

Notions en HTML/CSS

ELAN

202 avenue de Colmar - 67100 STRASBOURG

☎ 03 88 30 78 30 📧 elan@elan-formation.fr

www.elan-formation.fr

SAS ELAN au capital de 37 000 € -

RCS Strasbourg B 390758241 – SIRET 39075824100041 – Code APE : 8559A

N° déclaration DRTEFP 42670182967 - Cet enregistrement ne vaut pas agrément de l'Etat

SOMMAIRE

I.	Introduction	4
II.	Les fondations d'une page HTML5	4
1.	Signification des balises	5
a.	Le doctype	5
b.	La balise <html>	5
c.	L'en-tête <head>	5
d.	Le corps <body>	5
e.	L'encodage (charset)	5
f.	Le titre principal de la page	6
2.	Les commentaires	6
III.	Un mot sur la sémantique et la relation parent/enfant	7
1.	La sémantique HTML	7
2.	La relation parent/enfant	12
IV.	Structure d'une page HTML5 et balises courantes	14
V.	Introduction au CSS	15
1.	Mise en place	15
2.	Appliquer un style : class et id	16
VI.	La propriété CSS display	17
1.	Le concept de "boite"	17
2.	Un mot sur la propriété box-sizing	18
3.	L'enchaînement des boites à l'écran	19
a.	Valeurs de base de display	20
b.	Les valeurs de display spécifiques	21
VII.	Les boites flottantes	22
VIII.	L'overflow	25
IX.	Positionnement avancé et z-index	27
1.	La propriété position	27
2.	z-index	28
3.	Tableau récapitulatif	29
X.	Les sélecteurs avancés en CSS	30
1.	Les combineurs/sélecteurs de voisins	31
a.	Sélecteur de descendant	31
b.	Sélecteur de voisin direct	31
c.	Sélecteur de voisins	31
d.	Sélecteur d'élément fils	31
2.	Sélecteur d'attributs	31
3.	Pseudo-classes	32
4.	Pseudo-éléments	34
5.	Exemple récapitulatif	35

XI. flexbox et grid/subgrid.....	37
1. flex/inline-flex.....	37
2. Grid/subgrid.....	40
XII. Pour aller plus loin	40



I. Introduction

Il existe mille et une façon de modéliser une même page, mais les standards W3C¹ respectés par les navigateurs (plus ou moins strictement) et les bonnes pratiques étant de plus en plus partagées et reconnues par toute la profession, certaines notions deviennent indispensables à tout bon webdesigner !

Et ceci, dans le contexte professionnel, en tenant compte des maquettes de design qu'un graphiste aura conçu, aussi talentueux et conscient de l'ergonomie à apporter à une page web qu'il puisse être, mais dont les connaissances en HTML/CSS sont parfois proches du néant.

Ainsi, étant donné que structurer une page web en HTML consiste à emboîter les balises les unes dans les autres, s'arrêter à cette définition simpliste c'est aller à la catastrophe ! Il faut rigoureusement choisir les bonnes balises (et bien connaître leurs spécificités) pour une bonne hiérarchisation du contenu.

II. Les fondations d'une page HTML5

Le code suivant correspond à la structure de base d'une page web en HTML5 :

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Titre</title>
  </head>
  <body>

  </body>
</html>
```

Il est nécessaire de « décaler » les balises pour rendre le code source plus lisible. Cette mise en forme est appelée indentation. Dans l'éditeur de texte, il suffit d'appuyer sur la touche TABULATION pour obtenir le même résultat.

Les balises s'ouvrent et se ferment dans un ordre précis. Par exemple, la balise <html> est la première que l'on ouvre, et c'est aussi la dernière que l'on ferme (tout à la fin du code, avec </html>). Une balise ainsi ouverte dans un bloc doit être fermée dans ce même bloc.

Exemple : <html><body></body></html> : la syntaxe est correcte. Une balise qui est ouverte à l'intérieur d'une autre doit aussi être fermée à l'intérieur.
<html><body></html></body> : la syntaxe est incorrecte, les balises s'entremêlent.

¹ World Wide Web Consortium, organisation d'entreprises ayant pour objectif de standardiser les langages web (HTML, CSS, JavaScript) pour uniformiser et faciliter le travail des développeurs d'applications en ligne.

1. Signification des balises

a. Le doctype

```
<!DOCTYPE html>
```

La toute première ligne s'appelle le doctype. Elle est indispensable car c'est elle qui indique qu'il s'agit bien d'une page web HTML.

b. La balise <html>

```
<html>  
</html>
```

C'est la balise principale du code. Elle englobe tout le contenu de votre page. La balise fermante </html> se trouve à la fin du code.

c. L'en-tête <head>

Cette section donne quelques informations générales sur la page, comme son titre, l'encodage (pour la gestion des caractères spéciaux), etc. Cette section est généralement assez courte. Les informations que l'en-tête contient ne sont pas affichées sur la page, ce sont simplement des informations générales à destination de l'ordinateur. Elles sont cependant très importantes.

C'est aussi l'endroit où sont déclarées les feuilles de style CSS qui permettront de positionner et décorer les blocs déclarés dans la page HTML. Par exemple, le code suivant permettra de lier la feuille de style « style.css » à la page HTML afin d'appliquer à cette dernière tous les styles CSS désirés.

```
<link rel="stylesheet" href="style.css" />
```

Notez toutefois que cette balise est auto-fermante : il n'y a pas de balise </link> dans le code ci-dessus.

d. Le corps <body>

Le corps <body> : c'est là que se trouve la partie principale de la page. Tout ce qui y est écrit sera affiché à l'écran. C'est à l'intérieur du corps que se trouvera la majeure partie du code.

e. L'encodage (charset)

```
<meta charset="utf-8" />
```

Cette balise indique l'encodage utilisé dans votre fichier .html.

L'encodage indique la façon dont le fichier est enregistré. Il détermine comment les caractères spéciaux s'afficheront (accents, idéogrammes chinois et japonais, symboles arabes, etc.).

Il y a plusieurs techniques d'encodage en fonction des langues : ISO-8859-1, OEM 775, Windows-1253, etc. Un seul cependant devrait être utilisé aujourd'hui autant que possible : UTF-8. Cette méthode d'encodage permet d'afficher sans aucun problème pratiquement tous les symboles de toutes les langues.

Il faut également que votre fichier soit bien enregistré en UTF-8. C'est le cas le plus souvent sous Linux par défaut, mais sous Windows il faut généralement le dire au logiciel. (dans Notepad++ > Encodage > UTF-8 (sans BOM)).

f. Le titre principal de la page

```
<title>Titre</title>
```

C'est le titre de votre page, probablement l'élément le plus important ! Toute page doit avoir un titre qui décrit ce qu'elle contient.

Il est conseillé que le titre soit assez court (moins de 100 caractères en général).

Le titre ne s'affiche pas dans la page mais en haut de celle-ci (souvent dans l'onglet du navigateur).

2. Les commentaires

Un commentaire en HTML est un texte qui sert simplement de mémo. Il n'est pas affiché, il n'est pas lu par l'ordinateur, ça ne change rien à l'affichage de la page.

Un commentaire est une balise HTML avec une forme bien spéciale :

```
<!-- Ceci est un commentaire -->
```

Il est possible de l'insérer n'importe où au sein du code source : il n'a aucun impact sur la page, mais ils servent à se repérer dans le code source (surtout s'il est long).

