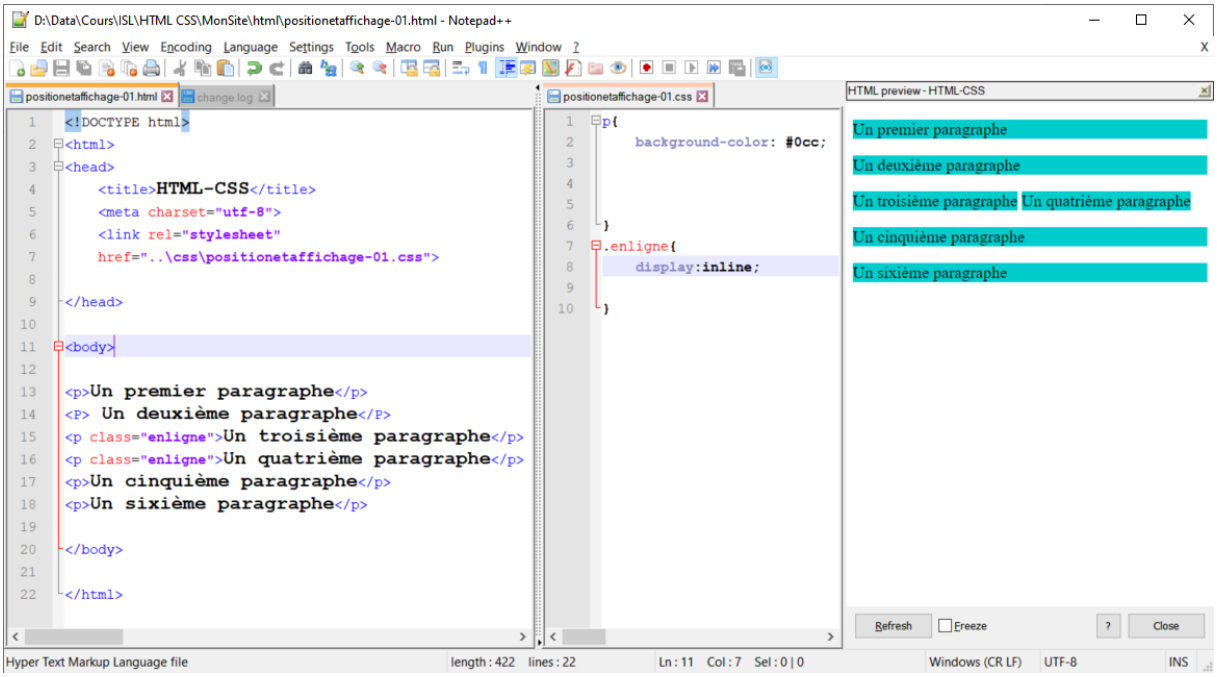
- Avec la valeur inline : l'élément va être de niveau inline ;
- Avec la valeur block : l'élément va être de niveau block ;
- Avec la valeur run-in : l'élément va être de niveau inline avec des règles particulières.

Notez que pour chacune de ces trois valeurs outer display, la valeur de l'inner display associée par défaut est flow.

Display : inline

En précisant un display : inline (qui est la valeur raccourcie de display : inline flow), on définit un élément de niveau inline (inline-level element) ou tout simplement de « type » inline. Un élément de niveau inline a les caractéristiques suivantes :

- Un élément de type inline ne va occuper que la largeur nécessaire à l'affichage de son contenu par défaut ;
- Les éléments de type inline vont venir essayer de se placer en ligne, c'est-à-dire à côté (sur la même ligne) que l'élément qui les précède dans le code HTML ;
- Un élément de type inline peut contenir d'autres éléments de type inline mais ne peut pas contenir d'éléments de type block.

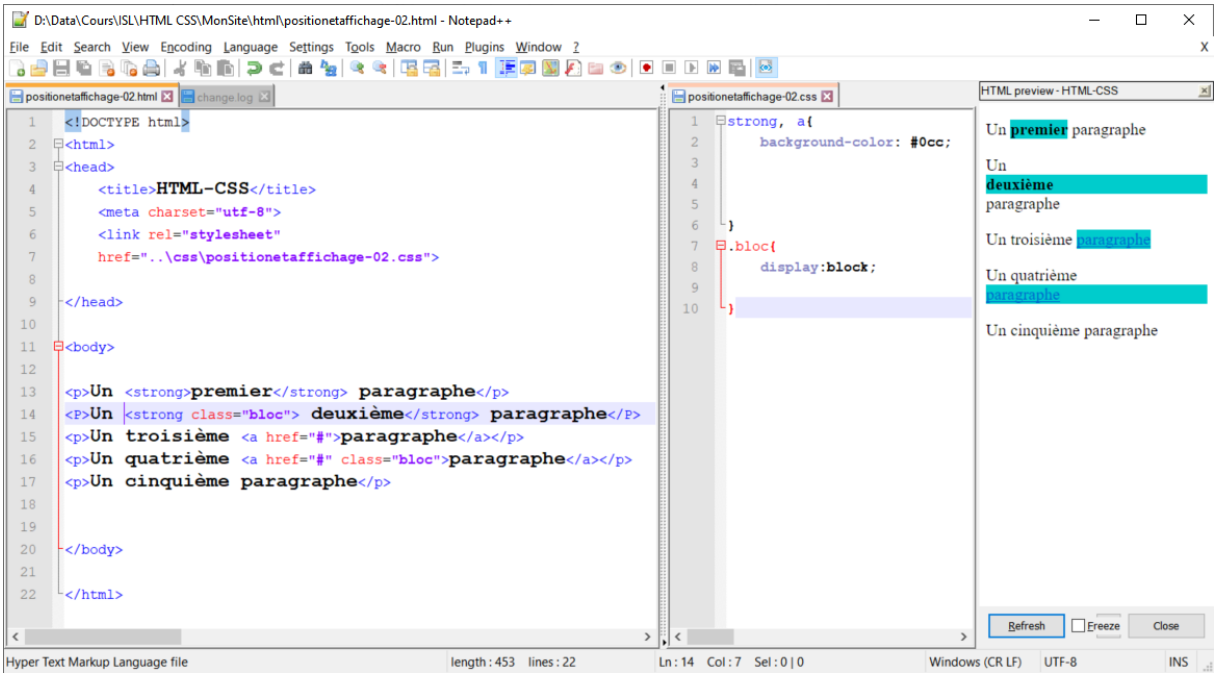


Par défaut, les éléments p possèdent un display : block. Ici, nous définissons un display : inline pour deux d'entre eux ; ces deux paragraphes en particulier vont donc se comporter comme des éléments inline.

Display : block

En précisant un display : block (valeur raccourcie de display : block flow), on définit un élément de niveau block (block-level element) ou encore de « type » block. Un élément de niveau block va posséder les caractéristiques suivantes :

- Un élément de type block va toujours prendre toute la largeur disponible au sein de son élément parent (ou élément conteneur) ;
- Un élément de type block va toujours « aller à la ligne » (créer un saut de ligne avant et après l'élément), c'est-à-dire occuper une nouvelle ligne dans une page et ne jamais se positionner à côté d'un autre élément par défaut ;
- Un élément de type block peut contenir d'autres éléments de type block ou de type inline.



Les éléments strong et a ont un type d'affichage inline qui leur est attribué par défaut. Ici, nous changeons ce type d'affichage pour un display : block pour notre deuxième élément strong et deuxième lien. Ces deux éléments vont donc se comporter comme des éléments de type block.

Display : run-in

En précisant un display : run-in (valeur raccourcie de display : run-in flow), on définit un élément de type inline avec un comportement spécial : l'élément va essayer de fusionner / s'insérer dans l'élément de type block suivant.

Notez que cette valeur ne fait pas encore partie des recommandations officielles du W3C et est toujours en développement. A éviter pour un développement en production donc.

Les valeurs de display-inside de display

Comme je l'ai précisé plus haut, nous ne préciserons généralement qu'une seule valeur à display et laisserons le CSS appliquer la deuxième valeur par défaut. Dans la majorité des cas, ce sera la valeur liée à l'outer display qui sera précisée.

Cependant, pour certains éléments particuliers et dans certains contextes nous passerons plutôt une valeur d'inner display à la propriété display (et lui laisserons donc appliquer l'outer display lié par défaut).

Ce sont ces valeurs qui vont nous intéresser ici et notamment les valeurs d'affichage table, flex, grid et list-item.

Inner display: table

En précisant un display : table a un élément, l'élément va visuellement se comporter comme un tableau. Nous étudierons la création de tableaux en HTML plus tard dans ce cours.

La valeur complète du display par défaut est display : block table. A partir de là, vous pouvez déduire qu'un tableau en HTML a un outer display de type block par défaut.

Notez ici que les tableaux vont avoir des structures d'affichage complexes puisqu'ils vont être composés de lignes et de cellules.

Pour rendre complètement le comportement visuel d'un tableau, nous allons également pouvoir utiliser les valeurs suivantes pour display sur certains éléments en répliquant la structure d'un tableau « normal » même si nous essayerons d'éviter de faire ça tant que possible pour des raisons évidentes de sémantique.

- display : table-header-group : l'élément se comporte visuellement comme un élément thead;
- display : table-footer-group : l'élément se comporte visuellement comme un élément tfoot ;
- display : table-row-group : l'élément se comporte visuellement comme un élément tbody ;
- display : table-row : l'élément se comporte visuellement comme un élément tr ;
- display : table-cell : l'élément se comporte visuellement comme un élément td ;
- display : table-column-group : l'élément se comporte visuellement comme un élément colgroup ;
- display : table-column : l'élément se comporte visuellement comme un élément col.

Nous pouvons également utiliser display : table-caption pour qu'un élément se comporte comme un élément caption. Cette valeur de display va créer un block avec un comportement relatif à celui du tableau.

Inner display: flex

En attribuant un display : flex a un élément, l'élément va visuellement se comporter comme une boite flexible, ce qui signifie que l'élément va se comporter comme un élément de type block pour l'outer display mais que l'intérieur de l'élément va suivre le modèle des boites flexibles ou flexbox (il va établir un nouveau contexte de formatage de type flex).

Nous allons consacrer une leçon au modèle des boites flexibles ou « flexbox » CSS dans ce cours car ce modèle est très intéressant pour créer des pages qui vont s'adapter à tous les écrans ou « responsives ».

La valeur complète de display : flex est display : block flex.

Inner display: grid

En attribuant un display : grid a un élément, l'élément va visuellement se comporter comme une grille : l'élément en soi va être de niveau block et va créer en interne un contexte de formatage de type grille ou grid c'est-à-dire disposer son contenu selon le modèle des grilles.

La valeur complète du display : grid est display : block grid.

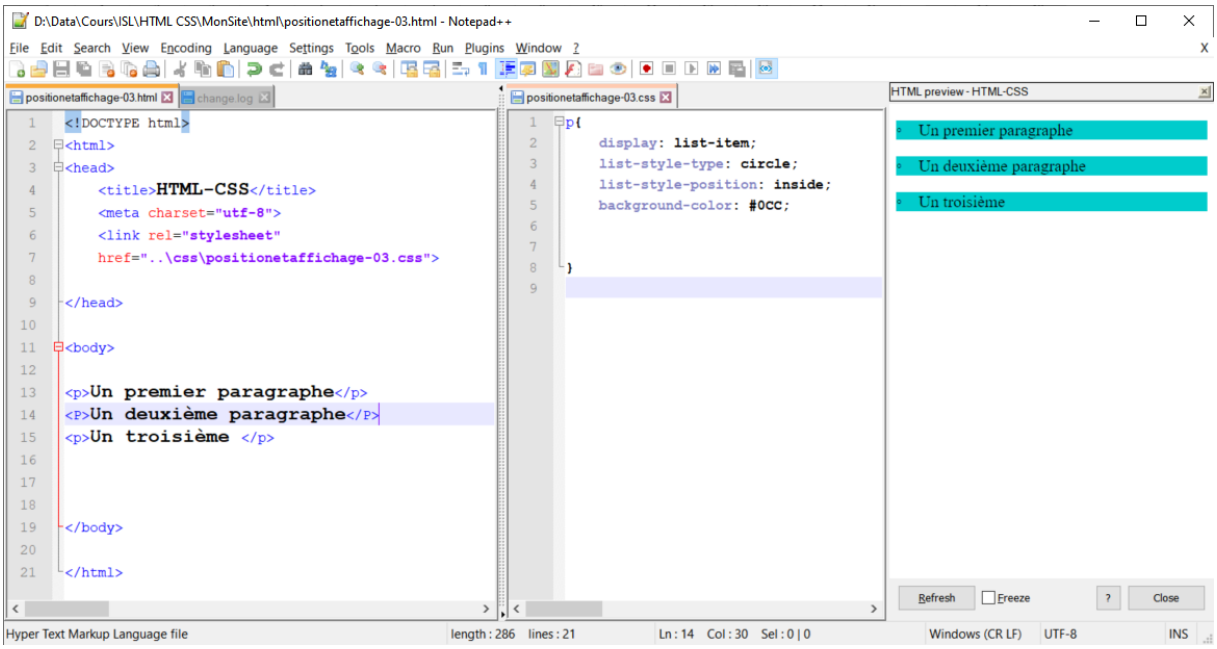
Un cas particulier d'affichage : le display : list-item

Lorsque l'on crée un élément de liste en HTML avec l'élément li, on crée au final deux boites : une boite contenant la puce ou marqueur et une autre contenant le contenu textuel de notre élément de liste.

