

14/03/2022

HTML 5 – Partie 5

Le ZONING

HTML



MODULE B1.WEB

.....	0
Introduction.....	2
Les balises Conteneurs.....	2
La balise <DIV>.....	2
La balise	2
L'élément <header>.....	3
L'ÉLÉment <footer>.....	3
L'ÉLÉment <aside>.....	3
L'ÉLÉment <nav>.....	3
L'ÉLÉment <main>.....	3
L'ÉLÉment <section>.....	3
L'ÉLÉment <article>.....	3
La mise en page des conteneurs en CSS.....	4
La notion de boîte.....	4
La largeur et la hauteur d'une boîte.....	5
Les marges extÉRIEURes.....	7
Les marges intÉRIEURes.....	8
Les bordures.....	9
GERER LA DIMENSION Total d'une boîte.....	11
La couleur des boîtes.....	12
Le positionnement des boîtes.....	12
Positionnement relatif :.....	13
Positionnement absolu :.....	14
Le positionnement fixe.....	15
Le positionnement sticky.....	16
La propriÉTÉ FLOAT :.....	16
Redimensionnement.....	17
Z-index.....	17
La propriÉTÉ display.....	18

INTRODUCTION

Nous avons abordé dans ce cours la plupart des éléments qui constitue le langage HTML : les balises pour le texte, les liens, les images, les tableaux.

Mais ces éléments mis dans une page ne constituent pas un site Web pour autant. L'enjeu est de mettre ses éléments ensembles de façon structurés dans une page pour que le tout forme un tout harmonieux.

LES BALISES CONTENEURS

Les balises conteneurs permettent de structurer vos pages. On parle aussi de l'architecture de la page ou du site. En générale une page Web contient une barre de navigation sur le haut de la page ou de façon latérale, des mentions légales en bas de pages, etc. ...

Enfin d'organiser cela il faut utiliser des "boîtes" dites conteneurs puisqu'elles vont elle-même contenir d'autres éléments HTML, ou même contenir d'autres boîtes.

En HTML4 cette organisation se faisait à l'aide de balise <DIV>. Depuis HTML5 de nombreuses balises sont apparues permettant cela. Toutes ses balises sont du type en bloc.

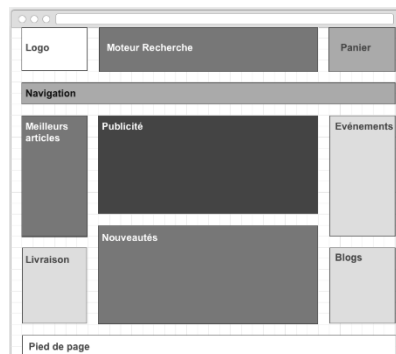
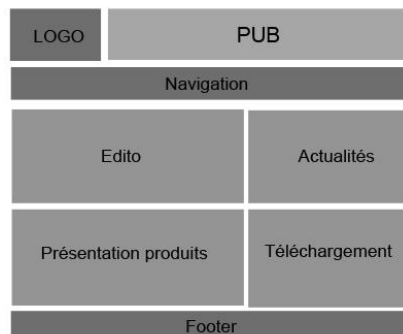
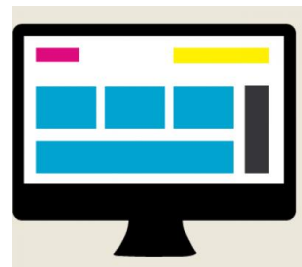
C'est ce que l'on appelle également "**Le zoning**" qui consiste en un découpage d'une page Web schématisée par des blocs ou des boîtes. Il permet d'identifier les différentes zones d'une page Web à savoir :

- Ses grandes fonctionnalités
- Ses zones de contenu

Au-delà des éléments disponibles sur la page, le zoning permet d'identifier :

- La position
- La taille
- Le rôle de chacun de ces éléments

Voici quelques exemples de résultats possibles :



Des outils spécialisés permettent de créer cette étape importante : pencil.

LA BALISE <DIV>

Les boîtes <div> sont souvent utilisées pour avoir des contenus "neutres", sans qu'il y ait besoin d'avoir un contenu sémantique précis. Pour revoir la notion de WEB Sémantique reprendre le cours HTML partie 1.

LA BALISE

 est très proche de l'élément <div>, mais l'élément <div> est un élément de bloc, alors que est un élément en ligne. En termes de sémantique cette balise est neutre. Il peut être utilisé pour grouper des éléments afin de les mettre en forme par la suite en CSS.

Exemple

```
<span>
  <span>home</span>:
  <span>+1.415.555.1212</span> →
</span>
```

home: +1.415.555.1212

L'ÉLÉMENT <HEADER>

Le nouvel élément `<header>` permet d'insérer une zone d'affichage pour les en-têtes. Ces en-têtes peuvent être utilisés à plusieurs endroits :

- Au niveau de la page, c'est le classique en-tête de page, souvent placé en haut de l'écran, avec un logo, un slogan, une barre de navigation principale...
- Au niveau des contenus, cela permet d'avoir une zone d'introduction au contenu qui va suivre, comme l'en-tête d'un article, par exemple.
- Ce conteneur peut contenir tout type d'élément : des titres, des paragraphes, des liens...

L'ÉLÉMENT <FOOTER>

Le nouvel élément `<footer>` permet d'insérer une zone d'affichage pour les pieds de page. Nous retrouvons la même sémantique que pour les en-têtes. Ces pieds de page peuvent être définis pour la page ou pour une autre zone d'affichage de celle-ci, ou pour un article. Notez que l'usage d'un `<footer>` n'implique pas forcément l'usage d'un `<header>`.

L'ÉLÉMENT <ASIDE>

L'élément `<aside>` permet d'afficher un contenu lié à un contenu principal auquel il est associé. Cela correspond souvent aux très classiques barres latérales (sidebar), à des zones de widgets, à des compléments sur des articles ou tout autre contenu rédactionnel.

Vous avez usuellement le contenu principal affiché dans la partie centrale de la page, avec souvent sur la droite l'affichage du contenu associé avec l'élément `<aside>`.