```
<!DOCTYPE html>
<html>
  <head> <!-- En-tête de la page -->
    <meta charset="utf-8" />
    <title>Titre</title>
  </head>
  <body> <!-- Corps de la page -->

  </body>
</html>
```

III. Un mot sur la sémantique et la relation parent/enfant

1. La sémantique HTML

Les langages HTML et CSS ont connus de nombreuses révisions depuis leurs premières standardisations, fin 1990. Les années 2000 et l'évolution des langages ne furent pas sans poser quelques problématiques.

Pour donner une idée, à l'époque, ont cohabitées en 2011 les spécifications HTML 4.0 strict, 4.01 transitionnel, XHTML 1.0, XHTML 1.1 et CSS1, CSS2, etc., certaines obligeant à une façon de faire et d'autres pas, certains navigateurs les implémentant et d'autres pas, et ce généralement sur de petits détails de code (en pratique toutefois, les spécifications W3C et les révisions associées ayant des raisons d'être plus complexes, voir lien en fin de document pour plus d'informations).

Avec HTML5 et CSS3, révisions sorties fin 2014, le W3C eut pour objectif "un seul web et pour tous", répondant ainsi aux réfractaires de la profession se plaignant de la multiplicité des standards HTML/CSS et de leur opacité.

HTML5 apporte avec lui de nombreuses balises inédites et une problématique nouvelle : structurer sa page doit aujourd'hui se faire avec une idée de son interprétation par un moteur de recherche.

Prenez cet exemple :

```
<!DOCTYPE html><!--rappel : indication au navigateur que la page
est écrite en HTML5-->
<html>
  <head>
    <meta charset="utf-8"/>
    <link href="style.css" rel="stylesheet"/>
  </head>
  <body>
    <div id="article">
      <div id="titre">Titre de l'article</div>
      <div id="image-article">
        
      </div>
      <div id="paragraphe" class="paragraphe-1">
        Lorem ipsum dolor sit amet, consectetur adipiscing
elit, sed do
        eiusmod tempor incididunt ut labore et dolore magna
aliqua.
      </div>
      <div id="paragraphe" class="paragraphe-2">
        Ut enim ad minim veniam, quis nostrud exercitation
ullamco laboris nisi
        ut aliquip ex ea commodo consequat.
      </div>
      <div id="paragraphe" class="paragraphe-3">
        Duis aute irure dolor in reprehenderit in voluptate
velit esse
        cillum dolore eu fugiat nulla pariatur.
      </div>
    </div>
  </body>
</html>
```

Et l'aperçu de la feuille de style associée (style.css) :

```
#titre{
  font-size: 2em;
  font-weight: bold;
  margin-bottom: 1em;
}
#image-article{
  margin-left: 40px;
}
.paragraphe{
  margin: 1em 0;
}
```


Ce qui donnera comme résultat sur un navigateur :



Pas trop mauvais comme rendu, pour un début ? Sauf que cette page ne respecte rien de la sémantique HTML5.

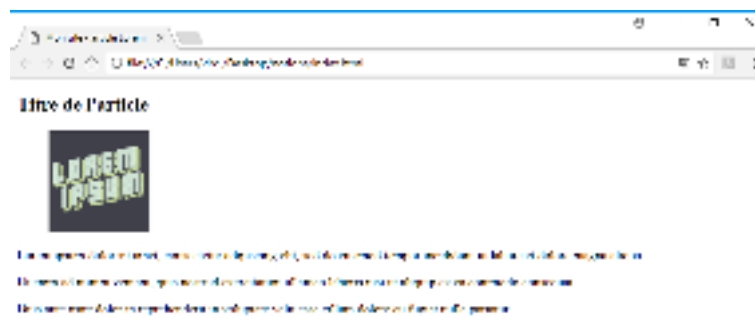
Les balises utilisées sont des "div", de simples divisions de pages qui n'apportent aucun sens intrinsèque au contenu qui y est placé. Les moteurs de recherche interpréteront les textes de cette page sans distinction d'importance ou de rôle :

- aucune différence entre le titre et un paragraphe,
- aucun texte alternatif pour l'image,
- pas de titre pertinent dans l'onglet du navigateur
- pas même la possibilité de déchiffrer la nature de cet ensemble de textes...

Voici une version web sémantique du code de cette même page, plus conforme à ce qu'attend le standard HTML5 :

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8"/>
    <title>Mon site - article Lorem Ipsum</title>
  </head>
  <body>
    <article>
      <h1>Titre de l'article</h1>
      <figure>
        
      </figure>
      <p>
        Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
      </p>
      <p>
        Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
      </p>
      <p>
        Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
      </p>
    </article>
  </body>
</html>
```

Et son résultat :



Remarquez un détail d'importance : il n'y a plus de feuille de style sur cette page, pas de CSS. C'est le même rendu qu'auparavant uniquement en utilisant les bonnes balises. Double avantage : moins de CSS à écrire et plus de sens apporté au contenu pour les moteurs de recherche, qui le référenceront plus efficacement.

Comment est-ce possible ? Tout simplement parce que certaines balises sémantiques comportent du CSS par défaut, que tous les navigateurs leur appliquent. Ainsi :

- La balise `p`, sémantiquement réservée à un paragraphe, possède des marges en haut et en bas mesurant 1 em, c'est-à-dire une unité de caractère.
- La balise `h1` également, qui elle sémantiquement habille le titre hiérarchique de niveau 1 de la page, en doublant la taille de la police et la mettant en gras automatiquement.
- La balise `figure` a pour objectif d'entourer une image légendée, possède des marges par défaut elle aussi (1 em en haut et en bas, 40 pixels à gauche et à droite). Ici pas de légende (balise `figcaption`), puisque la spécification HTML5 la permet facultative.
- La balise `img` possède l'attribut "alt", ou texte alternatif. Ce texte remplacera l'image lorsque celle-ci ne peut être affichée (problème de fichier ou affichage de la page dans un navigateur spécial pour malvoyants). Ce texte sera également pris en compte comme la définition de l'image référencée par les moteurs de recherche.
- Et enfin, la balise `article` : elle n'apporte aucune différence visuellement d'une `div` (pas de CSS particulier), son intérêt n'est que sémantique. Une division de page `lambda` n'indique pas le rôle de ce bloc de contenu, là où `article` explicite sa nature d'article (dans le sens article d'un blog ou d'un site informatif)

2. La relation parent/enfant

En HTML, toute balise (enfant) placée dans une autre (parent) recevra certaines caractéristiques de cette dernière, influençant son aspect ou sa position sur la page, mais également son intérêt à être placée là et pas ailleurs, en référence à son rôle sur cette page.

Selon l'exemple ci-dessus :

- Une balise `p` est sensée contenir le texte du paragraphe qu'elle représente. Contenir autre chose mais pas de texte sous-entend que cette balise ne devait pas être un `p` (par exemple, si un `p` ne contient que des images, alors `p` est abscons, figure est plus appropriée).
- La balise `h1` contient le titre de niveau 1 de la page. Dans le cas où le texte s'y trouvant est une phrase plus ou moins longue, ou autre chose, la sémantique d'une `h1` n'est pas respectée. Il faut alors revoir la conception de cette page (réduire le texte pour en faire un titre ou changer de balise...)
- De même que la balise `article` sert de conteneur à tout ceci. Logiquement, un article au sens où on l'entend possède un titre, du texte découpé en paragraphes et si besoin quelques illustrations. Utiliser cette balise apporte à elle seule tout ce sens rédactionnel à son contenu. Et plus encore, ajoutez à votre feuille de style CSS :

```
<link rel="stylesheet" href="style.css" />
```

... ce morceau de code :

```
article{
  width: 600px;
  border: 2px solid #bbb;
  padding: 1em;
  margin: 0 auto;
  font-family: Verdana, sans-serif;
  background-color: #aabbcc;
  text-align: justify;
}
```

Et vous obtiendrez ceci :



Incroyable ! Tout ça en apportant du CSS uniquement à la balise article ! Oui, c'est l'avantage de la notion de relation parent/enfant : les enfants subissent ce que leur parent est (comme dans la vraie vie, au fond...) :

- ✚ Ainsi, si article détermine que le texte s'y trouvant doit être justifié, tout texte à l'intérieur le sera, ce même si une balise intermédiaire habille également le texte (text-align : justify). Nul besoin de dire à tous les p de le faire chacun. Même chose pour la police à afficher (font-family).
- ✚ La largeur de l'article a été fixée (width: 600px), ce qui oblige les enfants de cette balise à s'y conformer : les paragraphes ne peuvent pas dépasser à droite, donc ils mettent leur texte à la ligne, augmentant leur hauteur en conséquence.
- ✚ L'article définit une marge interne d'un caractère de long (padding: 1em) sur tous les côtés. Le contenu de l'article se trouve donc obligé de le respecter (les paragraphes sont espacés des bords de l'article).
- ✚ Enfin, l'article détermine ses propres marges gauche et droite à auto. Cette astuce permet de centrer l'article dans la fenêtre du navigateur en lui demandant de calculer automatiquement la taille des marges latérales en fonction de l'espace restant, et ce quelle que soit la largeur de cette fenêtre. Le contenu de l'article subit donc naturellement le centrage de son parent.