En attribuant un display : list-item a un élément, on va pouvoir recréer ce comportement et faire en sorte que l'élément se comporte comme un élément de liste et génère un marqueur et une boite de type block par défaut.

La valeur complète d'un display : list-item est display : block flow list-item. Ici, la valeur complète de display est composée de trois valeurs.

C'est tout à fait normal au sens où le mot clef list-item ne sert véritablement qu'à générer un marqueur de liste et ne dicte ni le comportement de l'outer display ni celui de l'inner display.



Ici, nous attribuons un display : list-item à tous les paragraphes de nos pages. Ces derniers vont donc se comporter comme des éléments de liste et avoir chacun un marqueur ou puce.

On en profite pour définir l'apparence et la position des puces car par défaut la position est outside et comme mes paragraphes sont collés au bord gauche de ma page les puces seraient en dehors de la page. Notez qu'on pourrait également conserver un list-style-position: outside et ajouter une margin-left.

Les valeurs de display composées héritées du CSS2 : inline-block, etc.

Au départ, en CSS2, on avait imaginé une syntaxe avec un mot clef unique à fournir en valeur de la propriété display. Dans ce contexte, cependant, comment faire pour ne modifier que l'outer display ou que l'inner display quand on veut obtenir un comportement qui n'est pas le comportement par défaut ?

Pour faire cela ont été introduit des mots clefs composés dont nous héritons aujourd'hui comme par exemple le mot clef inline-block qui est l'équivalent d'un display : inline flow-root.

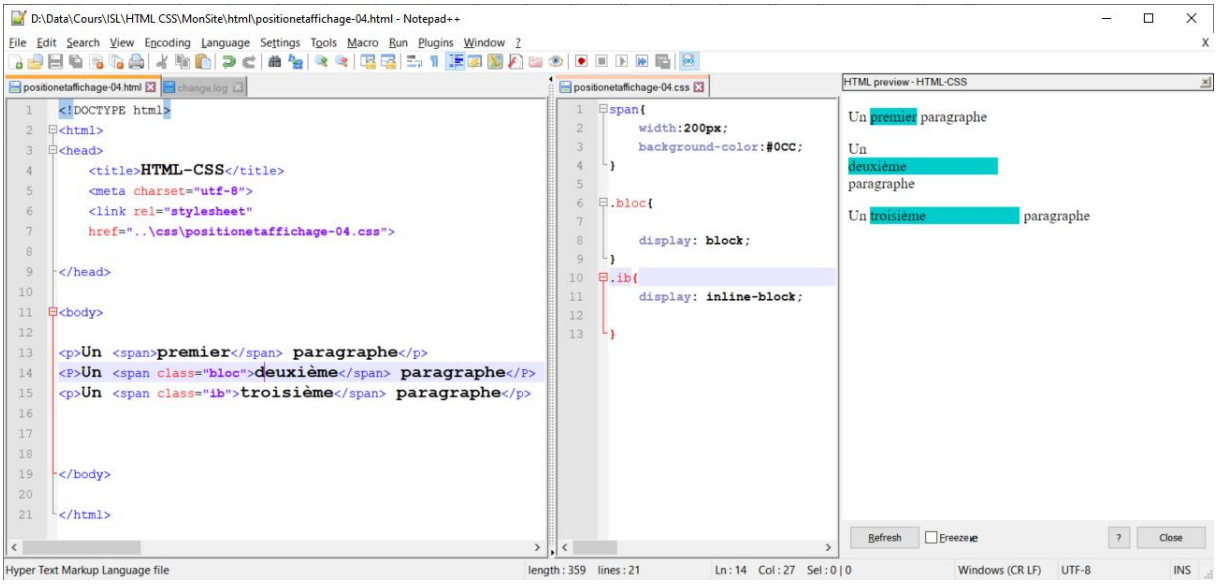
Ces mots clefs composés font toujours partie des recommandations du W3C et vous êtes donc invités à les utiliser même s'ils ne sont finalement que le reflet des limitations passées.

Display : inline-block

En attribuant un display : inline-block a un élément, l'élément en soi va être de niveau inline (l'outer display est inline) tandis que le contenu de l'élément va se comporter comme un block.

La valeur display : inline-block est l'équivalent de display : inline flow-root.

La valeur display : inline-block va ainsi être intéressante pour placer des éléments en ligne tout en pouvant mettre en forme le contenu de ceux-ci et notamment pour pouvoir attribuer une taille précise à chaque contenu (je vous rappelle que cela n'est possible qu'avec des éléments / boites de type block).



Ici, on commence par définir une largeur fixe pour nos différents éléments span avec la propriété width.

On n'indique pas de valeur particulière pour le display de notre premier élément span. Celui-ci aura donc un display : inline qui est le type d'affichage par défaut des éléments span. En tant qu'élément inline, il va ignorer la propriété width.

On indique un display : block pour notre deuxième span. Il va donc se comporter comme un élément de type block et occuper sa propre ligne mais tenir compte de la largeur passée.

On passe finalement un display : inline-block à notre troisième élément span. Celui-ci va donc se comporter comme un élément inline pour son outer display mais ses boites internes vont pouvoir être mises en forme comme un élément block. L'élément va donc rester en ligne mais nous allons pouvoir définir une largeur précise pour celui-ci.

Display : inline-table

En attribuant un display : inline-table à un élément, celui-ci va se comporter comme un tableau mais de niveau inline.

La valeur display : inline-table va donc nous permettre de placer un tableau à la suite d'un autre contenu plutôt que sur une ligne qui lui est propre.

La valeur display : inline-table est l'équivalent de display : inline table.

Display : inline-flex

En attribuant un display : inline-flex à un élément, celui-ci va se comporter comme un élément de niveau inline et organiser son contenu selon le modèle des boites flexibles. Notez que les flex-items ne sont pas affectés par le type du conteneur : ils vont continuer à se comporter comme des boites de niveau block (tout en possédant certaines propriétés liées aux inline-block).

La valeur display : inline-flex est l'équivalent de display : inline flex.

Display : inline-grid

De la même façon, attribuer un display : inline-grid a un élément va le faire se comporter comme un élément de type inline qui va organiser son contenu selon le modèle des grilles.

La valeur display : inline-grid est l'équivalent de display : inline grid.

Display : inline list-item

Attribuer un display : inline list-item a un élément de liste va modifier le comportement d'affichage de l'élément qui sera alors affiché comme un élément de niveau inline qui va également créer une boite contenant un marqueur.

Les valeurs de non affichage display : none et display : contents

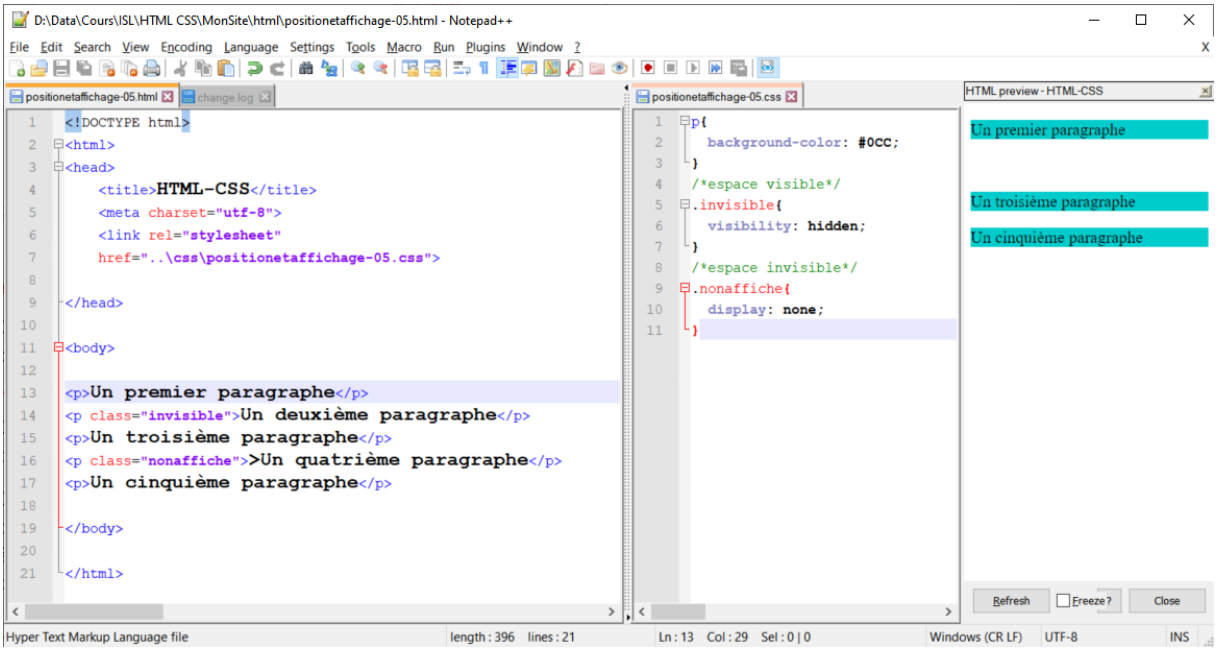
Finalement, nous allons terminer avec deux valeurs particulières de display : display : none et display : contents. Ces valeurs sont particulières car elles vont nous permettre de ne pas afficher certains éléments ou boites.

Display : none

Si un élément possède un display : none, il ne sera tout simplement pas affiché dans la page et les autres éléments se comporteront comme s'il n'existait pas (il ne prendra aucune place dans la page).

Il convient de ne pas confondre display : none et visibility : hidden. La propriété visibility va en effet pouvoir faire disparaitre (visuellement) un élément mais l'espace qu'il occupe va être conservé dans la page à la différence de display : none.

Notez qu'en appliquant un display : none à un élément, ses éléments enfants ne seront pas non plus affichés.



Dans l'exemple ci-dessus, on utilise la propriété visibility et sa valeur hidden pour cacher un de nos éléments p. Vous pouvez cependant remarquer que si le contenu de l'élément ne s'affiche pas, la place qui lui était réservée dans le document est conservée.

Cela ne va pas être le cas pour le paragraphe auquel on a appliqué un display : none : non seulement l'élément ne va pas s'afficher mais le document va faire comme si le paragraphe n'existait pas du tout et celui-ci ne va donc pas prendre d'espace dans la page.

Display : contents

La valeur display : contents va également nous permettre de faire disparaitre les boites générées par un élément mais, à la différence de display : none, le contenu textuel de l'élément va être affiché normalement et les enfants de cet élément vont continuer à générer des boites et à apparaitre dans la page de manière normale.

Notez que cette valeur ne fait pas encore partie des recommandations du W3C et est toujours en cours de développement et peut donc être sujette à des changements ou à un abandon.