L'ÉLÉMENT <NAV>

L'élément `<nav>`, comme son nom le laisse supposer, est dédié à l'affichage d'une navigation avec des liens hypertextes. Mais attention, ne vous sentez pas obligé de n'avoir qu'une seule zone de navigation par page. Vous pouvez créer autant d'éléments `<nav>` que vous avez de navigations dans vos pages, du moment que chacun d'entre eux est bien identifié. L'élément `<nav>` est plutôt dédié à la navigation principale du site, à la création de liens entre les pages du site.

Vous pourrez inclure une navigation principale `<nav>` dans un en-tête `<header>` et une navigation secondaire `<nav>` dans un pied de page `<footer>`, par exemple.

L'ÉLÉMENT <MAIN>

L'élément `<main>` permet d'indiquer le contenu principal de la page. Ce contenu doit être unique et ne pas être répété dans la page. De plus, le W3C indique précisément son contexte d'utilisation : il ne doit pas être utilisé à l'intérieur (élément descendant) des éléments `<article>`, `<aside>`, `<footer>`, `<header>` ou `<nav>`.

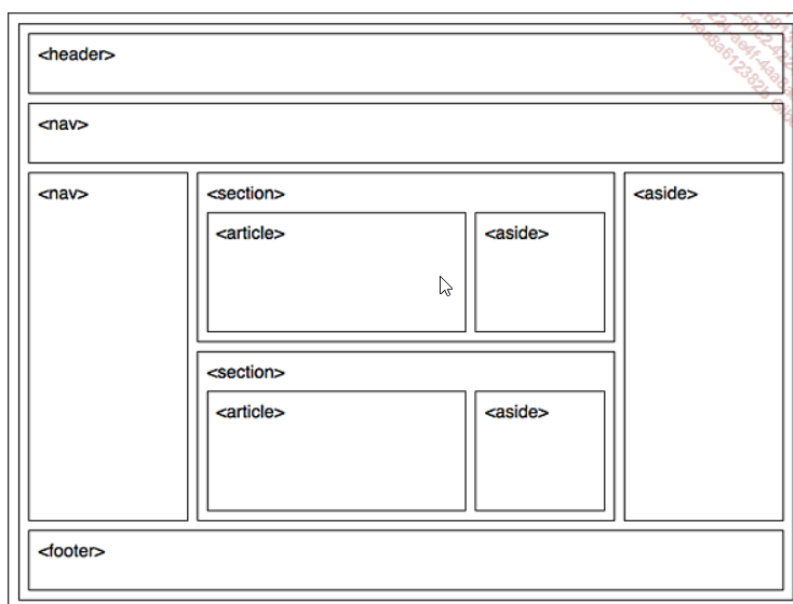
L'ÉLÉMENT <SECTION>

L'élément `<section>` permet de regrouper des éléments partageant une même thématique. Cela permet de regrouper dans un même élément un contenu structuré, avec son en-tête et son pied de page. Cela permet de distinguer plusieurs parties, plusieurs sections au sein d'une même page, avec d'autres éléments de structure imbriqués.

L'ÉLÉMENT <ARTICLE>

L'élément `<article>` permet d'insérer un contenu autonome, car il peut être réutilisé ailleurs dans le site, sans que sa compréhension en soit affectée. L'usage le plus courant reprend le nom de l'élément : création d'articles de blog et d'actualités.

Ainsi à l'aide de ses balises conteneurs il va être possible de structurer sa page de la façon voulue :



LA MISE EN PAGE DES CONTENEURS EN CSS

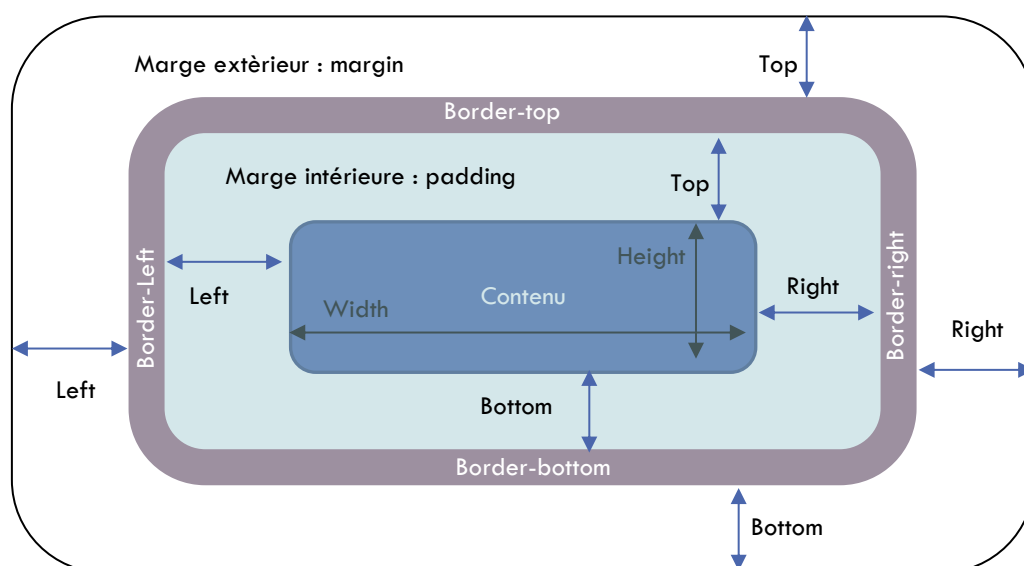
LA NOTION DE BOITE

Nous l'avons déjà vu mais répétons-le les éléments HTML se décomposent en 2 catégories : les éléments en bloc, et en ligne.

Les éléments de type bloc sont conçus pour prendre le maximum de place sur la largeur de la page, et s'adaptent en hauteur en fonction de son contenu. Les éléments blocs sont prévus pour contenir d'autres éléments : `<div>`, `<header>`, `<footer>`, `<main>`, `<section>`, ...

Le CSS sur ce type d'élément permet le positionnement. Ainsi une boîte est constituée

- D'un contenu largeur et hauteur (width et height),
- D'une marge intérieure (padding) dans les 4 directions : top, right, left, bottom
- D'une bordure (border) dimension, style, couleur
- D'une marge extérieure (margin) dans les 4 directions : top, right, left, bottom



La dimension totale d'un élément HTML sera la somme de toutes les dimensions.

Dans les exemples qui suivent nous verrons à chaque fois un exemple avec une balise bloc (`<div>`) et une en ligne (``).

LA LARGEUR ET LA HAUTEUR D'UNE BOITE

Pour spécifier la hauteur et la largeur du **contenu** de votre "boîte" on utilise la propriété.