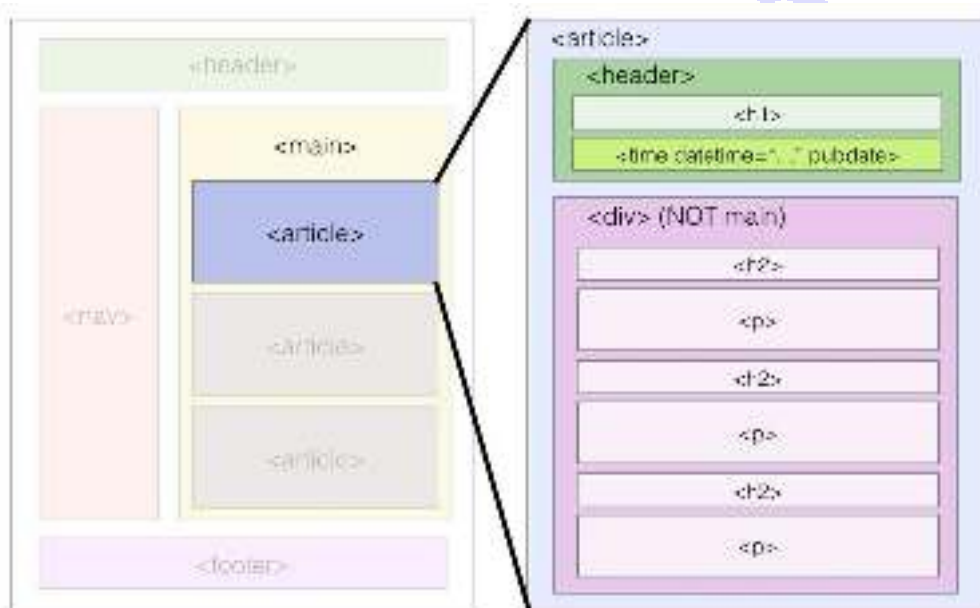
Toute l'importance de la notion d'héritage entre balises "parent" et "enfant" est là : choisir et placer les balises selon leur contenu, leur utilité et leur sens autant rédactionnel que sémantique.

Etant donné les centaines de balises existantes et leurs spécificités propres, ce document ne va logiquement pas toutes les énoncer. Cependant, le but de cette partie est de vous sensibiliser à cette problématique et de ne JAMAIS la sous-estimer. Rassurez-vous, la pratique du HTML/CSS apportera au fur et à mesure les automatismes nécessaires et la compréhension des comportements qui feront de vous un bon webdesigner.

Un bon webdesigner structure et met en forme des pages web avec une vision précise de leur usage et leur rendu finaux. C'est également un professionnel qui se renseigne constamment sur les bonnes pratiques, met à jour ses compétences au fil du temps et remet en question ses certitudes en s'appliquant ce qu'on appelle une veille technologique.

Après cette longue introduction, ce qui suit couvre les plus importantes notions HTML/CSS à connaître impérativement, sous peine de n'obtenir la page désirée que par bidouillages et omissions de concepts indispensables.

IV. Structure d'une page HTML5 et balises courantes



La partie de gauche représente la structure de la page globale alors que la partie de droite représente la structure d'un contenu, de façon plus détaillée. Ce schéma fait ressortir une structure très courante de balises :

- 🚩 `<header></header>` : concerne l'en-tête du site ou d'un article. Généralement la partie qui contient un gros titre ou des informations capitales.
- 🚩 `<nav></nav>` : comme son nom le laisse supposer, il s'agit de « la boîte » contenant le menu du site.
- 🚩 `<main></main>` : représente le contenu principal.
- 🚩 `<article></article>` : définit une zone de contenu.
- 🚩 `<section></section>` : permet un découpage de la page en plusieurs parties, comme un livre est découpé en chapitres.
- 🚩 `<footer></footer>` : concerne le pied du site ou d'un article.

V. Introduction au CSS

Le CSS (Cascading Style Sheets) est un langage qui vient compléter le HTML : il gère la mise en forme d'un site. Né 5 ans après le HTML, c'est-à-dire en 1996, son but était de rendre moins complexe les pages HTML, dans lesquelles aussi bien le fond et la forme y étaient gérés.

1. Mise en place

Le CSS s'écrit dans un fichier dont l'extension est en « .css » et est implémenté dans une page HTML grâce à la balise `<link />` qui sera ajoutée dans le `<head>`.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <link rel="stylesheet" href="style.css" />
    <title>Premiers tests du CSS</title>
  </head>
  <body>
    <h1>Mon super site</h1>
    <p>Bonjour et bienvenue sur mon site !</p>
    <p>Pour le moment, mon site est un peu <em>vide</em>.
    Patientez encore un peu !</p>
  </body>
</html>
```

La ligne `<link rel="stylesheet" href="style.css" />` indique que la page HTML doit rechercher un fichier nommé « style.css » se situant dans le même dossier que le fichier HTML. Il est important de bien faire attention au chemin d'accès au fichier, sinon rien ne fonctionnera.

Schématiquement, une feuille de style CSS ressemble donc à ça :

```
balise1 {
  propriete: valeur;
  propriete: valeur;
  propriete: valeur;
}
balise2 {
  propriete: valeur;
  propriete: valeur;
  propriete: valeur;
}
balise3 {
  propriete: valeur;
}
```

Dans un code CSS comme celui-ci, on trouve 3 éléments différents :

- 🚦 **Des noms de balises** : on écrit les noms des balises dont on veut modifier l'apparence. Par exemple, pour tous les paragraphes <p>, il faut écrire p.
- 🚦 **Des propriétés CSS** : les "effets de style" de la page sont rangés dans des propriétés. La propriété *color* permet d'indiquer la couleur du texte, *font-size* qui permet d'indiquer la taille du texte, etc.
- 🚦 **Les valeurs** : pour chaque propriété CSS, on doit indiquer une valeur. Par exemple, pour la couleur, il faut indiquer le nom de la couleur. Pour la taille, il faut indiquer quelle taille on veut, etc.

Si 2 balises doivent avoir la même présentation, il suffit de combiner la déclaration en séparant les noms des balises par une virgule comme ceci :

```
h1, em {  
    color: blue;  
}
```

Les commentaires CSS s'écrivent de la façon suivante : */* commentaire */*

```
/*  
style.css  
-----  
*/  
p {  
    color: blue; /* Les paragraphes seront bleus */  
}
```

2. Appliquer un style : class et id

La méthode énoncée précédemment présente tout de même un défaut : cela implique par exemple que TOUS les paragraphes soient écrits par exemple en bleu.

Pour résoudre le problème, on peut utiliser ces attributs spéciaux qui fonctionnent sur toutes les balises : **l'attribut « class » et l'attribut « id »**.

L'attribut « class » est un attribut qui peut être mis sur n'importe quelle balise et **plusieurs balises peuvent avoir la même valeur d'attribut « class »**.

```
<h1 class="mon-titre"> </h1>  
<p class="bleu"></p>  
<img class="bleu" />
```


Dans le fichier CSS, indiquer le nom d'une classe se fait en précédant le nom de la classe par un point, comme ceci :

```
.bleu {  
    Color : darkblue;  
}
```

L'attribut « id » fonctionne exactement de la même manière que class, à un détail près : **il ne peut être utilisé qu'une fois dans le code.**

En pratique, l'attribut « id » ne se place que sur des éléments qui sont uniques sur la page, comme un logo par exemple :