Gérer le positionnement avec la propriété CSS position

La propriété position est une propriété CSS très puissante qui va nous permettre de définir un type de positionnement pour nos éléments.

On va ainsi pouvoir positionner un élément relativement à partir de sa position par défaut ou de façon absolue par rapport à un point donné dans la page en utilisation position conjointement avec les propriétés top, left, bottom et right.

Dans cette leçon, nous allons découvrir les différentes valeurs qu'on va pouvoir donner à position et apprendre à les utiliser intelligemment en tentant de comprendre leurs implications.

Le fonctionnement et les valeurs de la propriété position

Nous allons pouvoir gérer et modifier le type de positionnement d'un élément HTML grâce à la propriété CSS position.

La propriété position ne va pas nous permettre de positionner un élément en soi dans une page mais simplement de définir un type de positionnement grâce aux valeurs suivantes :

- position : static ;
- position : relative ;
- position : absolute ;
- position : fixed ;
- position : sticky.

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

Ces quatre propriétés vont pouvoir prendre des valeurs absolue ou relative et vont servir à indiquer où le coin supérieur gauche de la boite représentant un élément doit être positionné par rapport à un certain point de référence (50px à droite et 30px en dessous de ce point par exemple).

Le type de positionnement défini pour l'élément va servir à définir ce point de référence et va donc affecter le fonctionnement de ces propriétés qui vont produire des résultats différents.

Les types de positionnement d'un élément HTML dans une page

Il existe trois types de positionnement en CSS. Il est très intéressant de les connaitre et de les comprendre afin de mieux comprendre comment fonctionne le CSS et comment les différents éléments vont venir se positionner les uns par rapport aux autres.

Connaitre et comprendre les types de positionnement va également nous permettre de comprendre comment fonctionne la propriété CSS position puisque selon la valeur donnée à code>position, un élément HTML va se conformer à un type de positionnement plutôt qu'un autre.

1. Le premier type de positionnement est ce qu'on pourrait appeler le positionnement « normal » ou par défaut des éléments. Ici, les éléments vont respecter le flux normal de la page et s'y intégrer sans le casser. *Ainsi, un élément de type block sera formaté comme tel (c'est-à-dire qu'il occupera tout l'espace possible et se placera à la ligne), un élément de type inline n'occupera que l'espace nécessaire et etc. ;*

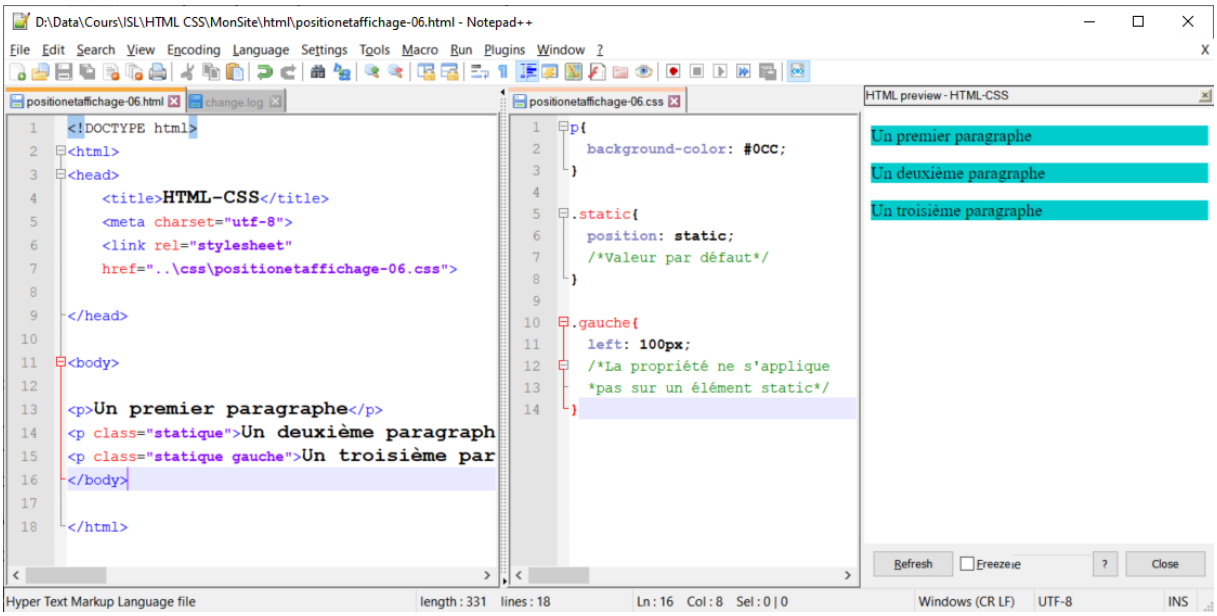
2. Ensuite, on va également pouvoir faire flotter des éléments HTML avec la propriété float. Ce type de positionnement est particulier puisque l'élément flotté va être retiré du flux normal de la page pour être repositionné ailleurs (généralement à gauche ou à droite) et va également permettre à d'autres éléments de type inline de se positionner à côté de notre élément flotté ;

3. Finalement, on va pouvoir positionner un élément de manière absolue dans notre page. Avec le type de positionnement absolu, un élément est complètement retiré du flux normal de la page pour être placé absolument par rapport à son élément parent direct et va ainsi pouvoir potentiellement passer au-dessus d'autres contenus.

Position : static

La valeur static est la valeur par défaut de la propriété position. Ainsi, par défaut, tous les éléments HTML sont positionnés de manière static. Un élément HTML positionné avec position : static sera positionné selon le flux normal de la page.

Notez ici que les propriétés top, left, bottom et right n'auront aucun effet sur les éléments positionnés avec position : static.



Position : relative

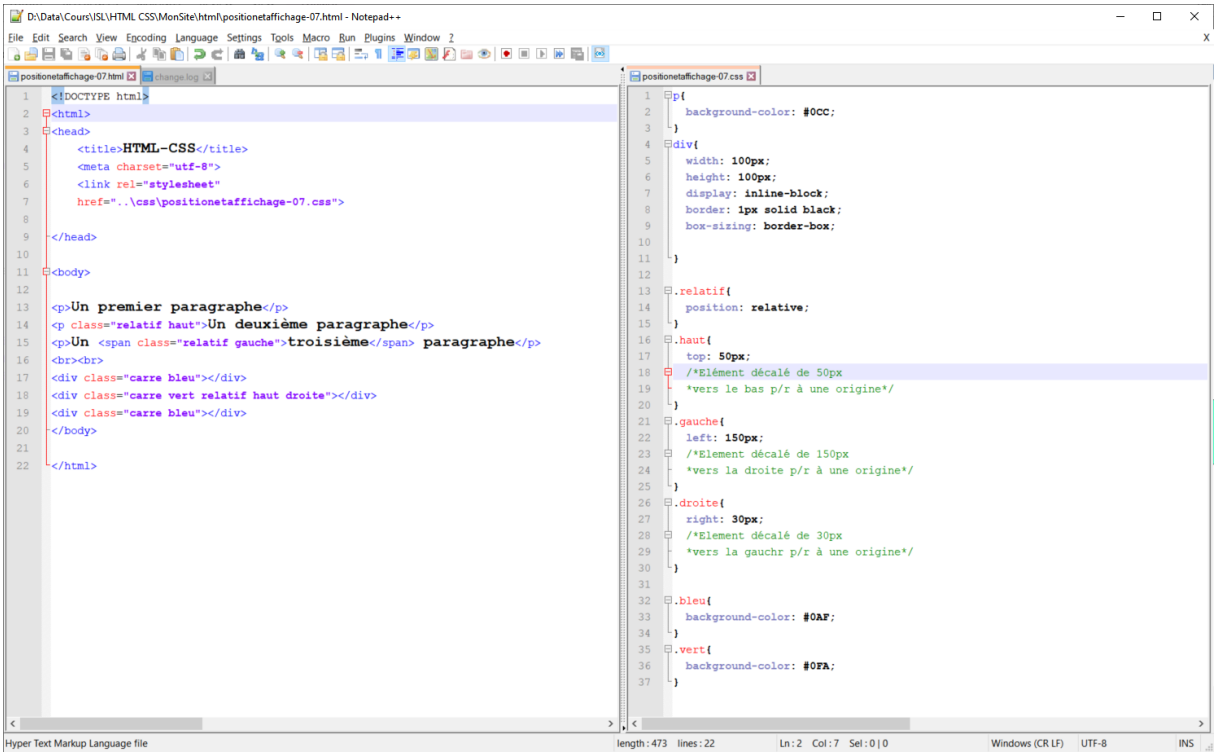
Attribuer une position : relative à un élément va positionner l'élément dans le flux normal de la page tout comme la valeur static de la propriété position : static.

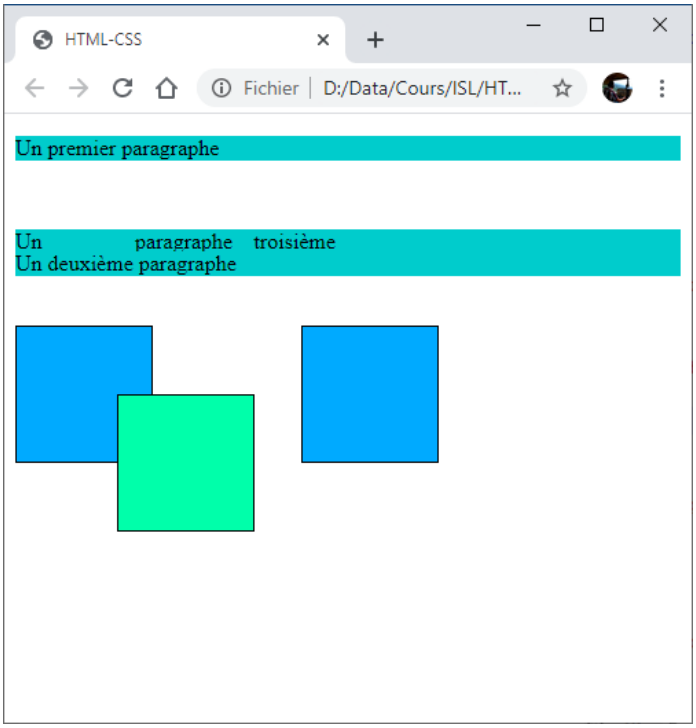
Cependant, à l'inverse d'un élément HTML positionné avec position : static, un élément positionné avec position : relative va ensuite pouvoir être décalé par rapport à sa position initiale grâce aux propriétés top, left, bottom et right.

Ces propriétés vont prendre comme origine la position initiale de l'élément. Nous allons ainsi pouvoir positionner un élément relativement à sa position de départ.

Notez qu'ici l'espace occupé initialement par l'élément va continuer à lui appartenir : les autres éléments ne seront pas affectés par le décalage de notre élément et ne vont pas se repositionner en fonction de celui-ci.