- Largeur : width avec les valeurs auto, valeur fixe, ou %
- Hauteur : height avec les valeurs auto, valeur fixe, ou %

Exemple 1 : Dans ce premier exemple 2 boîtes : une div et une span avec un style identique pour les 2. Une largeur et une hauteur **fixe**.

Dans le visuel on s'aperçoit que la propriété **width** et **height** n'ont pas d'effet sur la boîte 1b inline (ici) ni même et c'est logique le **text-align**, au contraire d'une balise bloc.

Conclusion : pour ce qui est d'élément de type en ligne, il n'est pas possible de modifier ni sa hauteur ni sa largeur.

La boîte 1a prend la largeur demandée mais reste un bloc comme le montre l'inspecteur d'élément. Il n'est donc pas possible d'ajouter une autre boîte à côté.

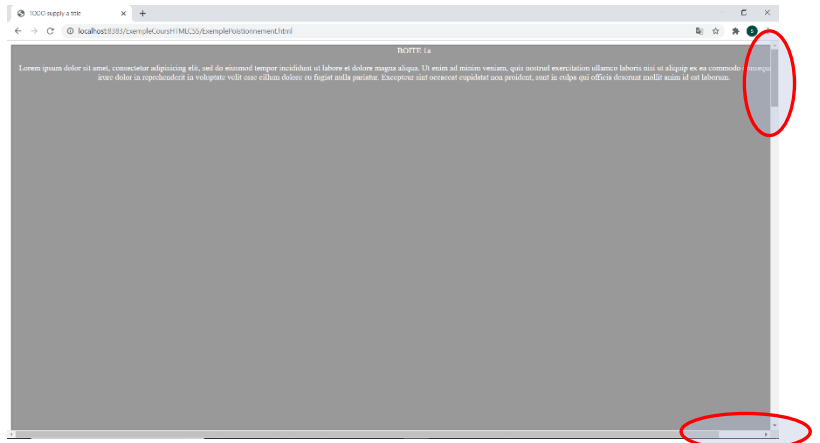
<pre>#boite1{ width: 600px; height:50px; } .bordure{ border: solid 2px #666666; background-color: #999999; color:white; text-align: center; }</pre>	<pre><div id="boite1" class="bordure"> BOITE 1a </div> BOITE 1b </pre>

Exemple 2 : c'est le même exemple que précédemment mais cette fois-ci avec beaucoup plus de contenu. Comme la boîte 1a est limitée en hauteur et en largeur le contenu n'est pas complètement affiché. Alors que dans la boîte 1b, la boîte s'adapte en largeur et en hauteur à son contenu.

<pre>#boite1{ width: 600px; height:50px; } .bordure{ border: solid 2px #666666; background-color: #999999; color:white; text-align: center; }</pre>	<pre><div id="boite1" class="bordure"> BOITE 1a <p> Lorem ipsum dolor ... sur 5 lignes ... non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. </p> </div> BOITE 1b Lorem ipsum dolor ... sur 5 lignes ... non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. </pre>

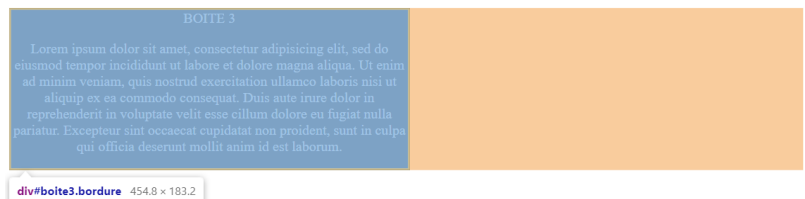
Exemple 3 : les mêmes boîtes avec une largeur beaucoup plus grande que la résolution de l'écran. Comme la boîte est plus grande que l'écran on note les scroll Bar horizontal et vertical, ce que l'on cherche à éviter généralement.

```
#boite2{
  width: 1600px;
  height:5000px;
}
```



Exemple 4 : en général on souhaite des boîtes qui s'adapte à la configuration de l'écran. Le plus simple pour cela est d'utiliser une valeur en %. Par défaut la largeur et la hauteur est en **auto** comme pour l'exemple de la **boite3**. Contrairement aux exemple précédent, cette boîte prend 50% de la largeur de l'écran, et s'adapte en hauteur **automatiquement** à son contenu comme le montre l'inspection.

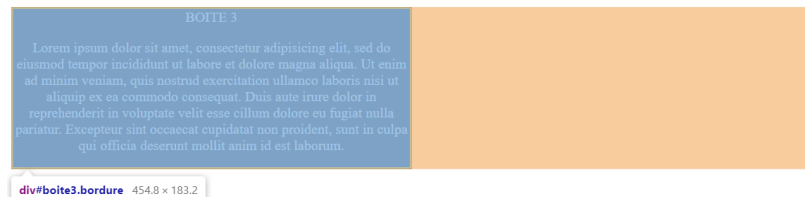
```
#boite3{
  width: 50%;
  height:auto; //pas
obligatoire puisque valeur par
défaut
}
```



Notons que si la hauteur est fixe (par exemple `height: 50px;`) comme dans l'exemple précédent le contenu n'aurait pas été complètement affiché puisqu'avec une hauteur fixe pas d'adaptation.

Exemple 5 : boîte qui s'adapte en hauteur. Nous venons de le voir, la hauteur d'un élément de type "bloc" s'adapte à son contenu (valeur "auto"). Nous reprenons ici notre **boite3** et nous lui appliquons une hauteur de 50%. Une valeur en pourcentage permet « théoriquement » de fixer la hauteur de la boîte **en fonction de son parent**.

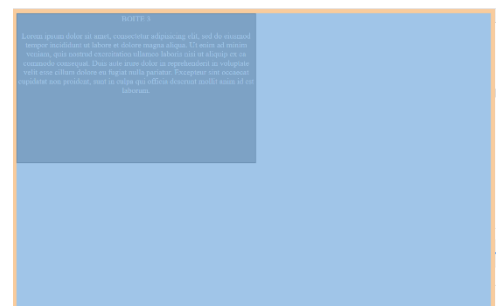
```
#boite3{
  width: 50%;
  height: 50%;
}
```



On note que la propriété `height: 50%;` n'a eu aucun effet. Pour quelle raison ? La hauteur de la balise `div` fera 50 % de la balise le contenant (ou balise parente) à savoir la balise `body` qui est une balise bloc pour laquelle aucune hauteur n'a été définie. Pour appliquer 50% de la hauteur du parent faut-il encore que le parent est une hauteur !!