```



```

Si on utilise des id, dans le CSS il faudra faire précéder le nom de l'id par un dièse (#) :

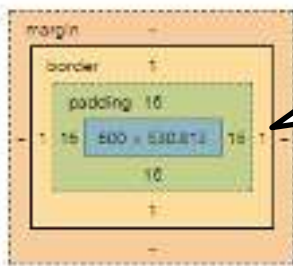
```
#logo {  
    /* Indiquez les propriétés CSS ici */  
}
```

VI. La propriété CSS display

De toutes les propriétés CSS que les navigateurs appliquent par défaut aux balises HTML, display est la plus importante. Elle régit le comportement de base de la "boîte" représentant la balise à l'écran. Elle peut prendre de nombreuses valeurs, mais pour bien comprendre, voici les deux valeurs principales :

-  **Block** : une balise de type "block" se comportera comme un bloc de contenu qui fera toute la largeur de son parent "block". Pour exemple : un paragraphe p fera 100 % de la largeur du bloc article dans lequel il est placé. Des balises "block" successives s'empileront donc verticalement.
-  **Inline** : une balise de type "inline" se comportera non comme une boîte, mais comme un mot dans une phrase (on dit qu'elle s'écoule selon le flux du texte). Ce qui veut dire que la boîte qui la délimitera fera la taille (en hauteur et en largeur) de son contenu et plusieurs balises "inline" successives s'afficheront les unes à côté des autres, horizontalement. Ainsi, une balise img occasionnera une boîte à l'écran de la même taille en pixels que le fichier image associé à son attribut src.

1. Le concept de "boîte"



Ce visuel présente l'espace en pixels occupé par la balise article de l'exemple précédent.

Il peut être obtenu en inspectant la page (clic droit → inspecter l'élément)

Ce qui est défini comme "boite" est l'espace rectangulaire qu'occupera la balise "block" à l'écran. Cet espace est d'abord influencé par la propriété display de ladite balise utilisée, mais également en CSS par :

- ✚ Width et height : largeur et hauteur. Si ces valeurs sont fixées dans la feuille de style, la boîte et son contenu seront contraints de les respecter. Attention : si la hauteur fixée n'est pas suffisante pour le contenu de la boîte, ce contenu dépassera vers le bas et "débordera" visuellement des bordures de la boîte. C'est pourquoi, généralement, il faut laisser le contenu d'une boîte lui dicter sa hauteur.
- ✚ Margin : les marges externes à la boîte, qui allongeront l'espace occupé d'autant de pixels (ex : une boîte de 600 px de large ayant des marges gauche et droite de 20 px fera à l'écran 640 px de large).
- ✚ Border : la bordure de la boîte, par défaut à 0. Spécifier une épaisseur, un type de bordure (solide, hachurée, pointillée, enfoncée...) et une couleur révéleront les contours de la boîte mais l'allongera d'autant de pixels à l'écran. (ex : une boîte de 600 px de largeur et une bordure de 2 px d'épaisseur fera à l'écran 604 px de large).
- ✚ Padding : les marges internes, qui sépareront le contenu de la boîte de sa bordure d'autant de pixels. Elles auront la même influence que les marges externes sur la largeur réelle de la boîte, mais la propriété

2. Un mot sur la propriété box-sizing

Ce qui suit est une bonne pratique conseillée par de très nombreux professionnels pour faciliter le dimensionnement des boîtes en CSS. La propriété box-sizing permet au navigateur de modifier le calcul de la largeur et la hauteur qu'occupera une boîte à l'écran.

Par défaut, toute balise "block" est implémentée avec la propriété **box-sizing : content-box**, qui spécifie que le contenu seul respectera la largeur et la hauteur de la boîte où il se trouve (autrement dit, toute marge, interne ou externe, et toute bordure augmentera les dimensions de la boîte).

Il est préférable en début de feuille de style de régler la propriété box-sizing à border-box sur toutes les balises de la page, comme suit :

```
*{ /* l'étoile indique que toutes les balises de la page devront  
respecter cette règle*/  
  box-sizing: border-box;  
}
```

Cette règle obligera le navigateur à inclure dans le calcul des dimensions de toute boîte "block" son padding et sa bordure, forçant le contenu à disposer de moins d'espace pour s'étirer. Ainsi, une boîte de 600 px de large ayant :

```
margin : 20px;  
padding : 20px;  
border: 1px solid black;
```

...n'occupera à l'écran que 640 px de large et non 682 pixels ! Idéal pour maîtriser convenablement les dimensions complexes que peuvent nécessiter les éléments

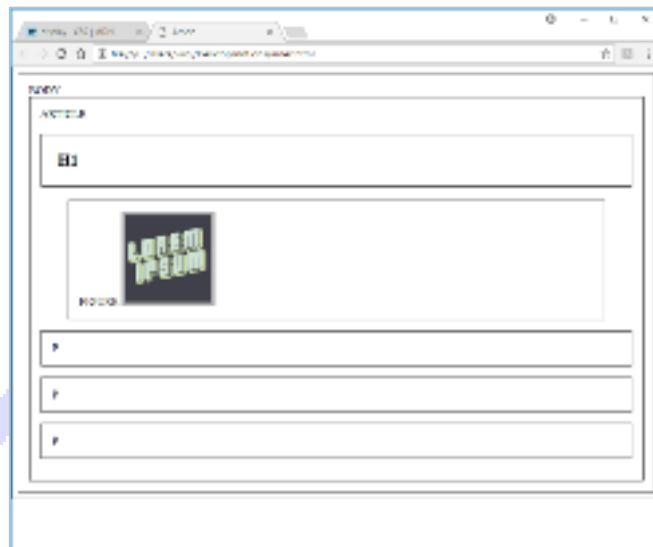
de certaines mises en page, car seules les marges externes auront une influence sur la taille globale de la boîte.

3. L'enchaînement des boîtes à l'écran

Ajoutez ce code à la place du CSS de l'exemple précédent :

```
*{ /*toutes les balises auront une bordure noire pleine de 1px d'épaisseur*/
  border: 1px solid black;
}
img{ /*la seule balise inline de cette page aura une bordure rouge de 2px d'épaisseur, ce qui réécrit (on dit 'surcharge') la règle précédente */
  border: 2px solid red;
}
article, h1, p, figure{ /*toutes les balises block auront une marge interne de 1em (16px) */
  padding: 1em;
}
```

...et vous obtiendrez ceci :



L'enchaînement des balises est mieux visible à l'écran. Notez deux particularités :

- La balise `body` est une balise de type "block". Comme elle a pour rôle de contenir l'intégralité du contenu de la page, il est normal qu'elle ait ce comportement de boîte par défaut. Mais observez le contour supplémentaire au-dessus de la boîte `body` : c'est la boîte de la balise `html`, elle aussi de type "block", qui est réellement la boîte comportant tout le contenu du site (elle ne contient en réalité que `body`, mais par ricochet, puisqu'elle contient `body`, elle contient tout le site). Observez également le petit espace entre `html` et `body` : ce sont les marges externes CSS par défaut de la balise `body` (8 px de tous côtés).
- La bordure de la balise `figure` ne démarre et ne se termine pas au même niveau, horizontalement, que les autres. Ce sont ses marges externes par défaut (1 em en haut et en bas, 40 px à gauche et à droite) qui empêche sa boîte de se délimiter plus loin. A l'intérieur de `figure`, le texte et l'image se placent côte à

côte puisque la balise `img`, de type "inline", se comporte comme du texte. La boîte de cette image est entourée d'une bordure épaisse délimitant l'espace occupé à l'écran par l'image-source (ici, le fichier fait 128*128 pixels, la boîte en fait donc de même).

a. Valeurs de base de display

Il existe d'autres valeurs pour la propriété `display`, plus spécifique à certaines balises ou pour certaines mises en page précises à l'écran. Il n'est pas nécessaire de les connaître toutes, étant donné qu'elles sont définies par défaut en CSS.

Voici un tableau récapitulatif des principales valeurs de `display`, implémentée par défaut par les navigateurs :

Valeur de display	Comportement de boîte	Restrictions
block	Fait 100% de la largeur de son parent block S'empile de haut en bas. Sa hauteur est dictée par son contenu	A ne pas placer dans une balise inline. Laisser le plus souvent possible le contenu définir la hauteur de la boîte (éviter de fixer la hauteur en CSS)
inline	Fait la largeur et la hauteur de son contenu S'écoule selon le flux d'un texte (de gauche à droite)	Ne peut contenir que des balises "inline" uniquement. Impossible de spécifier une largeur ou hauteur différente de celles que forcera le contenu.
table, table-row, table-cell...	Par défaut, spécifiques aux balises de structure de tableau (<code>table</code> , <code>thead</code> , <code>tbody</code> , <code>tr</code> , <code>td</code> , <code>tfoot</code>) Permettent une certaine souplesse dans la construction d'un contenu tabulé (listes de noms, etc.)	Réservées aux balises mentionnées ci-contre. Connait beaucoup de variantes (voir lien en fin de document)
list-item	Par défaut, spécifique aux balises étant des éléments de listes (<code>li</code>) placées dans une liste ordonnée ou non (<code>ol</code> ou <code>ul</code>).	Réservé aux éléments de liste. Laisser le navigateur interpréter le contenu d'une <code>ul</code> ou <code>ol</code> de lui-même.

b. Les valeurs de display spécifiques

La plupart du temps, il est inutile voire dangereux de changer la valeur de la propriété display d'une boîte dans votre propre feuille de style, sous peine de ne plus respecter les standards du web et donc de voir le contenu de la page mal interprété par le navigateur.

Trois valeurs supplémentaires sont disponibles en CSS pour apporter à la boîte d'une balise un comportement plus précis, en fonction du résultat attendu.

Valeur de display	Comportement de boîte	Restrictions
none	N'affichera pas la boîte à l'écran : son contenu sera absent à l'écran et n'occupera pas l'espace normalement sollicité	Ne rend pas le contenu "invisible" pour autant, à ne pas utiliser pour cacher du contenu sensible aux visiteurs du site ! L'utilité de "none" réside dans le fait de masquer du contenu qui se révélera à l'utilisateur lors d'une manipulation (un menu s'affichant au survol d'un lien, etc.)
inline-block	Hybride d'inline et de block : la boîte aura le comportement d'un block (largeur et hauteur personnalisables, peut contenir d'autres "block") tout en s'écoulant horizontalement selon le flux du texte (comme un "inline"). Peut être très utile	Est assez capricieux à utiliser, la faute aux navigateurs ne l'implémentant pas toujours correctement. A tester
flex/inline-flex grid/subgrid	Nouveaux comportements de boîtes arrivées avec la spécification CSS3, ces deux valeurs sont très puissantes et permettent de structurer du contenu complexe dans une démarche "responsive-design"	En fin de document, un chapitre y est consacré

VII. Les boites flottantes

Reprenez l'article en exemple plus haut : l'image contenue dans la balise figure occupe un espace de block, c'est-à-dire toute la largeur de l'article pour elle seule. Ce qui explique le vide à sa droite. Cette figure pourrait, pour récupérer l'espace perdu, se présenter en "médaillon" au texte de l'article : elle "flottera" dans l'article, le texte épousant ses contours.

Ajoutez ce code CSS à votre feuille de style :