Cela implique également que l'élément décalé va pouvoir être à cheval par-dessus d'autres éléments puisque la position de ces autres éléments ne va pas changer en fonction de l'élément décalé possédant une position : relative





Position : absolute

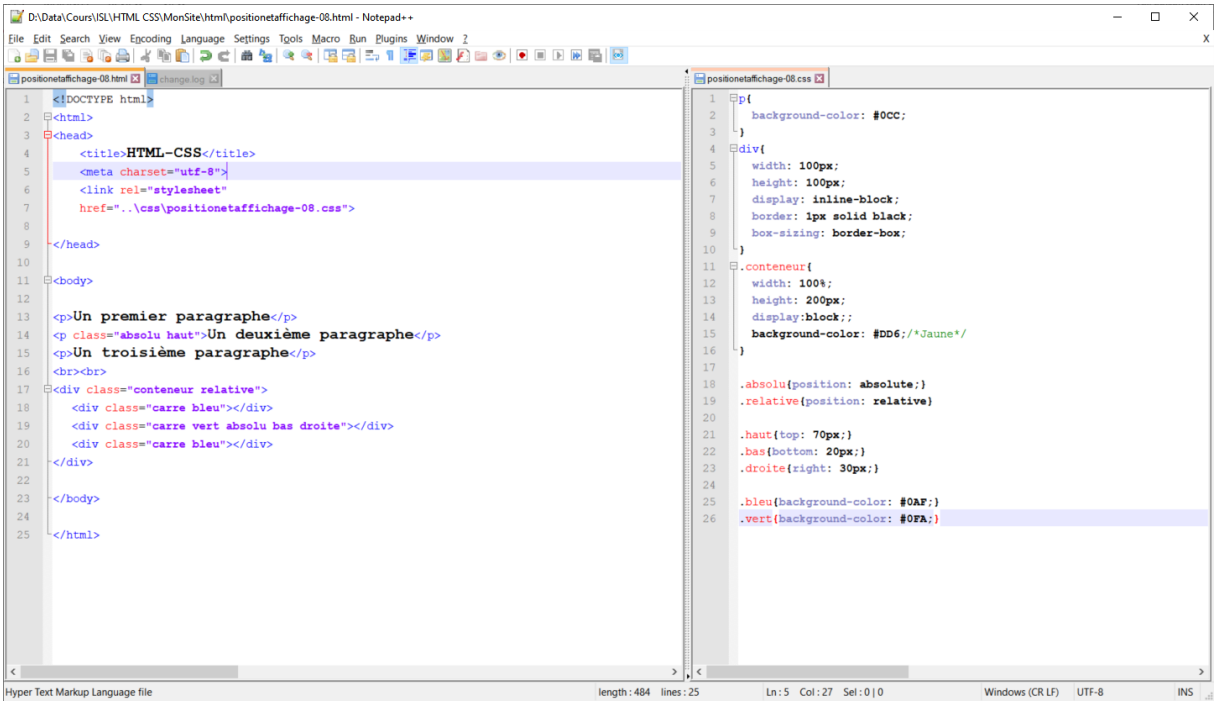
Un élément positionné avec position: absolute va être positionné par rapport à son parent le plus proche positionné (avec une valeur de position différente de static).

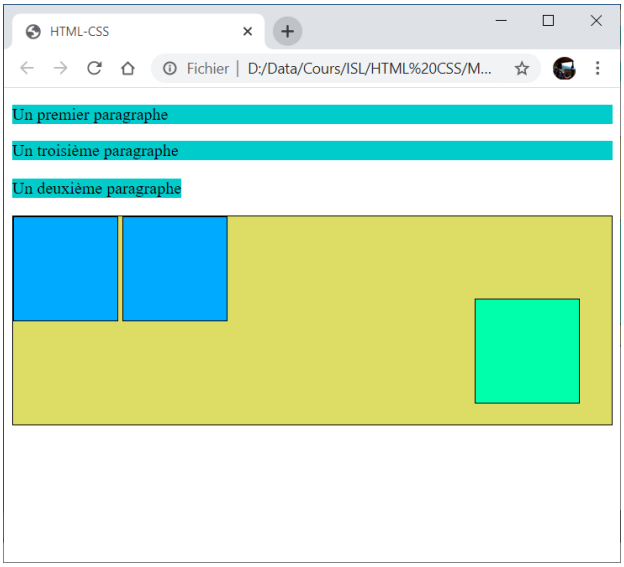
Si aucun parent positionné n'est trouvé, alors l'élément sera positionné par rapport à l'élément racine représentant la page en soi.

Le point de référence pour les propriétés top, left, bottom et right va ainsi être le côté de l'élément parent liée à la propriété (côté gauche pour left, supérieur pour top , etc.).

De plus, un élément positionné avec position : absolute va être retiré du flux normal de la page. Cela signifie et implique que l'espace initialement attribué à un élément au positionnement absolu (espace attribué selon le flux normal de la page) va être occupé par les éléments suivants. Si l'on enlève la classe relative du div conteneur, le carré vert sort du conteneur et revient dans le flux normal de la page.

Un élément positionné avec position: absolute va ainsi pouvoir se placer par-dessus d'autres éléments.





Ici, nous avons un paragraphe et un div positionnés de manière absolue. Notre paragraphe ne possède pas de parent positionné. Il va donc être positionné par rapport à l'élément racine de la page, c'est-à-dire par rapport à la page en soi. Ici, top : 70px signifie donc que notre paragraphe sera placé à 70px du point le plus haut de la page auquel il aurait pu être placé (en tenant compte des marges appliquées).

Notre div possède lui un parent positionné qui est le div jaune « conteneur ». Ce dernier est positionné de manière relative. Notez qu'on lui a déclaré une position : relative mais qu'on n'a pas modifié sa position avec une propriété top, left, etc. Cela n'empêche pas au div conteneur d'être positionné.

Notre div vert va donc être positionné par rapport au div jaune. Ici, bottom : 20px et right : 30px signifie que notre div vert sera positionné à 20px du bord inférieur et à 30px du bord droit de son parent.

Vous pouvez également observer que les éléments positionnés de manière absolue sont bien retirés du flux normal de la page et les autres éléments vont pouvoir venir se positionner à la place de ces éléments, comme s'ils n'existaient pas. On voit bien cela avec notre deuxième div bleu qui vient se coller au premier et vient donc prendre la place du div vert ainsi qu'avec notre troisième paragraphe qui prend la place initiale du deuxième.

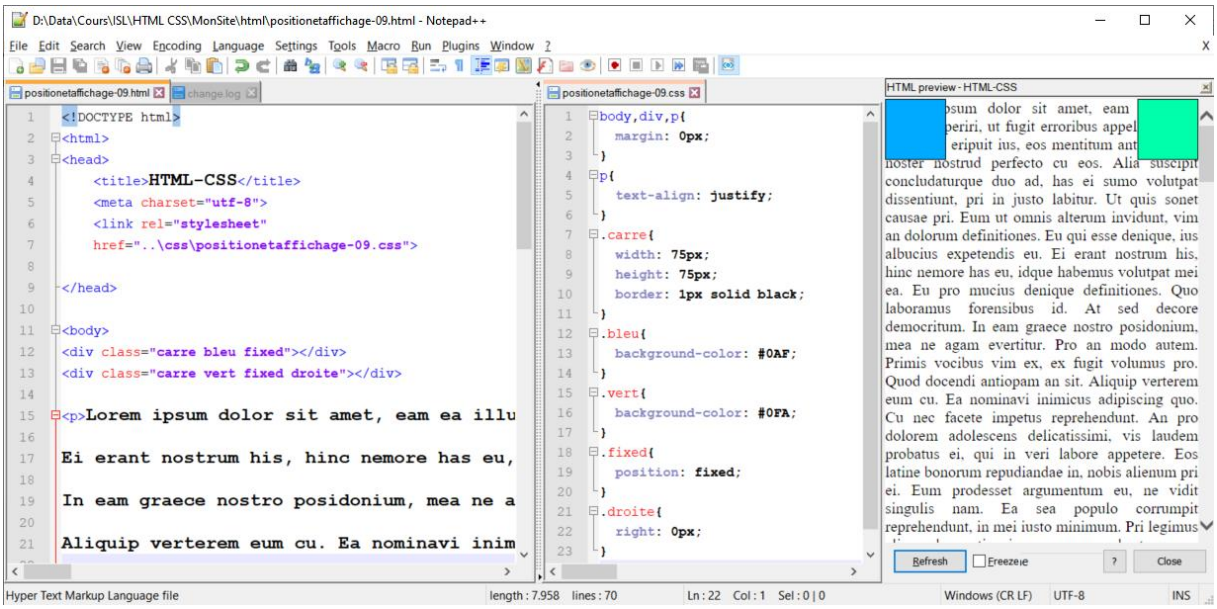
Position : fixed

Le positionnement fixe est très proche du positionnement absolu. Un élément positionné avec position: fixed va également être retiré du flux de la page et l'espace qui lui était attribué selon le flux normal de la page va également pouvoir être utilisé par d'autres éléments.

La seule différence entre position: fixed et position: absolute est que l'élément ne va plus être positionné par rapport à son parent le plus proche mais par rapport au viewport, c'est-à-dire par rapport à la fenêtre visible à moins que l'un de ses parents possède une propriété transform, filter ou perspective dont la valeur est différente de none.

En dehors de ces cas particuliers, un élément positionné avec position: fixed apparaitra toujours à la même place même dans la fenêtre si on descend ou on monte dans la page : il sera fixe par rapport à la fenêtre. En effet, sa position va être calculée par rapport à la fenêtre visible.

A noter ici une exception pour les contenus paginés : dans ce cas-là, l'élément possédant une position: fixed sera répété dans chaque page.



Position : sticky

La dernière valeur de la propriété CSS position est la valeur sticky. Un élément positionné avec position: sticky sera d'abord positionné selon le flux normal de la page puis va pouvoir être décalé de manière similaire à un élément positionné de manière relative. Les éléments suivants ne verront pas leur position changée : ils seront toujours placé « comme si » l'élément positionné avec position: sticky occupait sa place d'origine.

La différence ici entre un élément positionné avec position: sticky et position: relative est que la position d'un élément sticky va être calculée par rapport à son parent possédant un mécanisme de défilement (scrolling) le plus proche.

Ainsi, un élément positionné avec position: sticky va avoir une position relative au départ puis son positionnement va devenir fixe dès qu'un certain point sera franchi, c'est-à-dire à partir d'un certain niveau de défilement de la page. Les propriétés top, left, bottom et right vont nous permettre de pouvoir préciser à partir de quel moment l'élément positionné avec position: sticky va devoir être fixe.

Notez que la valeur sticky est une valeur assez récente de la propriété position et est à ce titre toujours en développement. On évitera donc de l'utiliser pour le moment sur un site « live ».

Un mot sur l'accessibilité du contenu

Lorsqu'on code, il faut toujours s'efforcer de réfléchir en termes d'accessibilité à tous, et notamment pour les personnes souffrant de déficiences comme des déficiences visuelles.

En effet, un des principes de base du web est d'être accessible à tous ou du moins c'est l'une des valeurs fondamentales vers laquelle tendre.

Ici, il faudra donc faire bien attention à ce que les contenus positionnés ne cachent pas d'autres contenus de façon non désirée lorsqu'un utilisateur par exemple zoome sur la page pour augmenter la taille du texte.

Définir l'ordre d'affichage des éléments en cas de chevauchement avec la propriété z-index

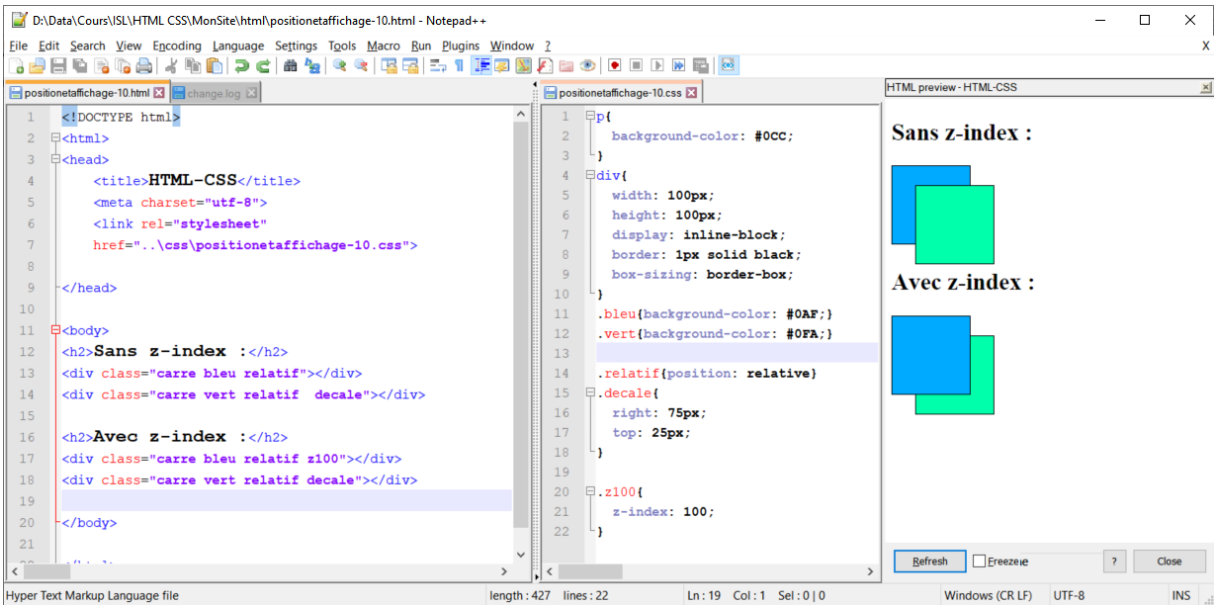
Les éléments HTML vont pouvoir être positionné dans une page en CSS selon 3 dimensions : selon la largeur (axe horizontal ou axe des X en math), la hauteur (axe vertical ou axe des Y) et également selon une épaisseur ou un ordre d'empilement (axe des Z).