Exemple 6 : ici l'élément `body` fait 600px de hauteur. La balise `<div>` aura donc une hauteur de 50% la hauteur de son parent, soit 300px.

```
body{
  height: 600px;
}
```

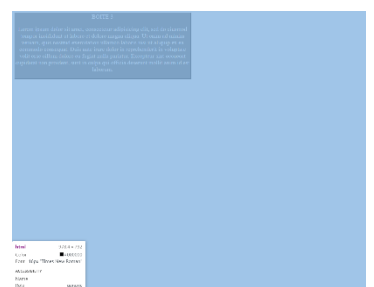


Exemple 7 : dans cet exemple l'élément `html` a une hauteur de 100% , et l'élément `body` n'a pas de hauteur spécifiée. Comme le montre le visuel la boîte `<div>` a une hauteur qui s'adaptera à son contenu, bien que le document prenne 100% en hauteur

```
html{
  height: 100%;
}
```

La raison est que la propriété `height` est une propriété qui est non hérité. Donc pour que notre boîte prenne 50% de l'écran il faut appliquer 100% sur `html` et sur `body`. La solution :

```
html,body{
  height: 100%; }
```



Exemple 8 : pour terminer sur les dimensions d'une boîte rappelons les propriétés déjà étudiées avec les images que sont **max-height**, **min-height**, **max-width**, **min-width** qui permettent de définir les hauteur et une largeur minimum et maximum. Dans le visuel on a la version 1 avec les dimensions minimums ou le contenu n'est pas complètement affiché, et la version 2, la même boîte avec les dimensions maximums et ou on peut lire tous le contenu.

```
#boite4{
  width: 50%;
  min-width: 100px;
  max-width : 300px;
  height:50%;
  min-height : 100px;
  max-height: 300px;
  background-color: #999999;
  color:white;
  text-align: center;
}
```



LES MARGES EXTÉRIEURES

Les propriétés permettant de modifier les marges extérieures sont : **margin-top**, **margin-right**, **margin-bottom**, **margin-left**, et **margin** la propriété raccourcie qui permet de modifier 4 propriétés en 1 fois (dans le sens des aiguilles d'une montre et en commençant par top).

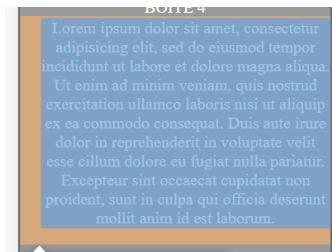
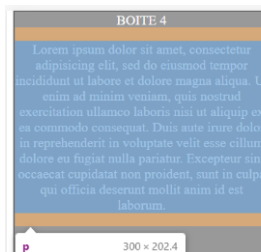
Les marges créent un espace supplémentaire à l'extérieur de l'élément HTML (après la bordure) par rapport au parent ou aux autres boîtes. Sans marge les éléments inline ou même bloc se collent les uns aux autres ce qui rend le visuel souvent désagréable.

Comme pour **width** et **height** les valeurs sont auto, fixe, ou %

Les marges haute et basse n'ont aucun effet sur les éléments en ligne (inline) qui ne sont pas remplacés (par exemple les `` ou `<code>`).

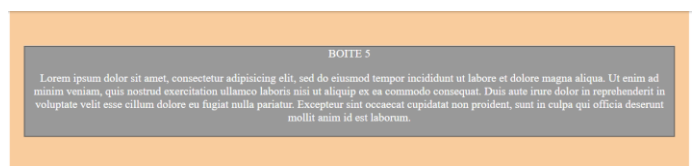
Exemple 1 : de nombreuses balises html ont dans leur style par défaut une marge. C'est les cas de la balise body qui ajoute une marge de 8px ou bien la balise `<p>` qui ajoute une marge de 16px en haut et en bas. Cet exemple montre comment modifier cette marge. Le visuel montre sans la modification et avec.

```
body{
  margin:0px;
}
p{
  margin-left: 20px;
  margin-top: 2px;
}
```



Exemple 2 : autre exemple avec la propriété raccourcie. Dans cet exemple nous avons une boîte centrée sur l'écran étant donné que la boîte5 a une largeur auto.

```
#boite5{
  margin: 50px 20px 50px 20px;
}
```



Exemple 3 : lorsque notre boîte a une largeur fixe ou en % pour centrer celle-ci horizontalement il faut appliquer une marge à gauche et à droite. Supposons une boîte de 50% en largeur. Pour la centrer il faut ajouter une marge à gauche et à droite de 25%. Cette solution nécessite des calculs. Si la boîte fait 80% de la largeur il faut appliquer alors une marge de 10% à gauche et à droite. Pour éviter les calculs il suffit de mettre une marge gauche et droite en **auto**. Avec cette solution quel que soit la largeur de la boîte elle sera toujours centrée.

Note importante : lorsqu'une marge est exprimé en pourcentage le calcul se fait relativement à la largeur du bloc englobant.

BOITE 6 Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.


```
#boite6{
  margin-left: 25% /*ou plus simplement */ auto;
  margin-right : 25% /*ou plus simplement */ auto;
  width: 50%;
}
```

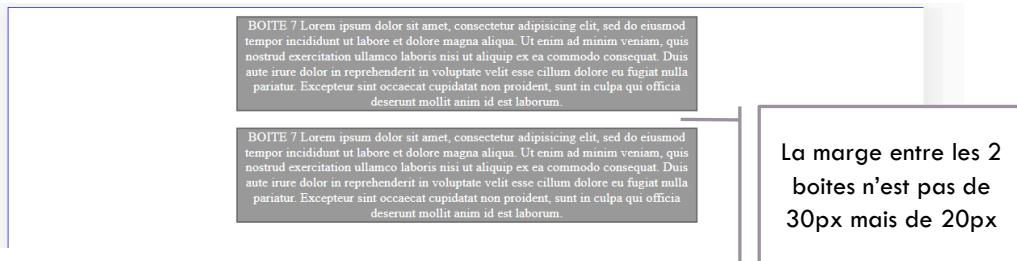
Exemple 4 : on pourrait vouloir faire la même chose mais en centrant notre boîte sur la verticale également. Cela ne fonctionnera pas puisque la marge calculée en top et en bottom se fera sur la largeur du bloc englobant et non sur sa hauteur comme indiqué dans la note. Le résultat obtenu ne sera pas satisfaisant comme on peut le constater sur le visuel. Nous verrons plus loin dans ce cours comment réaliser un centrage vertical

```
#boite6{
  margin-left: 25% ;
  margin-right : 25% ;
  width: 50%;
  margin-top: 25% ; //25% de la largeur
  margin-bottom : 25% ; //et pas 25% de la hauteur
  height: 50%;
}
```