```
figure{  
    float: left;  
}
```

...et le résultat :



La propriété CSS float est souvent mal perçue, mal utilisée : elle entraîne de nombreuses subtilités qu'il faut anticiper sous peine de ne plus maîtriser les balises suivant la boite flottante. Retenez bien que :

🚦 La boite flottante n'est plus considérée comme un enfant dans le flux de l'article : si l'article ne comportait plus de paragraphes, la figure sortirait visuellement de l'article. Commentez-les et constatez le résultat ci-contre. La figure n'étant plus "dans le flux", l'article ne peut la contenir. Et ce dernier n'ayant plus de contenu, sa hauteur ne sera que celle que le titre lui soumet. Ce comportement constitue la principale source d'erreurs de mise en page avec des boites flottantes.



🚦 Les flottantes ne font, pour ainsi dire, qu'occuper un espace gauche (float: left) ou droite (float: right) dans la boite parente. Cet espace leur est réservé, et le flux suivant (les boites suivantes non-flottantes) ne pourra s'y placer, il ira donc s'inscrire dans l'espace restant.

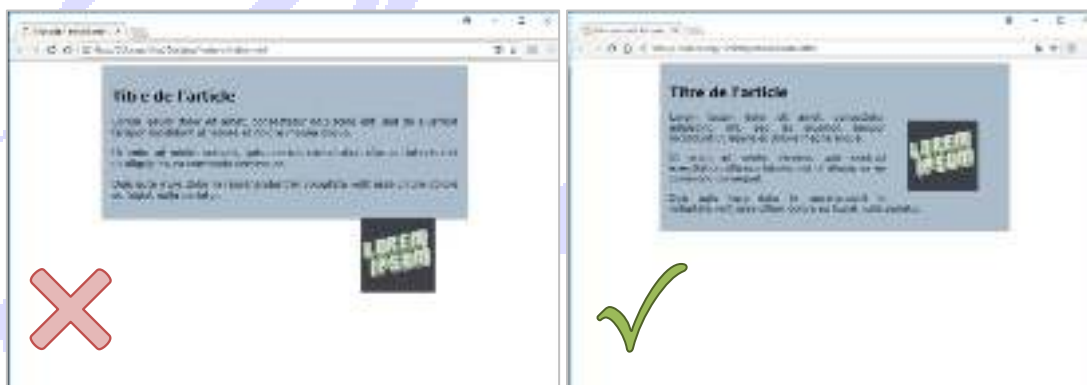
Attention : TOUTES les balises suivant une flottante subissent l'espace occupé par elle, même si ces balises ne font pas partie du même parent que la boîte flottante !

- Une balise flottante doit être, dans le parent où elle se trouve, placée avant tout contenu dudit parent. Ce qui est logique puisque la boîte flottante se réservant un espace gauche ou droite, le contenu suivant doit subir sa présence. Si vous inversez l'ordre des balises de l'article comme suit :

```
...  
<article>  
  <h1>Titre de l'article</h1>  
  <p>  
    Lorem ipsum ...  
  </p>  
  <p>  
    Ut enim ad ...  
  </p>  
  <p>  
    Duis aute irure ...  
  </p>  
  <figure>  
      
  </figure>  
</article>  
...
```

Les paragraphes étant placés avant la flottante, ils n'ont aucune raison de ne pas occuper leur espace "block" normal : 100 % de la largeur de l'article. La flottante, placée à la fin de l'article, ne fera subir sa présence qu'à ce qui est écrit après elle (en l'occurrence ici : rien).

Attention : tout ceci est également valable pour une flottante à droite ! Elle doit être placée avant le contenu devant la subir, alors que la logique de notre lecture "de gauche à droite" voudrait qu'elle soit naturellement placée après :



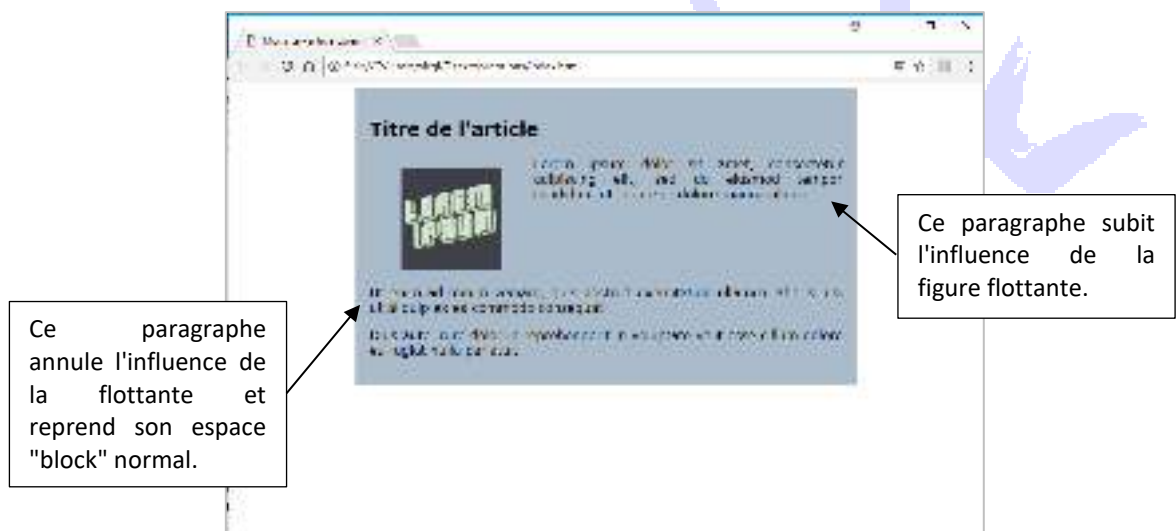
Float:right sur figure placée après les p

Float:right sur figure placée avant les p

- Enfin, si l'influence de la flottante doit cesser à partir d'une boîte précise, dans le but de forcer les balises suivant la flottante à récupérer leur espace de "block" normal, il faut ajouter à la première balise concernée la propriété CSS clear.

Clear connaît trois valeurs : left (annule l'influence des flottantes à gauche), right (annule l'influence des flottantes droite) et both (annule l'influence de toutes les flottantes précédant cette balise). Rajoutez ceci à votre code CSS :

```
figure{
    float: left;
}
article p:nth-child(4){ /*pas de panique, cette écriture est
expliquée dans ce document*/
    clear: left;
}
```



Certains webdesigners ont tendance à abuser des balises flottantes pour obtenir le résultat escompté. Généralement, ceux-ci méconnaissent d'autres propriétés qui auraient sensiblement simplifié leur code et amélioré l'adaptativité (responsive) de la page sur un mobile.

En conclusion, il ne faut pas voir en "float" une solution miracle à tout. Il est plus prudent de réserver cette propriété à ce qu'elle permet dans son utilisation simple (une image en médaillon d'article, une lettrine de texte, un bouton ou une note incorporée à un contenu conséquent...).

Un classique de mauvaise utilisation des flottantes : les éléments d'un menu horizontal qui flottent tous pour se mettre à la queue leu leu... Faire ceci implique de nombreuses précautions en CSS alors qu'il existe de bien meilleures manières d'obtenir ce rendu (voir la partie Flexbox).

VIII. L'overflow

Sûrement la propriété la plus méconnue de CSS, l'overflow détermine le comportement du contenu d'une balise lorsque celui-ci ne respecte pas les dimensions de son parent, généralement dans ces deux cas :

- La largeur et/ou la hauteur fixée au parent ne lui permet pas de contenir l'intégralité de son contenu (celui-ci déborde donc)
- Les dimensions du contenu subissent l'influence du positionnement différent d'un élément adjacent.

L'utilité de la propriété overflow se résume à garder le contrôle des dimensions du contenu dans une boîte en le rognant ou en lui permettant d'afficher des barres de défilement.

Dans l'exemple (ajoutez deux paragraphes supplémentaires à l'article pour la démonstration) et ajoutez une classe au premier paragraphe :