En effet, vous avez pu remarquer dans les exemples précédents que lorsque deux éléments se chevauchaient, il y en avait toujours un au-dessus de l'autre : il y a donc une notion d'ordre d'empilement selon cet axe 3D qui est l'axe des Z.

Par défaut, lorsque deux boites se chevauchent, l'élément déclaré en dernier apparaitra par-dessus l'élément déclaré avant en HTML. C'est une règle implicite de tout document HTML.

La propriété z-index va nous permettre de modifier ce comportement et de choisir quel élément doit apparaitre au-dessus de quel ordre en donnant un index sous forme de nombre à un ou plusieurs éléments. Ainsi, lorsque deux éléments se chevauchent, celui possédant la plus grande valeur pour son z-index apparaitra au-dessus de l'autre.

Notez que la propriété CSS z-index ne va fonctionner (et n'a de sens) qu'avec des éléments HTML positionnés, c'est-à-dire qu'avec des éléments possédant une propriété position dont la valeur est différente de static en CSS.



La propriété CSS float

La propriété CSS float permet de sortir un élément du flux normal de la page et de le faire “flotter” contre un bord de son élément parent conteneur ou contre un autre élément flottant.

Une utilisation bien connue de la propriété float est de s’en servir pour faire flotter une image à droite ou à gauche d’un texte et ainsi l’entourer avec du texte.

La propriété float est donc une autre propriété qui va impacter la disposition dans la page et qu’il convient de manier avec précaution pour ne pas obtenir de comportement indésirable. Le but de cette nouvelle leçon est d’apprendre à la manipuler.

Définition et fonctionnement de la propriété float

La propriété float va retirer un élément du flux normal de la page puis le placer contre un bord de son élément parent (ou élément conteneur) ou contre le bord d’un élément flottant le précédant.

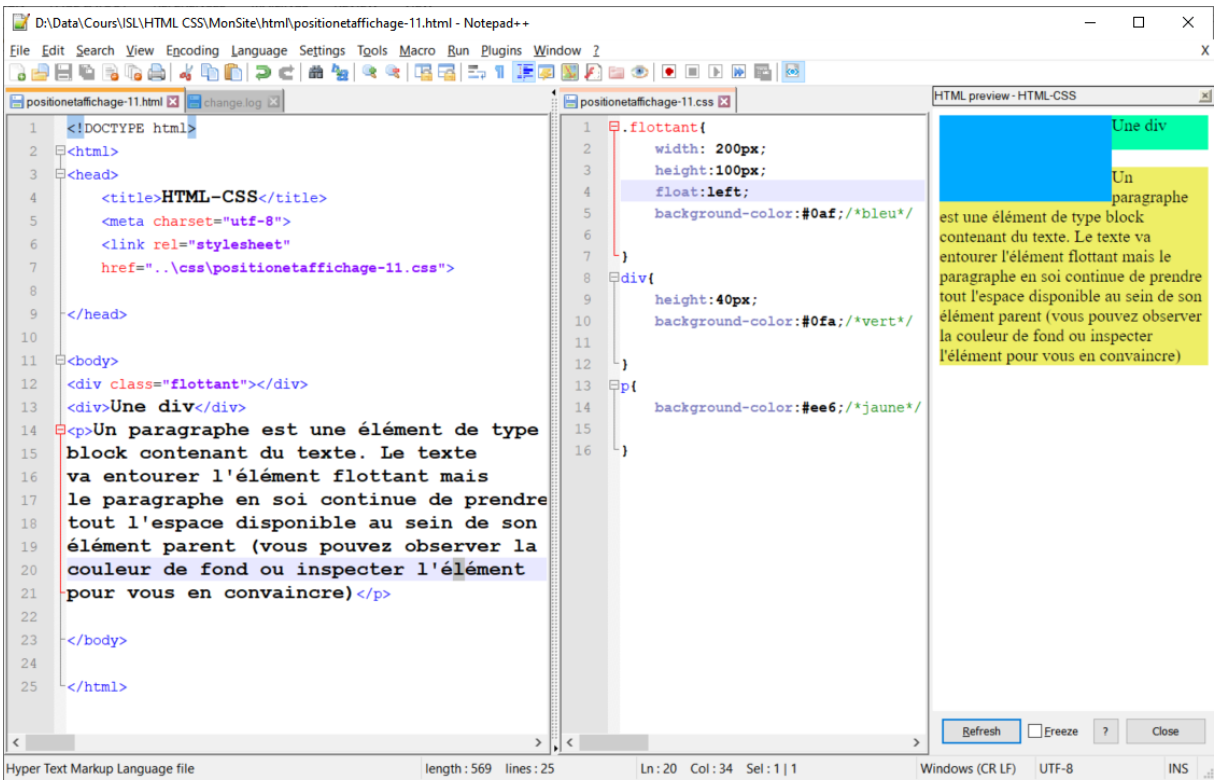
Cette propriété va également impacter les éléments environnants puisque le texte et les éléments inline suivants un élément possédant un float différent de none vont essayer de venir se placer à ses côtés.

La propriété float était à l’origine principalement utilisée pour incruster des images dans du texte en les faisant flotter. L’utilisation « normale » de float est donc d’appliquer le float (de faire flotter) des éléments inline dans la page et de laisser les textes se positionner autour.

En effet, les éléments de type block suivants un élément flottant vont également se placer sur la même ligne que l’élément flottant mais continuer à prendre tout l’espace disponible dans la ligne. La partie des éléments block chevauchant un flottant va être cachée derrière le flottant.

On va cependant également tout à fait pouvoir faire flotter un élément block même si généralement nous utiliserons plutôt un display : inline-block pour placer deux éléments de type block côte-à-côte.

Notez que si vous voulez faire flotter un élément block sans contenu, alors il faudra lui donner une largeur et une hauteur explicites avec les propriétés width et height car dans le cas contraire les valeurs calculées seront égales à 0.



Notez également déjà que la propriété float ne va pas fonctionner avec des éléments positionnés de manière absolue et ne va avoir aucun effet sur des éléments affichés avec display : flex ou display : inline-flex. Nous aurons l'occasion de revenir sur ces sujets plus tard.

Les valeurs de la propriété float

Historiquement, nous avions le choix entre 3 valeurs à passer à la propriété float :

- float : left : L'élément va venir se positionner à l'extrémité gauche de son élément conteneur ou va être décalé sur la gauche jusqu'à toucher un autre élément avec float : left ;
- float : right : L'élément va venir se positionner à l'extrémité droite de son élément conteneur ou va être décalé sur la droite jusqu'à toucher un autre élément avec float : right ;
- float : none : Valeur par défaut. L'élément n'est pas un élément flottant.

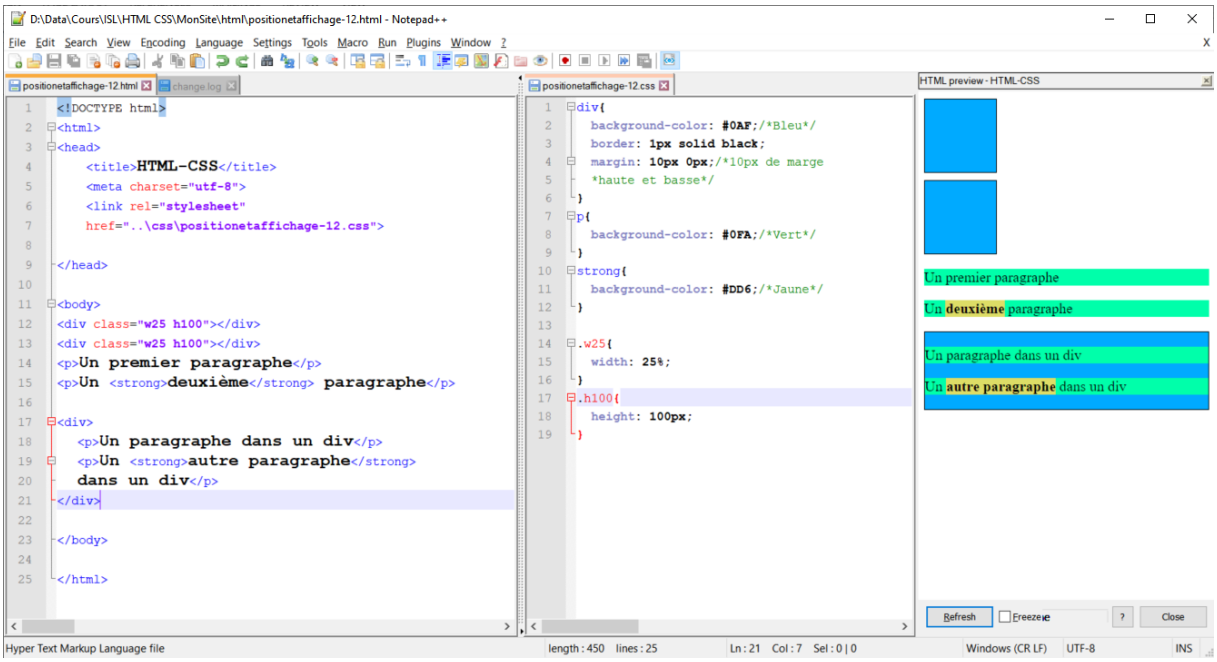
Le CSS3 va apporter un nouveau choix élargi de valeurs que nous allons pouvoir passer à float. Je vous rappelle ici que le CSS3 est toujours en développement et qu'ainsi tout ce qui est en train d'être établi par celui-ci n'est pas forcément encore passé comme recommandation du W3C. En effet, certaines valeurs et propriétés actuellement à l'étude dans le cadre du CSS3 vont potentiellement être modifiées ou abandonnées en cours de route.

Parmi les nouvelles valeurs apportées à float, cependant, nous pouvons déjà en citer deux qui possèdent déjà un bon support par les navigateurs :

- float : inline-start : L'élément va venir se positionner au début de son élément conteneur (c'est-à-dire à gauche pour des documents dont l'écriture se fait de gauche à droite ou à droite dans le cas contraire) ou va être décalé vers le début de son conteneur jusqu'à toucher un autre élément avec float : inline-start ;
- float : inline-end : L'élément va venir se positionner à la fin de son élément conteneur (c'est-à-dire à droite pour des documents dont l'écriture se fait de gauche à droite ou à gauche dans le cas contraire) ou va être décalé vers la fin de son conteneur jusqu'à toucher un autre élément avec float : inline-end.

Float : none

La valeur par défaut de float est none. Cette valeur correspond à l'absence de flottement.



Ici, nous avons deux éléments div de largeurs égales à 25% de celle de leur parent et de hauteurs égales à 50px. Les div sont des éléments de type block et vont donc aller à la ligne et occuper une ligne chacun quelle que soit leurs dimensions.

Ensuite, nous avons deux éléments p dont un qui contient un élément strong. Les éléments p sont également de type block et vont donc par défaut occuper tout l'espace disponible dans leur parent et occuper une ligne chacun. L'élément strong est lui un élément inline par défaut et va donc ne prendre que la place nécessaire dans son parent et ne pas aller à la ligne.

Finalement, nous avons créé un div qui contient deux paragraphes dont un contient lui-même un autre élément strong.

Par défaut, aucun de ces éléments ne flotte. Ne rien préciser ou préciser un float : none est ici identique et le comportement de ces éléments est connu.