Exemple 5 : ici on a 2 boîtes centrées horizontalement avec une largeur de 50%, et des marges en **top** de 10px et en **bottom** de 20px. Dans le visuel proposé on s'aperçoit que les marges ne se cumulent pas mais fusionnent en gardant la plus grande marge.

```
#boite7{
  margin-left: auto ;
  margin-right : auto ;
  width: 50%;
  margin-top: 10px ;
  margin-bottom : 20px;
}
```



Exemple 6 : autre particularité la possibilité de mettre des marges négatives. Cela peut être pratique de jouer sur les marges pour parfaire le positionnement.

LES MARGES INTÉRIEURES

Les propriétés permettant de modifier les marges intérieures sont : **padding-top, padding-right, padding-bottom, padding-left, et padding** qui permet de modifier 4 propriétés en 1 fois (dans le sens des aiguilles d'une montre et en commençant par top).

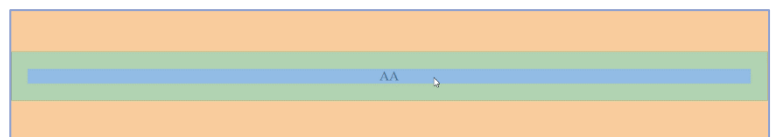
Les marges intérieures ajoute un espacement entre le contenu d'une boîte et sa bordure.

Contrairement au marge extérieur Les valeurs négatives ne sont pas autorisées. Mise à part cette différence les principes sont les même que les marges extérieures.

Comme pour les « margin », lorsque « padding » est exprimé en % le calcul se faite relativement à la largeur du bloc englobant.

Exemple :

```
div{
  border: 1px solid gray;
  background-color: rgb(215,230,245);
  text-align: center;
  margin: 50px 0px 50px 0px;
  padding : 20px;
  /* ici les 4 marges son modifies */
}
```



LES BORDURES

L'ensemble des propriétés qui permettent de modifier l'aspect des bordures des éléments HTML est :

Border, border-width, border-style, border-color, border-image. Toutes ses propriétés sont ensuite modifiables avec les déclinaisons

- **border-top, border-top-style, border-top-width, border-top-color, border-top-image**
- **border-right, border-right-style, border-right-width, border-right-color, border-right-image**
- **border-bottom, border-bottom-style, border-bottom-width, border-bottom-color, border-bottom-image**
- **border-left, border-left-style, border-left-width, border-left-color, border-left-image**

Si l'on utilise la propriété **border-width** pour modifier la taille de la bordure, elle sera appliquée aux 4 côtés. Si l'on souhaite modifier l'aspect d'un côté il faudra utiliser **border-top** par exemple.

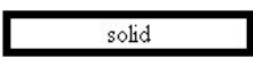
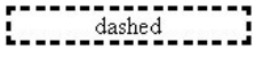
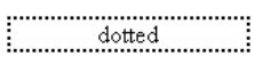
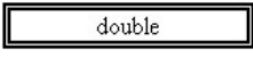
Ces propriétés ont déjà été étudié pour les tableaux HTML. Revenons dessus plus en détail.

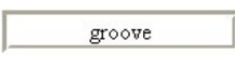
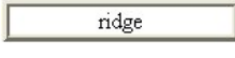
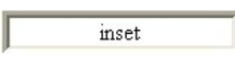

LA TAILLE, LE STYLE ET LA COULEUR DE LA BORDURE

Une bordure n'est affichée que si une épaisseur (**border-width**) à la bordure est définie ainsi qu'un style (**border-style**).

Si un style de bordure est défini, mais pas de taille la taille par défaut sera de 1px; Ne figure pas dans le tableau le style 'none' pour absence de bordure.

Inversement si une taille est définie, mais pas de style aucune bordure n'est affichée.

solid	
dashed	
dotted	
double	

groove	
ridge	
inset	
outset	

Si aucune couleur (**border-color**) n'est définie, la couleur "black" s'applique par défaut.

Exemple :

```
border-style: dashed;
border-width: 2px;
border-color: brown;
```



```
/* style précédent peut être écrit de la façon suivante quel que soit l'ordre d'écriture des valeurs. */
border : dashed 2px brown;
border : 2px brown dashed;
border : 2px dashed brown;
```

Comme nous l'avons expliqué, il est possible de fixer le style, l'épaisseur et la couleur pour chaque côté individuellement.

```
border-style: dashed solid none double;
border-width: 2px 4px 0px 10px;
border-color: brown blue red green;
```

```
/* équivaut à (mais plus long à écrire)*/
border-top-style: dashed ;
border-right-style : solid;
border-bottom-style : none;
border-left-style : double;
```



Le CSS propose également 3 largeurs prédéfinies :

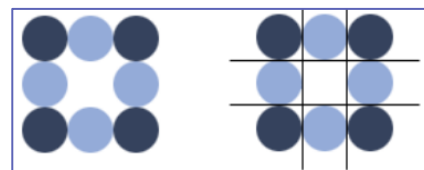
- **thin** : fin.
- **medium** : moyen.
- **thick** : épais.

LES BORDURES PERSONNALISÉES AVEC DES IMAGES

La propriété CSS `border-image` permet de personnaliser ses bordures avec ses propres images. Il faut pour cela une image rectangulaire

On divise l'image en 8 parties égales avec les dimensions a,b,c, et d. Si l'image est carrée, une seule dimension n'est en réalité utile.

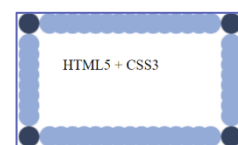
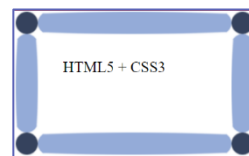
Les 8 parties sont décomposées ainsi : Le coin en haut à gauche, la partie centrale en haut, le coin en haut à droite, etc ...