```
...
<article>
  <h1>Titre de l'article</h1>
  <figure>
    
  </figure>
  <p class="trop-large"><!--ce paragraphe sera plus large que
l'article -->
    Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed
do
    eiusmod tempor incididunt ut labore et dolore magna aliqua.
  </p>
  <p>
  ...
```

Le paragraphe "trop-large" fera 200 px de plus en largeur que son conteneur (l'article) :

```
article{
  width: 600px;
  overflow: /* c'est ici que vous testerez les différentes
valeurs d'overflow*/;
  border: 2px solid #bbb;
  padding: 1em;
  margin: 0 auto;
  font-family: Verdana, sans-serif;
  background-color: #aabbcc;
  text-align: justify;
}
figure{
  float: left;
}
.trop-large{
  width: 800px;
}
```

Overflow peut prendre les valeurs suivantes :



Visible : la valeur par défaut. Le contenu débordera des dimensions de son parent si celles-ci sont fixées à des valeurs inférieures. Ici, l'article ne faisant que 600 px de large, le paragraphe "trop-large" est forcé de déborder de 200 px à droite. Le conteneur ne s'allonge pas en conséquence.



Hidden : lorsque le contenu d'une boîte débord des dimensions de son parent, ce contenu est rogné visuellement. Les largeur et hauteur du conteneur sont donc respectées mais des parties du contenu sont pour cela masquées.



Scroll ou Auto : le conteneur fera apparaître des barres de défilement horizontal ou vertical pour permettre de voir le contenu débordant sans redimensionner le conteneur. La valeur auto fera intervenir la ou les barres nécessaires suivant que le contenu dépasse en largeur ou en hauteur, alors que scroll affichera les deux barres, même si l'une d'elles est inutile.

Vous en conviendrez, il est assez rare d'avoir besoin de cette propriété. Néanmoins, elle permet, par une pirouette, d'obtenir un résultat très utile sur l'exemple traité. Enlevez la classe "trop-large" au premier paragraphe, supprimez la propriété overflow de l'article et appliquez les règles overflow: hidden et background-color: yellow sur tous les paragraphes :



Avant, les paragraphes subissaient la flottante mais leur largeur effective continuait à faire 100 % de l'article. Le texte à l'intérieur était seul à réellement s'habiller autour de la flottante (d'où la couleur jaune qui "bave" derrière l'image).



Désormais, les balises p avec overflow: hidden rognent leur largeur pour n'occuper que l'espace nécessaire par leur contenu (le texte). C'est le troisième paragraphe qui est intéressant : sa largeur étant rognée, son contenu s'écrit d'un bloc, et non autour de l'image comme précédemment.

IX. Positionnement avancé et z-index

1. La propriété position

C'est une des propriétés les plus importantes du langage CSS, mais également une des moins faciles à maîtriser : la propriété position. Grâce à elle, il est possible de sortir un élément du flux de contenu normal de son parent et le placer où on le souhaite à l'écran, tout en lui permettant de suivre ou pas le défilement de la page.

Prenez la figure de l'exemple et supprimez sa propriété float, remplacez-là par position: absolute :



L'image passe par-dessus le texte mais reste à sa place initiale. La balise figure est totalement sortie du flux de contenu de l'article. Ajoutez ce code CSS :

```
figure{  
    position: absolute;  
    right: 0;  
    top: 0; /* top, left, right et bottom sont spécifiques au  
    positionnement CSS avancé*/  
}
```



Il semblerait que la figure n'est même plus un enfant de la balise article. Bien sûr que non, notre code HTML n'ayant pas changé, mais par les deux règles top et right à 0, la figure est "aimantée" au coin en haut à droite de la page.

Pourquoi le coin de la page et pas celui de l'article puisque figure en est toujours un enfant ? C'est simple : **un élément positionné en absolute**

s'aimante à son parent positionné le plus proche ! En l'occurrence, article ne dispose d'aucune règle "position" dans le CSS, c'est donc le seul élément HTML positionné par défaut qui "attirera" la figure : la balise html elle-même !

En réalité, toutes les balises HTML ont par défaut la propriété position à static, c'est-à-dire non-positionné. Elles se placent les unes les autres tel que le flux de contenu l'oblige.

Dans l'exemple, s'il s'agit de voir la figure dans le coin en haut à droite de l'article et non de la page entière, il faut positionner l'article : ajoutez-lui la règle position: relative.

(NB : les marges de la figure ont été mises à 0 pour l'exemple ci-contre.)



2. z-index

Bien entendu, l'image passe toujours par-dessus le premier paragraphe, elle en masque une partie du contenu. Ajoutez ce code et il n'y paraîtra plus :

```
article p:nth-child(3){ /* pas de panique, cette notation est
expliquée dans ce document */
    position: relative;
    z-index: 1;
}
```



La propriété z-index permet de spécifier l'ordre des éléments positionnés sur l'axe z (la profondeur). Par défaut, z-index est à auto, les balises sont alors dans l'ordre de leur écriture HTML.


A ce sujet, supprimez la propriété z-index du code précédent et rien ne changera : le p est écrit à la suite de la figure en HTML, donc il sera d'emblée devant l'image sur l'axe z. Ajoutez z-index: 1 à la figure pour la voir repasser devant le paragraphe.

Attention : il est tentant d'écrire un chiffre énorme (comme 100 ou 1000) en z-index pour s'assurer que l'élément visé sera au-dessus de toute autre balise positionnée. C'est inutile ! En revanche, il est possible d'attribuer un index négatif (-1, -2) pour faire passer une balise en dessous d'une autre. A ce moment-là, elle sera également en dessous des balises non-positionnées.

```

figure{
  position: absolute;
  right: 0;
  top: 0;
  margin: 0;
  z-index: -1;
}
p{
  background-color: yellow
}
article p:nth-child(3){
  /*vide*/
}

```



Pour voir l'image, il a été nécessaire d'enlever la couleur d'arrière-plan de l'article, la figure étant passée au-dessous...

3. Tableau récapitulatif

Valeur de position	Comportement	Restrictions
static	La position par défaut de toutes les balises HTML. Les balises suivent leur ordre de flux et ne peuvent passer l'une sur l'autre. N'est pas une "position" au sens où les autres valeurs l'entendent.	z-index et top, right, left ou bottom n'auront aucun effet sur une balise non-positionnée.
absolute	La balise passe au-dessus des éléments non-positionnés. Elle sort du flux de contenu de son parent et s'aligne soit à sa position initiale (là où elle est placée en HTML), soit dans un coin de son premier parent positionné (en utilisant les propriétés top, left, right et bottom).	Il faut s'assurer que le parent sur qui la balise absolue doit "s'accrocher" soit positionné au moins en "relative". Sans valeur de z-index, n'importe quelle balise positionnée et écrite après elle en HTML lui passera par-dessus !
relative	La balise conserve sa position initiale et son comportement dans le flux de contenu de son parent, mais peut être positionnée sur l'axe z et se décaler grâce aux propriétés top, left, right ou bottom.	Voir absolue.
fixed	La balise est comme figée à la fenêtre	Contrairement à une absolue,

	du navigateur : elle ne défile pas avec le reste de la page lorsque le visiteur scrolle. A le même comportement que position: absolute, à une exception (voir ci-contre).	une balise fixed n'accroche qu'à la fenêtre du navigateur : positionner en relative son parent est inutile. Certains navigateurs mobiles peuvent mal interpréter cette position.
sticky	La balise est positionnée comme une relative tant qu'elle est visible à l'écran. Si le visiteur scrolle la page au-delà, elle adopte automatiquement le même comportement qu'une position fixed, en s'accrochant à la fenêtre du navigateur en fonction de la direction du scroll.	Sticky est une valeur très récente en CSS3. En conséquence, seuls les navigateurs les plus récents l'implémentent (sauf Microsoft Edge). Il vaut mieux ne pas l'utiliser et attendre une meilleure compatibilité des navigateurs. Dommage !





Certaines propriétés CSS ne fonctionnent qu'avec des éléments positionnés :

Propriété	Valeurs possibles	Incidence
z-index	Tout nombre entier, positif ou négatif.	Avance ou recule l'élément positionné sur l'axe z (profondeur)
top, right, left, bottom	Toute valeur en pixels, en pourcentage de la taille de son parent ou zéro. Valeur par défaut : auto (différent de zéro ici, c'est le navigateur qui placera l'élément selon sa position et sa place dans le code HTML)	Permet un décalage de l'élément positionné par rapport à sa place d'origine. Une valeur à zéro "colle" l'élément au bord de son parent positionné (haut, bas, gauche ou droite selon la propriété choisie)

X. Les sélecteurs avancés en CSS

Dans le cas d'une mise en page complexe et/ou optimale, certaines règles CSS doivent parfois s'appliquer sur des éléments très précis (ex : le 3^{ème} paragraphe d'un article, un li sur deux dans une liste ul, une balise disposant d'un attribut particulier...).

Pour rappel, les sélecteurs simples en CSS sont :

-  La balise elle-même (p, article, li...)
-  Sa classe (.classe)
-  Son identifiant (#id)
-  Le sélecteur universel (*)

Le langage CSS permet de cibler des balises plus finement, en remplissant certaines conditions grâce aux sélecteurs avancés. Cette partie du support liste tous les sélecteurs existants et présentera leur utilisation dans un exemple final.

1. Les combinateurs/sélecteurs de voisins

a. Sélecteur de descendant