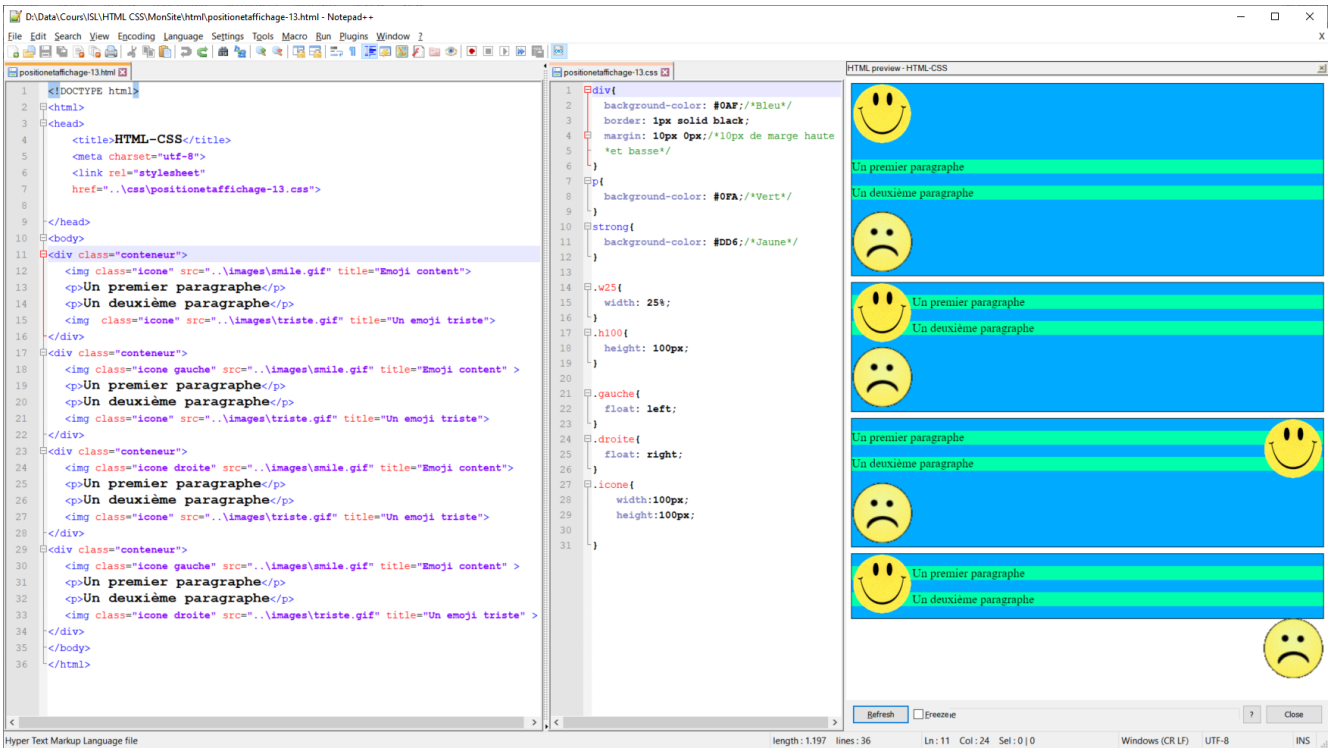
Note : parfois, il sera utile de préciser un float : none pour un élément pour annuler par exemple un comportement d'héritage.

Float : left et float : right

En appliquant un float: left à un élément, l'élément va venir se positionner contre le bord gauche de son élément conteneur ou va être décalé vers la gauche jusqu'à toucher un autre élément flottant.

En appliquant un float: right à un élément, l'élément va venir se positionner contre le bord droit de son élément conteneur ou va être décalé vers la droite jusqu'à toucher un autre élément flottant.

Ces deux valeurs vont se comporter et pouvoir être appliquées de manière similaire.



Il y a des choses intéressantes à noter dans cet exemple. Ici, on va faire flotter nos différentes images d'emoji à droite ou à gauche. La première chose à retenir est qu'un élément flottant va toujours flotter sur sa ligne. Dans le dernier exemple, par exemple, notre premier emoji est le premier élément déclaré dans notre conteneur, il flottera donc en haut tandis que notre deuxième emoji est le dernier élément déclaré et il flottera donc en bas du conteneur.

Dans notre dernier div, en particulier, l'emoji « triste » se retrouve en dehors du div. Cela est dû au fait que float retire l'élément du flux normal de la page : le div ne tiendra plus compte de cet élément et va donc se redimensionner à la taille des éléments qu'il contient, c'est-à-dire dans ce cas les deux paragraphes. L'élément flotté va lui en revanche continuer à flotter sur sa ligne.

Si vous regardez de plus près, vous pouvez constater qu'il se passe exactement la même chose avec notre premier emoji mais c'est moins visible car c'est le premier élément déclaré dans notre div et donc il va toujours être inclus dedans grâce aux paragraphes le suivant.

Cependant, ne vous y trompez pas : le div ne tient plus compte de l'emoji flotté dans le calcul de sa taille mais seulement des éléments non flottés et va se redimensionner en conséquence. C'est la raison pour laquelle nos deuxième et troisième div sont plus petits en hauteur que le premier. Pour éviter ce genre de rendu visuel, on peut toujours augmenter artificiellement la taille de notre élément conteneur en utilisant height.

La deuxième chose à bien comprendre est le fait qu'un élément flottant n'impacte pas les propriétés des éléments environnants à proprement parler : les éléments suivants un élément flottant vont pouvoir se placer à côté de l'élément flottant dans la limite de la hauteur de l'élément flottant. Les éléments sous l'élément flottant vont avoir un comportement « normal » comme on peut le voir avec notre deuxième paragraphe pour nos deuxième et troisième div.

Contrôler le comportement des éléments autour d'un flottant avec la propriété clear

La propriété CSS clear va nous permettre d'empêcher un élément de se positionner à côté d'un élément flottant. Cette propriété va être extrêmement utile dans de nombreuses situations pour mieux contrôler le design de nos pages.

Nous allons pouvoir lui passer l'une des valeurs suivantes :

- clear : none : Valeur par défaut. Laisse les éléments se positionner à côté d'éléments flottants ;
- clear : left : Empêche un élément de se positionner à côté d'éléments possédant un float : left ;
- clear : right : Empêche un élément de se positionner à côté d'éléments possédant un float : right ;
- clear : both : Empêche un élément de se positionner à côté d'éléments possédant un float : left ou un float : right ;
- clear : inline-start : Empêche un élément de se positionner à côté d'éléments possédant un float : inline-start ;
- clear : inline-end : Empêche un élément de se positionner à côté d'éléments possédant un float : inline-end.

Notez ici que la propriété clear va avoir un comportement légèrement différent selon qu'on l'applique à des éléments non flottants ou au contraire à des éléments déjà flottants.

Dans le cas où l'on applique clear à un élément non flottant, l'élément va être déplacé de telle sorte à ce que sa bordure supérieure se place directement sous le bord de la marge basse extérieure des éléments flottants concernés et il va y avoir fusion des marges (collapse en anglais)

Dans le cas où clear est appliquée à un élément flottant, l'élément va être déplacé de telle sorte à ce que l'extrémité de sa marge supérieure se place directement sous le bord de la marge basse des éléments flottants concernés. Les deux marges vont donc être conservées. La position des potentiels

éléments flottants suivants va être impactée puisqu'un élément flottant ne peut pas être situé plus haut qu'un autre élément flottant qui le précède.

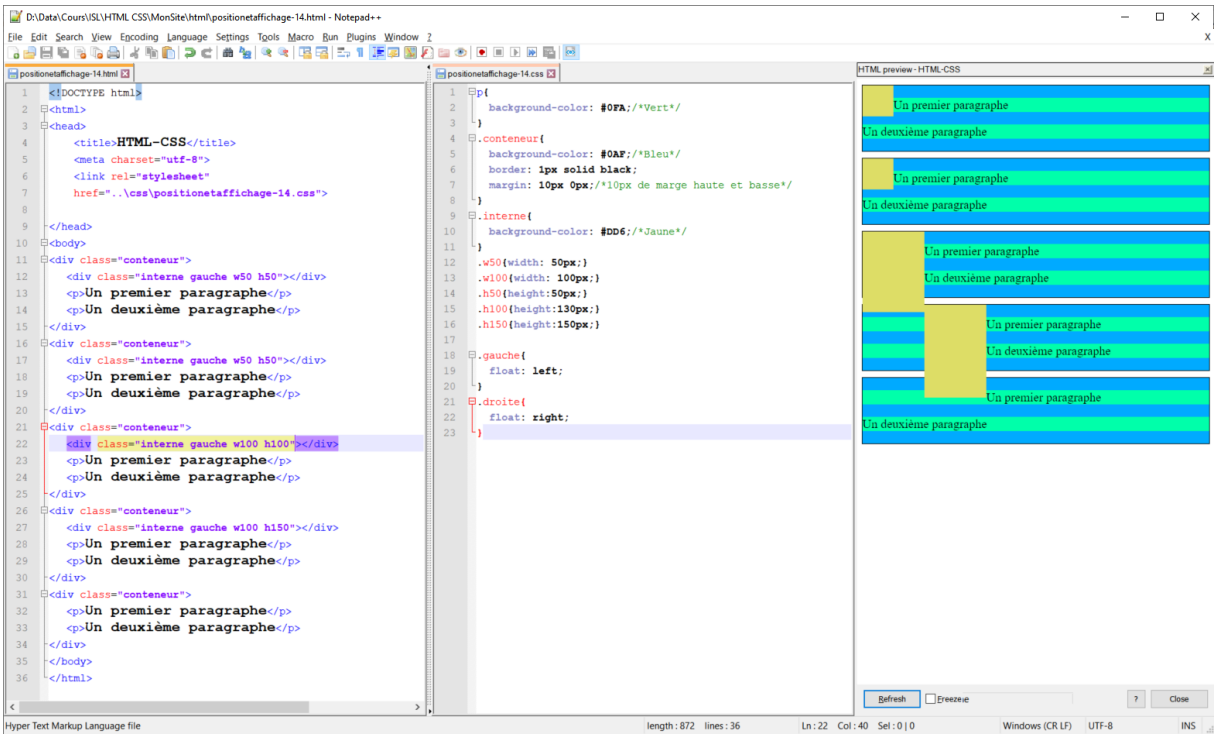
Cas pratiques d'utilisation de la propriété clear

Généralement, nous allons donc utiliser la propriété clear après avoir appliqué un float à un élément pour empêcher certains éléments de venir flotter à côté de l'élément en question.

En effet, bien souvent, lorsque nous créerons le design d'une page, nous n'allons pas vouloir que les éléments se positionnent tant qu'ils le peuvent à côté d'un élément flottant mais pouvoir contrôler quels éléments vont pouvoir se positionner aux côtés d'un élément flottant et quels éléments ne doivent pas le faire.

Illustration du problème réglé par la propriété clear

Pour bien comprendre l'intérêt de la propriété float, je vous propose de regarder l'exemple suivant :



Ici, nous faisons flotter chacun des div internes à gauche dans nos div conteneurs. Pour nos deux premiers div, le résultat est celui espéré. En revanche, on observe un décalage qui se crée pour nos trois derniers div.

En fait, c'est tout à fait normal : le fait que l'affichage se fasse bien avec nos deux premiers div est simplement un « coup de chance » dû au fait que nous n'avons pas deux éléments flottants qui se touchent puisque les div conteneurs soient plus grands que les div flottés et qu'ils ne contiennent qu'un élément flottant.

Ici, il faut savoir que par défaut tous les éléments suivants un flottant vont essayer de se positionner sur la même ligne que l'élément flottant et cela dans la limite de la hauteur de l'élément flottant.

Dans nos deux premiers exemples, le deuxième paragraphe va à la ligne tout simplement car il n'a pas la place pour se positionner sur la même ligne que le flottant (la hauteur du flottant est déjà remplie par le premier paragraphe).

Pour les div flottants suivants, en revanche, la situation va être différente. Notre troisième div flottant dépasse en effet de son div conteneur et va arriver jusqu'à la ligne sur laquelle se situe le div flottant suivant.