Exemple :

```
.bordure{
  border: 30px solid;
  /* cette propriété est indispensable. Une bordure image reste une bordure */
  border-image: url("../images/bordure.png") 53 stretch;
  /* ici chaque partie fait 53px de côté, stretch permet l'étirement */
}

.bordure{
  border: 30px solid;
  border-image: url("../images/bordure.png") 53 round;
  /* round permet de reproduire les images */
}
```



LES BORDURES ARRONDIES

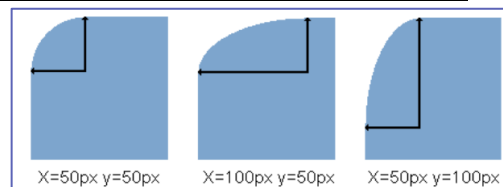
Les bords arrondis des quatre extrémités sont introduits par la propriété :

```
border-radius: x y; /* où x et y sont une valeur ou un pourcentage */
```

Les valeurs de x et y déterminent les rayons horizontaux et verticaux d'un quart d'ellipse, ce qui induira la courbure de l'angle. Une seule valeur peut être indiquée. Dans ce cas, la valeur de x est égale à la valeur de y.

Il est possible de modifier individuellement les bords avec les propriétés :

- `border-top-right-radius`
- `border-bottom-right-radius`
- `border-bottom-left-radius`
- `border-top-left-radius`



Exemple :

```
.bordarrondi{
  border : 10px solid gray;
  border-radius: 8px 12px;
}
```

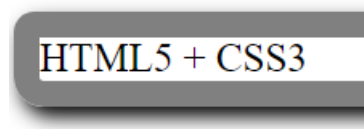
HTML5 + CSS3

Les bords arrondis peuvent s'appliquer à un tableau (balise `<table>`) ou bien encore à une image (``).

LES EFFETS D'OMBRES

La propriété **box-shadow** permet de faire des effets d'ombres sur votre boîte. Elle possède les propriétés "x y z couleur" où :

- x est le déport de l'ombre vers la droite.
- y est le déport de l'ombre vers le bas.
- z est l'intensité du dégradé ou du flou (facultatif, par défaut 0).
- Couleur est la couleur de l'ombre.



Exemple :

```
box-shadow: 1px 1px 2px black;
```

GERER LA DIMENSION TOTAL D'UNE BOITE

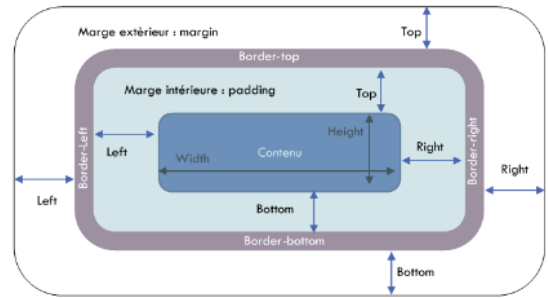
Rappelons ce qu'est une boîte :

Un contenu + marge intérieure + bordure + marge extérieure.

Au final la place prise en largeur de la boîte est la somme suivante :

$\text{margin-left} + \text{border-left-width} + \text{padding-left} + \text{width} + \text{padding-right} + \text{border-right-width} + \text{margin-right}$.

Même chose en hauteur.



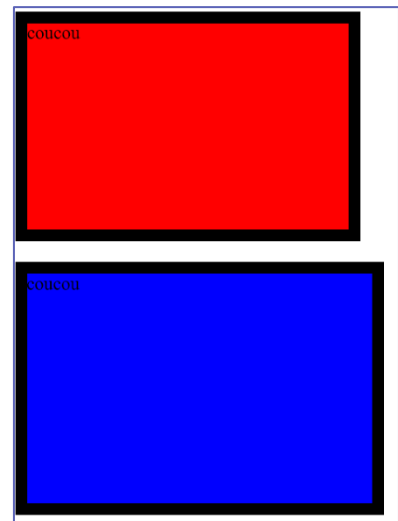
Le CSS introduit la propriété **box-sizing** permettant de savoir comment est calculé cette largeur avec 2 valeurs possibles :

- **content-box** : indique que le calcul de la largeur et de la hauteur d'affichage de la boîte se fait avec les valeurs des propriétés width, height, padding et border. C'est la valeur par défaut
- **border-box** : indique que le calcul de la largeur et de la hauteur d'affichage de la boîte se fait uniquement avec les valeurs des propriétés width et height, sans tenir compte de la bordure (border), ni le remplissage (padding).

Exemple 1 : Ici 2 boîtes avec des propriétés identique (largeur, hauteur bordure) mais en fonction de la méthode de calcul le visuel sera différent. En **rouge** le calcul **border-box**, et en **bleu** le calcul par défaut **content-box**

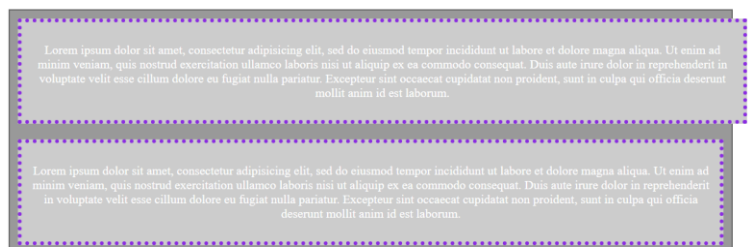
```
.div1{
  width: 300px;
  height: 200px;
  background-color: red;
  border: 10px solid black;
  box-sizing: border-box;
/* ➔ ici le contenu aura une largeur de 300-10-10 : 280px
   La hauteur de 200-10-10
*/
}
.div2{
  width: 300px;
  height: 200px;
  background-color: blue;
  border: 10px solid black;
  box-sizing: content-box;
/* ➔ ici le contenu aura une largeur de 300px, la hauteur de 200px
   La largeur totale sera de 300 +10 +10 : 320px, la hauteur de 220px
*/
} ...

<div class="div1"> coucou </div> <br/>
<div class="div2"> coucou </div>
```