```
div p{  
  ...  
}
```

Séparer deux sélecteurs simples avec un espace est le sélecteur de descendant : cette règle cible tous les paragraphes p contenus dans une div, qu'il y ait des intermédiaires entre la div et le p ou pas.

b. Sélecteur de voisin direct

```
div+p{  
  ...  
}
```

Cette règle cible tous les paragraphes p suivant immédiatement une div dans le HTML (la div et le p sont au même niveau hiérarchique)

c. Sélecteur de voisins

```
div~p{  
  ...  
}
```

Cette règle cible tous les paragraphes p voisins d'une div ayant le même parent (toujours au même niveau hiérarchique)

d. Sélecteur d'élément fils

```
div>p{  
  ...  
}
```

Cette dernière règle cible les paragraphes p directement enfants d'une div (les paragraphes ciblés ainsi sont les enfants p de la div, sans intermédiaire).

2. Sélecteur d'attributs

Un sélecteur puissant en CSS consiste à cibler un élément disposant d'un attribut précis ou selon la valeur donnée à cet attribut. En CSS, il suffit d'entourer l'attribut voulu avec des crochets :

```
a[title] {  
  color: purple;  
}
```

Attention : cette règle cible les balises hyperliens a disposant d'un attribut title, et non pas le contenu de cet attribut. C'est bien tout le texte du lien qui sera coloré en violet !

Quelques exemples :

```
a[href="http://www.facebook.com"] {  
    /* l'élément a ciblé doit avoir cette url précise comme valeur  
    d'attribut href */  
}  
img[alt^="Portrait"] {  
    /* les éléments img ciblés doivent avoir le texte de leur  
    attribut alt qui commence par le mot "Portrait" */  
}  
img[alt^="Portrait" i] {  
    /* Même règle que la précédente, à ceci près que la lettre i  
    ajoutée avant le crochet fermant rend la recherche du mot  
    "Portrait" insensible à la casse */  
}  
a[href$=".com"] {  
    /* les éléments a ciblés contiennent une url finissant par  
    ".com" dans l'attribut href */  
}  
a[href*="#"] {  
    /* l'élément a ciblé doit avoir dans la valeur d'attribut href  
    au moins un # */  
}  
img[alt~="maison"] {  
    /* les éléments img ciblés doivent avoir au moins le mot  
    "maison" séparé des autres mots par des espaces dans le texte de  
    leur attribut alt (texte alternatif) */  
}  
a[href$=".com"][title^="Site"]{  
    /* Il est également possible de cumuler les conditions de cette  
    manière */  
}
```

3. Pseudo-classes

Les pseudo-classes sont des mots clés prédéfinis en CSS permettant à la balise ciblée par le sélecteur de comporter une mise en forme particulière selon son statut (Nième balise de ce type dans un parent, élément requis ou désactivé d'un formulaire...) ou un évènement survenant sur lui (survol de la souris, clic maintenu...).

Une pseudo-classe se déclare ainsi : sélecteur:pseudo-classe{...}

Quelques exemples :

```
a:hover { /* les balises a survolées par le pointeur de la souris
se souligneront */
    text-decoration: underline;
}
a:visited { /* les balises a ayant déjà été cliquées une fois se
coloreront en vert */
    color: green;
}
input[type="text"]:focus{ /* les champs texte d'un formulaire se
borderont de vert si ce champ est sélectionné (au clavier ou à la
souris). */
    border: 1px solid green;
}
button:active{ /* le bouton se remplira en jaune à l'action du
clic sur celui-ci jusqu'à ce que le clic soit relâché */
    background-color: yellow;
}
button:not(.petit-bouton){ /* tous les boutons, sauf ceux ayant la
classe "petit-bouton" doubleront la taille de leur police */
    font-size: 2em;
}
input[type="checkbox"]:checked{ /* les champs "case à cocher" des
formulaires seront en rouge lorsqu'ils sont cochés */
    background-color: red;
}
input[type="radio"]:default{ /* le champ "bouton radio" coché par
défaut (ayant l'attribut checked) sera rempli en jaune - ATTENTION
: ne pas confondre l'attribut checked et la pseudo-classe :checked
-- */
    background-color: yellow;
}
input[type="text"]:required:empty{
    /* les champs texte obligatoires d'un formulaire (comportant
l'attribut required), s'ils sont laissés vides, se borderont de
rouge */
    border: 1px solid red;
}
```

Les éléments les plus disposés à recevoir une pseudo-classe en CSS sont bien entendu les balises sur lesquelles le visiteur peut interagir : les liens hypertextes et les formulaires. Mais il est également possible d'appliquer ces pseudo-classes à d'autres balises pour leur permettre de réagir aux actions de l'utilisateur (survol d'un élément de liste li, par exemple).

D'autres pseudo-classes ont pour effet de cibler l'élément en fonction de sa position dans le code HTML :

```
span:first-child{...} /* cible un span étant premier enfant de son parent */
span:last-child{...} /* cible un span étant dernier enfant de son parent */
span:nth-child(an+b){...} /* cible tous les span étant les nièmes enfants de leur parent selon le mode de calcul an+b (ou a et b sont des chiffres)
ex : 3n+1 ciblera tous les 3 span + le premier */
span:only-child{...} /* cible tous les span unique enfant de ce type au sein de leur parent */
```

N'oubliez pas que les exemples ci-dessus sont parmi les plus utilisés, mais qu'il en existe beaucoup d'autres qui ne peuvent pas tous être listé ici.





Faites un tour du côté de la dernière partie de ce document ("Pour aller plus loin") afin de les découvrir !

Il faut néanmoins faire attention au fait que toutes les balises ne peuvent connaître l'état attendu par la définition de ce type de règle CSS. Par exemple, un paragraphe p classique ne peut pas comporter de paramètre "required", donc une règle CSS du type "p:required{...}" n'aura aucun effet et sera même ignorée par le navigateur.

4. Pseudo-éléments

Un pseudo-élément CSS permet de cibler une partie d'un élément HTML, sans avoir besoin de le décrire dans l'arborescence des balises. Par exemple, viser la première ligne d'un paragraphe ou ajouter du contenu textuel avant le contenu réel d'une balise (la notation conseillée est **sélecteur::pseudo-élément**)

Les pseudo-éléments les plus utilisés sont :

-  ::before, ::after : les plus courants, utiles pour apporter du style ou du contenu avant ou après les balises enfants d'un élément.
-  ::placeholder : vise le texte de remplacement intégré dans certains champs de formulaire
-  ::first-line, ::first-letter : vise la première ligne ou la première lettre d'un contenu texte intégré à l'intérieur d'une balise
-  ::selection : permet de styler le texte sélectionné à la souris (en lieu et place du traditionnel bleu des navigateurs)

Les utiliser permettent de varier encore plus la précision des règles CSS d'une page, mais là encore, il ne faut pas s'en servir si ce n'est pas indispensable mais surtout si leur implémentation par les navigateurs n'est pas totale.

5. Exemple récapitulatif

Intégrez les codes HTML et CSS suivants et constatez les résultats :

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8"/>
    <title>Mon site - article Lorem Ipsum</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <form action="index.html">
      <p obligatoire>
        <input type="text" placeholder="Votre nom..."
value="" required/>
      </p>
      <p obligatoire>
        <label for="sexe">
          Vous êtes :
        </label><br/>
        <input name="sexe" value="H" type="radio" checked
required/>
        <span>Homme</span>
        <input name="sexe" value="F" type="radio" required/>
        <span>Femme</span>
      </p>
      <p obligatoire>
        <input type="email" placeholder="Votre e-mail..."
required/>
      </p>
      <p>
        <textarea rows="3" placeholder="Votre
message..."></textarea>
      </p>
      <p>
        <input type="submit" value="Envoyer"/>
      </p>
    </form>
  </body>
</html>
```

```

*{
    box-sizing: border-box;
}
*::selection{
    background-color: black; color: white;
}
html, body{
    width: 100%; height: 100%;
    padding: 0; margin: 0;
    font-size: 16px;
}
form{
    padding: 1em;
    width: 300px;
}
form>p{
    margin: 0; padding: 0.5em;
}
p:nth-child(2n){
    background-color: #ddd;
}
p[obligatoire]::after{
    content: "*";
}
label:first-child{
    color: white;
}
input[type="text"]:required, input[type="email"]:required{
    background-color: pink;
}
input[type="radio"]:checked+span {
    color: blue;
}
input:not([type="submit"]):focus, textarea:focus{
    border: 2px dashed green;
}
*::placeholder{
    font-style: oblique;
}
input[type="submit"]:active{
    background-color: black; color: white;
}