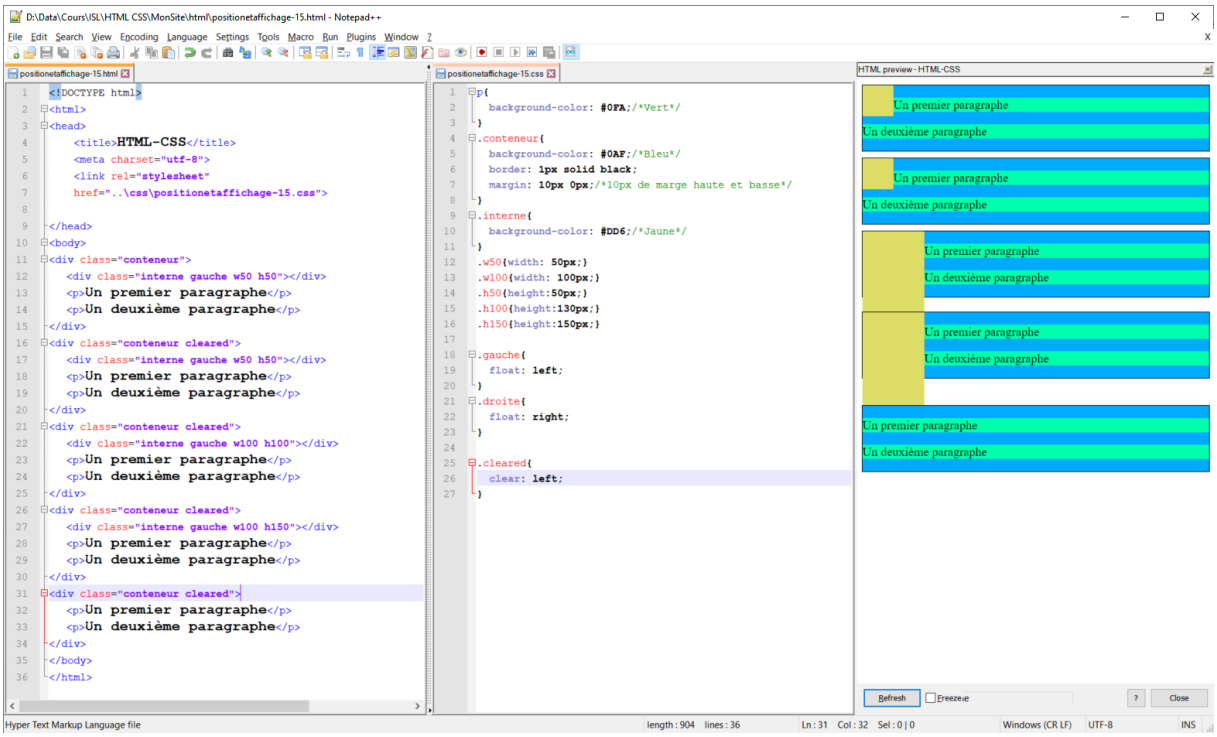
Le quatrième div flottant va donc toucher le div flottant précédent. Or, un élément flottant va essayer se positionner soit contre le bord de son élément parent soit contre le bord d'un élément flottant précédent si un tel élément existe.

C'est ce qui se passe ici : notre dernier div flotté va rencontrer le div flotté précédent et va donc se coller contre son bord.

Enfin, notre quatrième et dernier div flottant est suffisamment haut pour arriver jusqu'aux lignes occupées par les paragraphes du dernier div conteneur. Le texte des paragraphes va donc se positionner contre le flottant du div conteneur précédent.

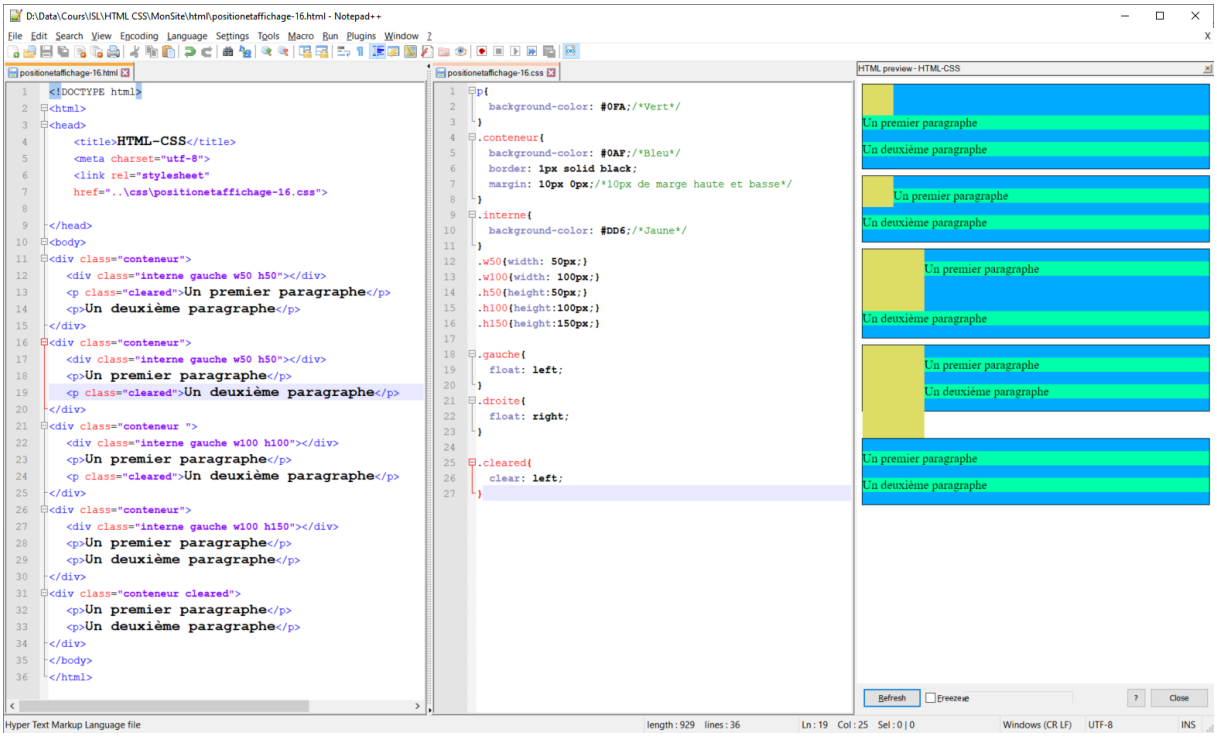
La propriété clear va justement nous permettre d'éviter ce genre de comportements généralement non souhaités.

Ici, par exemple, il suffirait d'appliquer un clear : left à chacun de nos div conteneurs pour résoudre une grande partie du problème :



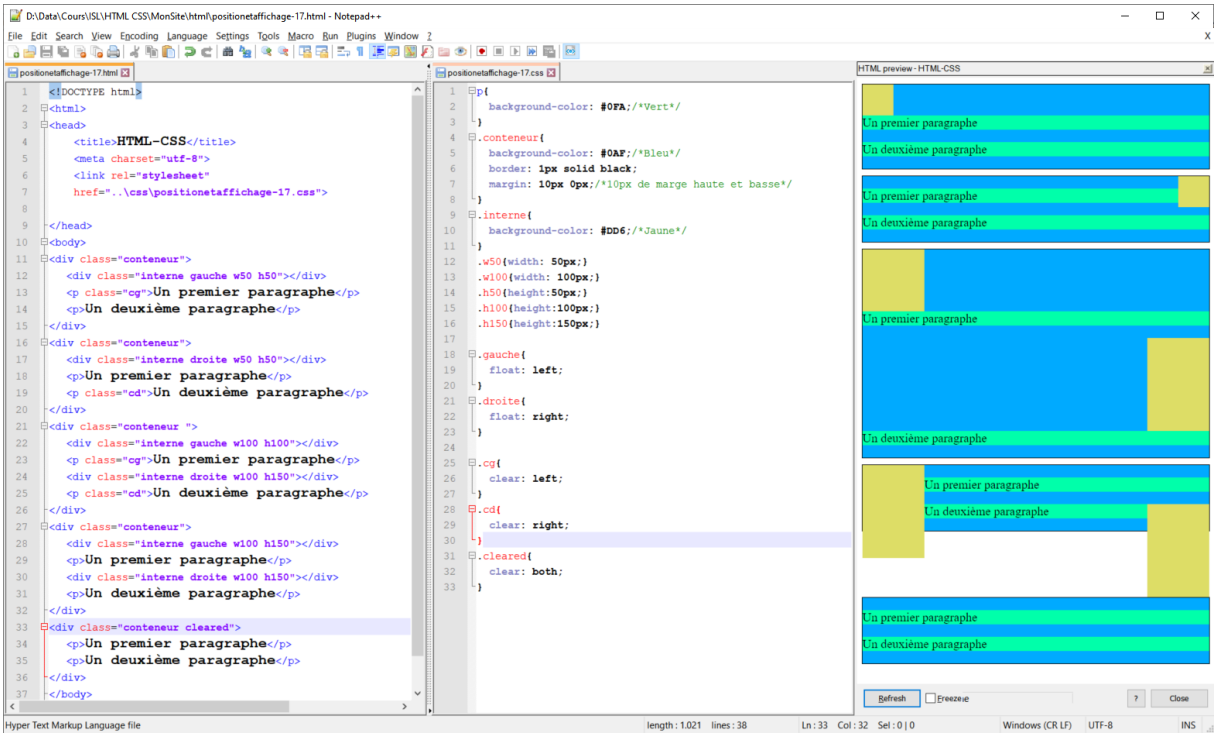
La propriété clear empêche ici nos div conteneurs (et donc les éléments qu'ils contiennent) de venir se positionner à côté des éléments flottants précédents. Les div conteneurs vont donc se positionner juste en dessous des flottants si les éléments flottants précédents dépassaient de leur conteneur.

Notez qu'on aurait ici pu également tout-à-fiat appliquer le clear sur un autre élément comme par exemple sur l'un des paragraphes de nos différents div. Dans ce cas, le paragraphe en question et tous les éléments suivants auraient été libérés du flottement et se seraient positionnés sous le flottant.



Bien évidemment, la propriété clear va fonctionner exactement de la même façon avec sa valeur right pour empêcher des éléments de se positionner à côté d'un élément possédant un float : right.

La valeur both va elle servir à libérer les éléments d'un flottant gauche ou d'un flottant droit.
Regardez plutôt l'exemple ci-dessous que vous devriez être capable de comprendre par vous-même :

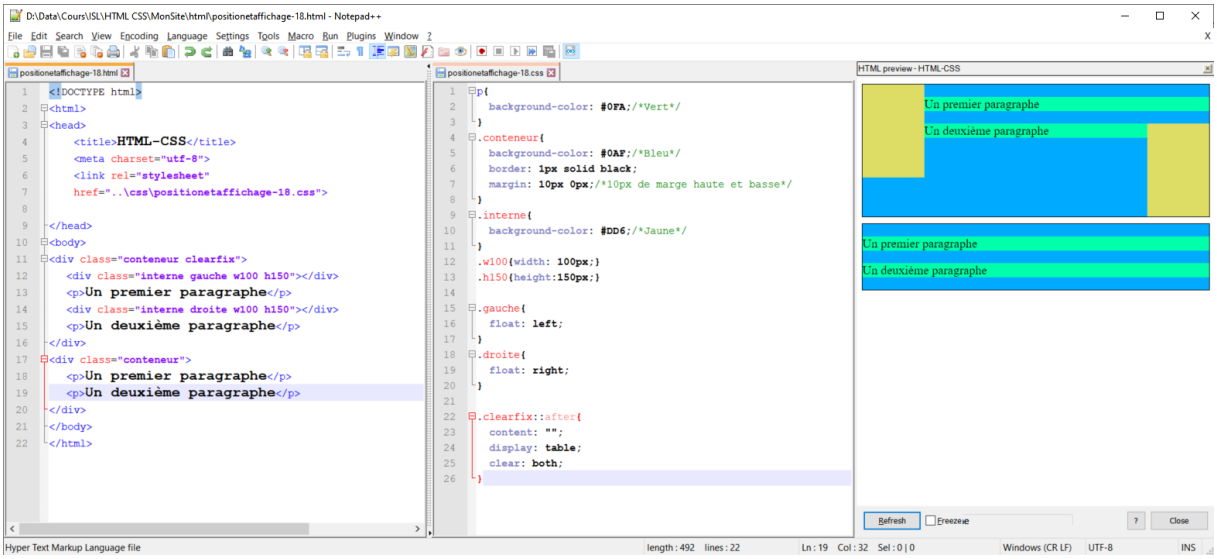


Découverte et utilisation du « clearfix » CSS

Il nous reste un problème à régler : comment faire lorsque le flottant est le dernier élément dans son conteneur pour ne pas que celui-ci dépasse du conteneur ?