Exemple 2 : cette propriété s'avère très pratique pour éviter que le contenu d'une boîte dépasse son conteneur parent. Ici nous avons un conteneur parent qui prend 60% de la largeur. Dans ce conteneur 2 boîtes identiques qui font 100% du parent mais diffèrent sur la méthode de calcul. L'une dépasse de son parent, l'autre s'adapte.

```
#conteneur1{
  margin: auto;
  width: 60%;
  padding: 10px;
  margin: 10px auto;
}
#boite9{
  border: dotted 5px blueviolet;
  padding: 10px;
  width: 100%;
  box-sizing: content-box;
  margin-bottom: 20px;
  background-color: #cccccc;
}
#boite10{
  border: dotted 5px blueviolet;
  padding: 10px;
  width: 100%;
  box-sizing: border-box;
  background-color: #cccccc;
}
```



LA COULEUR DES BOITES

La propriété `background` permet de fixer la couleur de fond de la boîte en utilisant une des façons dans le chapitre couleur. Elle contient différentes sous-propriétés qui sont : **background-color**, **background-image**, **background-repeat**, **background-attachment**, **background-position**. Ces propriétés ont déjà été étudiés avec les images

Exemple :

```
background : rgb(215,230,245);
/* est identique à */
background-color : rgb(215,230,245);
```

Il est possible de faire des dégradés de couleur avec la valeur **linear-gradient** qui définit le point de départ du dégradé, la couleur de début et la couleur de fin. Le point de départ peut s'exprimer par rapport à un angle

Exemple:



```
background : linear-gradient(to left bottom, white, black);
background : linear-gradient(45deg, blue, red);
```

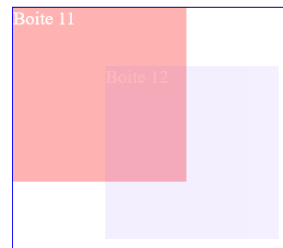
Il est possible de faire des dégradés de couleur circulaire avec la valeur **radial-gradient** qui définit la position de départ (top, right, bottom, left, center), la Forme (circle ou ellipse), couleur début couleur fin.

```
background: radial-gradient( circle at center, white, black);
```

Il est possible de gérer l'opacité des boîtes avec la propriété **opacity** (valeur entre 0 pour transparent et 1 pour opaque)

Dans l'exemple qui suit, 2 balises `<div>` sont superposées en jouant sur les marges. Les 2 boîtes ont une opacité à 0.5.

```
#boite11{
  width: 150px;
  height: 150px;
  background-color: #ff6666;
  color : white;
  opacity: 0.5;
}
#boite12{
  width: 150px;
  height: 150px;
  background-color: #9966ff;
  color : white;
  opacity: 0.1;
  margin-left: 80px;
  margin-top: -100px;
}
```



Il est également possible d'appliquer une image en fond de boîte.

Exemple :

```
background-image: url("../images/img1.png");
```

LE POSITIONNEMENT DES BOITES

Dans cette partie nous allons voir comment positionner nos boîtes. La propriété CSS permettant de gérer le positionnement est **position**. Le positionnement est fixé par défaut à la valeur **static**. Avec cette valeur les éléments HTML sont placés dans le « **flux normal** » c'est-à-dire apparaissent là où ils doivent naturellement apparaître et cela selon son type.

Ainsi un élément bloc se place naturellement sur en dessous d'un autre bloc, un élément inline à côté d'un autre élément inline.

Avec ce positionnement par défaut les propriétés **top**, **bottom**, **left** et **right** qui permettent de définir la position n'auront aucun effet. C'est ce que montre l'**exemple 1** suivant. Le bloc2 se positionne naturellement en dessous du bloc1 à gauche, même si ici la largeur de chacune est redéfinie. La propriété `top` et `left` n'a eu aucun effet sur la position.

```
#bloc1{ width: 100px;
        background-color:red;
        border : solid black 2px;
        top:30px;
        left: 100px;
}
#bloc2 { width: 200px;
        background-color: blue;
        border : solid black 2px;
```

TODO write
content

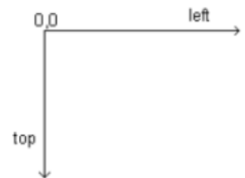
TODO write content

En plus de static, position accepte 4 autres valeurs.

POSITIONNEMENT RELATIF :

C'est le positionnement d'un élément **relativement** à sa position normale de départ. L'élément est placé dans le **flux normal**, puis de décaler horizontalement et/ou verticalement. La position est définie par les coordonnées (x,y) où :

- x est la distance par rapport au bord gauche de l'élément parent ou de la fenêtre du navigateur (axe horizontal). Ainsi, **left** détermine la distance entre la gauche de l'élément et la gauche de la page et **right**, la distance entre la droite de l'élément et la droite de la page.
- y est la distance par rapport au bord supérieur de l'élément parent ou de la fenêtre du navigateur (axe vertical). Ainsi **top** détermine la distance entre le bord supérieur de l'élément et le bord supérieur de la page et **bottom**, la distance entre le bord inférieur de l'élément et le bord inférieur de la page.
- Dans la pratique on utilise le plus souvent les propriétés **left** et **top**.
- Les valeurs de top, left, right et bottom sont des valeurs ou % et peuvent être négatives.

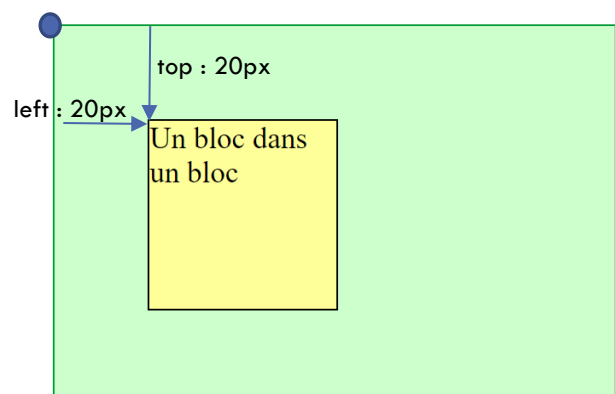


Exemple 1 : Nous avons ici une div de taille fixe (bloc4) dans une div également de taille fixe (bloc3). Le bloc 4 est positionné de façon « **relative** » à sa position normal (repéré par dans la figure)

```
<div id="bloc3">
  <div id="bloc4">Un bloc dans un bloc</div>
</div>
```

```
#bloc3{
  width: 300px;
  height: 200px;
  border:solid 1px #009933;
  background-color: #ccffcc;
}
#bloc4{
  position: relative;
  width: 100px;
  height: 100px;
  border:solid 1px black;
  background-color: #ffff99;
  left :20px;
  top :20px;
}
```

exemple 2



Notons ici que l'utilisation des marges (margin-top : 50px et margin-left : 50px) auraient le même effet ici. Il n'y a pas de règles précises quant à l'utilisation d'une technique par rapport à une autre.