```

XI. flexbox et grid/subgrid

flexbox et grid/subgrid sont des valeurs réservées à la propriété display en CSS, apparues avec CSS3.

Elles répondent aux problématiques de mise en page responsive, devenues indispensables avec la démocratisation des smartphones et des tablettes, du fait de la taille réduite de leurs écrans.

1. flex/inline-flex

Lorsqu'il est souhaité qu'une boîte, composée d'autres boîtes, aligne son contenu horizontalement et/ou verticalement sur une surface de page, deux solutions sont utilisées la plupart du temps :

- 🚦 Faire en sorte que les boîtes contenues flottent (float : left ou right), ce qui oblige à une mesure précise de leurs tailles pour couvrir la zone voulue.
- 🚦 Changer les boîtes de block en inline-block, ce qui engendre quelques problèmes d'affichage, surtout si ces boîtes contiennent d'autres « block ».

C'est alors qu'intervient la notion de conteneur flexible (display : flex). Elle est à appliquer sur le conteneur, cette valeur de la propriété display va étaler le contenu de celui-ci sur toute sa largeur et/ou sa hauteur, en permettant de nombreuses options de comportement, aussi bien au niveau du contenu que du conteneur lui-même.

Attention toutefois : seul le contenu directement enfant d'une flexbox profitera la flexibilité de son parent !

Un exemple classique : la barre de navigation. Soit ce code HTML :

```
<header>
  <div id="logo">LOGO</div>
  <nav>
    <ul>
      <li><a href="">LIEN 1</a></li>
      <li><a href="">LIEN 2</a></li>
      <li><a href="">LIEN 3</a></li>
    </ul>
  </nav>
</header>
```

...qui, sans règles CSS, s'afficherait ainsi :

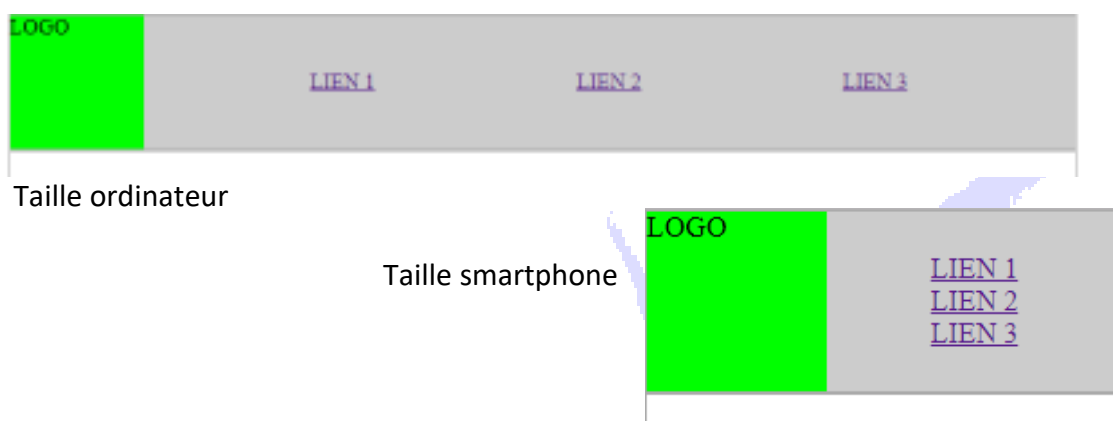


Deux éléments vont alors devenir flexibles : la balise nav, qui étalera le logo et la liste (ul) sur une seule ligne, et cette ul également, qui répandra ses trois éléments de liste (li) sur toute sa largeur.

```
header{
  width: 100%;
  background-color: #ccc;
  display: inline-flex;
}
div#logo{
  background-color: #0f0;
  width: 100px;
  height: 100px;
}
nav{
  flex: 1 0 auto;
  text-align: center;
}
nav ul{
  height: 100%;
  display: flex;
  flex-wrap: wrap;
  align-items: stretch;
  justify-content: space-evenly;
  list-style: none;
  padding: 0;
  margin: 0;
}
nav ul li{
  align-self: center;
}
```

En rouge, les propriétés spécifiques aux flexbox. Un tableau en page suivante les présentera toutes.

Voici le résultat dans deux tailles de navigateur différentes :



La taille et le comportement normal des blocs contenus dans une balise flexible ont été complètement modifiés. C'est tout l'intérêt de flexbox : permettre à une partie d'un site de se comporter selon la largeur de la fenêtre du navigateur et indiquer rapidement à cette partie quoi faire avec son contenu si cette fenêtre se réduit.

Le tableau suivant explique les règles CSS spéciales flexbox du morceau de code précédent. Il en existe de nombreuses autres, non-utilisées ici et très utiles. Découvrez-les par vous-même au chapitre « pour aller plus loin ».

	Propriété	Valeurs possibles	Incidence
A APPLIQUER AU CONTENEUR	display	flex inline-flex	A appliquer à l'élément contenant. Rendra l'élément visé flexible, soit sur une seule ligne (inline-flex), soit sur plusieurs (flex).
	flex-wrap	nowrap wrap wrap-reverse	Indique à l'élément flexible que son contenu pourra, si la taille de la fenêtre l'oblige, passer à la ligne (wrap, wrap-reverse) ou l'interdit (nowrap).
	align-items	flex-start center baseline flex-end stretch	Aligne le contenu de la flexbox en hauteur, ou l'étale sur toute la hauteur disponible (stretch).
	justify-content	flex-start center flex-end space-around space-between space-evenly	Aligne le contenu de la flexbox en largeur ou ajoute de l'espace (entre ou autour) des éléments contenus dans la flexbox.

	Propriété	Valeurs possibles	Incidence
A APPLIQUER AU CONTENU	flex	Trois valeurs à la suite : Flex-grow (numérique) Flex-shrink (numérique) Flex-basis (pixels)	Spécifie au contenu s'il a le droit de s'élargir (flex-grow), de rétrécir (flex-shrink) et s'il doit disposer d'une taille de base indépassable (flex-basis). Les deux premières valeurs sont un entier booléen (1 ou 0)
	align-self	flex-start center baseline flex-end stretch	Similaire à align-items, mais sur le contenu directement. Le contenu s'alignera lui-même dans l'espace rendu disponible par la flexbox.

2. Grid/subgrid

XII. Pour aller plus loin

Sélecteurs CSS :

https://developer.mozilla.org/fr/docs/Web/CSS/S%C3%A9lecteurs_CSS

Flexbox :

https://www.w3schools.com/css/css3_flexbox.asp#flex

<https://css-tricks.com/snippets/css/a-guide-to-flexbox/>

<https://www.alsacreations.com/tuto/lire/1493-css3-flexbox-layout-module.html>

Grid :

<https://www.alsacreations.com/article/lire/1388-css3-grid-layout.html>

https://www.w3schools.com/css/css_grid.asp

https://developer.mozilla.org/fr/docs/Web/CSS/CSS_Grid_Layout/Les_concepts_de_base