En effet, bien souvent, on voudra que les éléments flottants restent dans la limite de leur conteneur ou plus exactement que les conteneurs s'adaptent pour inclure les flottants dans leur taille.

Pour faire cela, on va utiliser ce qu'on appelle un « clearfix » qui est un hack bien connu en CSS utilisant le pseudo-élément ::after. Attention : il faudra cette fois-ci l'appliquer au conteneur qui contient le flottant qui pose des problèmes de design pour qu'il fonctionne. Ce clearfix est le suivant :



Nous étudierons les pseudo-éléments plus tard dans ce cours. Cependant, je vais quand même essayer de vous expliquer brièvement comment fonctionne le clearfix. Le pseudo élément ::after nous permet de créer un pseudo élément qui va être le dernier enfant de l'élément sélectionné.

L'idée est ici de l'utiliser avec la propriété content pour ajouter un contenu ou plus exactement une chaine de caractères vide en fin de l'élément.

Ensuite, nous allons appliquer le clear à ce pseudo élément. Pour que cela fonctionne, cependant, il va falloir lui définir un display. Le type d'affichage qui se prête le mieux à l'opération est ici display : table

La propriété float et la hauteur des conteneurs

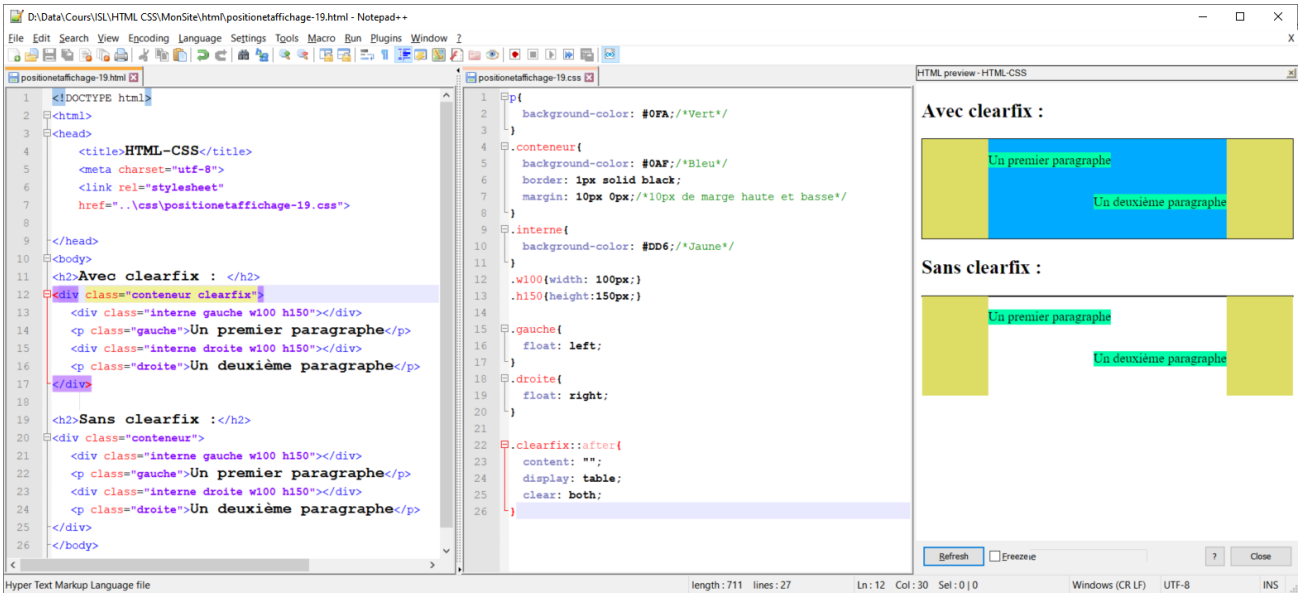
Comme nous avons pu le voir et l'évoquer plus haut, un élément flottant est retiré du flux normal de la page.

Cela implique que l'élément qui contient le flottant ne va pas tenir compte de ce dernier dans le calcul de ses dimensions.

Cela peut donc mener à des problèmes de dépassement du flottant de son élément conteneur comme on a pu le voir précédemment.

Il y a un cas que nous n'avons cependant pas encore étudié : le cas où le conteneur ne contient que des éléments flottants. Dans ce cas-là, la hauteur du conteneur va être égale à la taille de ses bordures et de son padding et donc nulle si le conteneur ne possède ni bordures ni marges internes.

Ce comportement sera rarement souhaité. Pour rétablir la hauteur du conteneur, il suffit une nouvelle fois d'utiliser le clearfix vu dans le chapitre précédent.


D:\Data\Cours\ISL\HTML CSS\MonSite\html\positionetaffichage-19.html - Notepad++
positionetaffichage-19.html
positionetaffichage-19.css
HTML preview - HTML-CSS
Avec clearfix :
Un premier paragraphe
Un deuxième paragraphe
Sans clearfix :
Un premier paragraphe
Un deuxième paragraphe
Refresh
Freezeve
Close

Gestion des conflits entre display, position et float

En fonction des valeurs passées, la définition d'une propriété display, float ou position peut créer des conflits ou être incompatible avec chacune des deux autres si elles ont été également définies.

En effet, les propriétés display, position et float servent toutes les trois à modifier la disposition des éléments dans la page et vont pour cela modifier le flux normal de la page.

Dans ces cas-là le CSS définira de nouvelles valeurs qui ne poseront pas de conflit pour l'une ou l'autre des propriétés. C'est ce qu'on appelle une « valeur calculée » par opposition à la « valeur spécifiée ».

Il va être très important de bien connaitre ces situations pour pouvoir prévoir le comportement de chaque propriété et pour pouvoir les définir correctement les unes par rapport aux autres et ainsi obtenir in-fine le design de page souhaité.

Gestion des conflits et valeurs calculées

Résumons ici ce qu'il se passe lorsqu'on utilise les propriétés display, position et float sur un même ensemble d'éléments en fonction des valeurs données.

Nous ne parlerons pas ici des valeurs des propriétés toujours en développement et ne faisant pas encore partie des recommandations officielles car leur impact n'est pas encore bien défini et car leur définition pourrait changer ou qu'elles pourraient être tout simplement abandonnées.

Voici les règles qui vont s'appliquer dans leur ordre d'application :

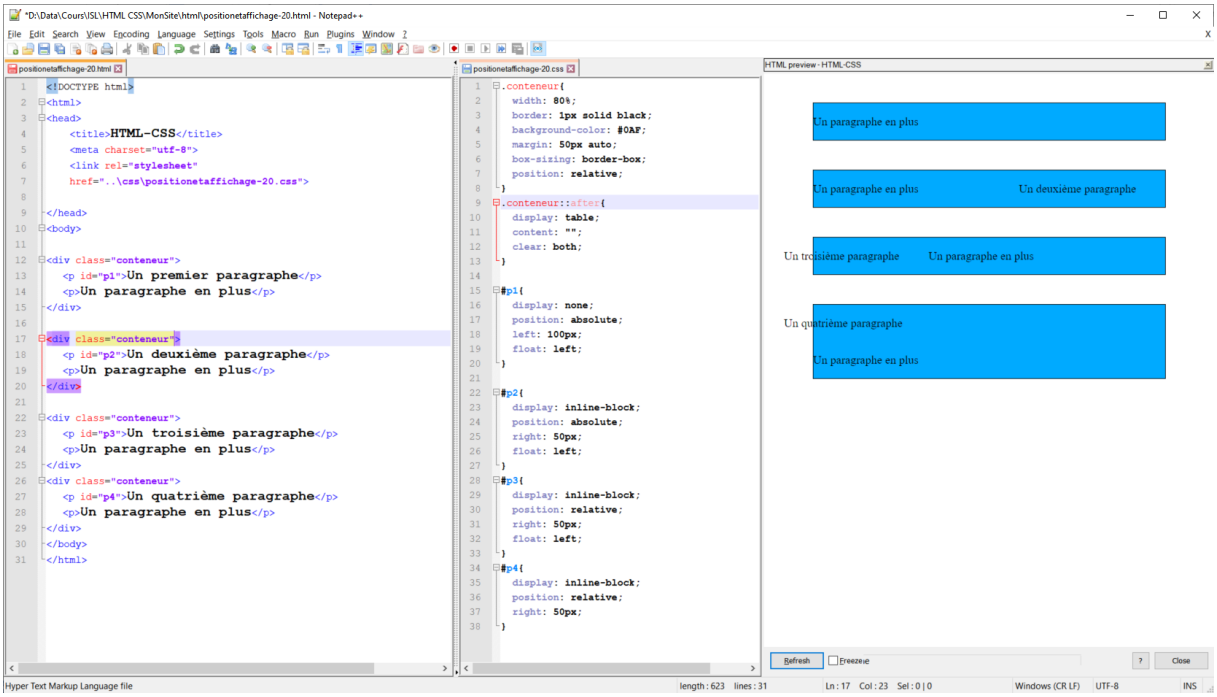
1. Si un élément possède un display : none, alors les propriétés float et position ne s'appliqueront évidemment pas ;
2. Si un élément est positionné avec position : absolute ou position : fixed, alors la propriété float ne s'appliquera pas (sa valeur calculée sera none et la valeur calculée du display sera celle du tableau ci-dessous ;
3. Si un élément N'est PAS positionné avec position : absolute ou position : fixed, la propriété float s'appliquera bien avec la valeur spécifiée. La valeur de display sera calculée selon le tableau ci-dessous ;
4. Si un élément N'est PAS positionné avec position : absolute ou position : fixed, qu'on NE lui applique PAS de float et que c'est l'élément racine du document, alors la valeur de display appliquée sera la valeur calculée précisée dans le tableau ci-dessous ;
5. Pour tout autre élément NON positionné avec position : absolute ou position : fixed et auquel on N'applique PAS de float, le valeur de display appliquée sera la même que celle spécifiée.

Valeur spécifiée	Valeur calculée / appliquée
inline-table	table
inline, table-row-group, table-column, table-column-group, table-header-group, table-footer-group, table-row, table-cell, table-caption, inline-block	block
autre	identique à la valeur spécifiée

Illustration des règles de calcul des valeurs en cas de conflit

Les exemples suivants illustrent quelques situations conflictuelles entre les propriétés display, position et float.

Je vous laisse vous référer aux règles précédemment citées pour comprendre les valeurs appliquées !



Ici, notre premier paragraphe p id="p1" possède un display : none. Les propriétés position (et top, left, etc.) et float vont donc être ignorées.

Le paragraphe p id="p2" possède une position : absolute. La valeur calculée de sa propriété float va donc être none et la valeur calculée du display va être block,

Notre paragraphe p id="p3" est positionné avec position : relative et son display n'est pas none. La propriété float va donc ici bien s'appliquer avec la propriété right. L'élément va donc flotter à gauche et être décalé de 50px vers la gauche par rapport à sa position d'origine, ce qui va le faire sortir de son conteneur. Comme l'élément flotte, la valeur calculée de son display va être block.

Finalement, notre paragraphe p id="p4" est positionné avec position : relative et ne possède pas de float. La valeur spécifiée du display va donc bien être appliquée.

Une astuce : en cas de doute sur les valeurs des propriétés appliquées à vos éléments, vous pouvez les inspecter (en faisant clic droit sur la page après avoir activé les outils pour développeur de votre navigateur, ou via la touche F12) et aller voir les valeurs calculées (« computed » en anglais). Attention cependant aux inconsistances possibles entre les différents navigateurs.