Exemple 2 : Dans cet exemple, la balise `` est placée dans le flux normal avec un décalage de 5px par rapport au "bottom" de la balise parente `<p>`

```
.jaune {
  position: relative;
  bottom: 5px;
  background-color: #ffff00;
}
```

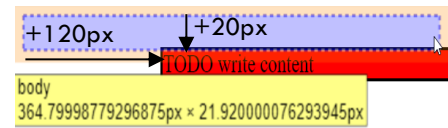
Lorem **boîte en position relative** ipsum dolor.

```
<p> Lorem <span class="jaune"> boîte en position relative </span> ipsum dolor. </p>
```

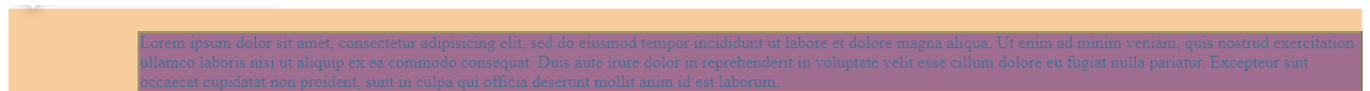
Exemple 3 : Attention : modifier la position ne modifie ni la largeur ni la hauteur de votre bloc.

Dans l'exemple suivant, la largeur du bloc est celle de la page. En ajoutant un décalage à gauche et en haut, c'est le bloc tout entier qui est décalé. Pour permettre de visualiser l'ensemble de la boîte, le navigateur ajoute ici une scroll barre horizontale.

```
#bloc1{
  background-color:red;
  border : solid black 2px;
  position: relative;
  top: 20px;
  left: 120px;
}
<div id="bloc1">TODO write content</div>
```



Si on fait la même chose avec *margin-top :20px* et *margin-Left: 120px*, la scroll barre horizontale disparaît. La largeur du bloc s'adapte à la largeur de la page comme illustré.



POSITIONNEMENT ABSOLU :

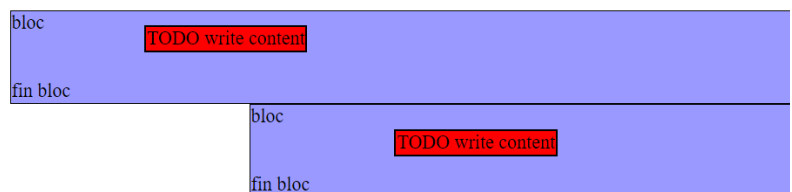
Le positionnement absolu crée un élément indépendant du reste du document. Les éléments définis en position absolue sont retirés du flux normal et se positionnent à l'emplacement exact défini par le concepteur **par rapport à son plus proche parent positionné (relative ou absolue)** ou par rapport au début de la page (<body>) si aucun parent proche positionné n'est trouvé.

Exemple 1 : le premier bloc parent est dans positionné en **static** dans le flux normal. Le premier bloc de classe « bloc7 » est positionné en **absolute**. Sa position est calculée par rapport au début de la page (<body>) puisque son parent n'est pas positionné (static).

Le deuxième bloc est positionné en « **relative** » avec un décalage à gauche de 120px. Le second bloc7 est toujours positionné en **absolute** mais cette fois ci sa position n'est plus calculée par rapport au <body>, mais par rapport à son plus proche parent positionné la div conteneur.

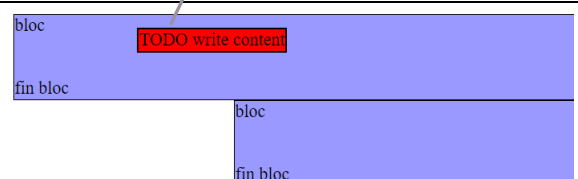
```
HTML
<div class="conteneur">
  <div class="bloc7">TODO write content</div>
  bloc<br/><br/><br/> fin bloc
</div>
<div class="conteneur" style="position: relative; left:200px; ">
  <div class="bloc7">TODO write content</div>
  bloc<br/><br/><br/> fin bloc
</div>

CSS
.bloc7{
  background-color:red;
  border : solid black 2px;
  position: absolute;
  top: 20px;
  left: 120px;
  width: auto;
  height: auto;
}
.conteneur{
  background-color: #9999ff;
  border: solid 1px black;
}
```



Les 2 bloc7 sont superposés

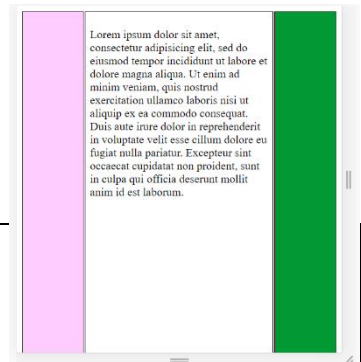
Remarque : On aurait pu décaler le second conteneur à l'aide de *margin-left : 200px* et laisser le bloc en **static**. Dans ce cas le plus proche parent du 2nd bloc7 redevient <body>. Sur le visuel on a l'impression que ce bloc7 vient de disparaître mais en réalité il est venu se superposer au 1^{er} bloc7.



Dernière remarque : étant donné que le bloc7 est retiré du flux normal, à l'inspecteur d'élément on remarque que son comportement n'est plus celui d'un bloc mais celui d'une balise en ligne. En temps

normal un bloc prend toute la largeur de la page. Ici on remarque (malgré **width à auto**) que le bloc prend uniquement la place nécessaire pour afficher le contenu.

Exemple 2 : Le positionnement absolu peut-être une solution pour positionner des conteneurs côtes à côtes. Dans le code HTML la partie centrale est positionné dans le flux normal, centré sur la largeur. C'est donc un bloc, impossible d'y accoler un autre bloc. Sauf si l'élément à droite et à gauche est retiré du flux normal. C'est le cas dans cet exemple. La partie gauche et droite est en position **absolute**.



HTML <pre><div class="auCentre"> <p> Lorem ipsum [...] laborum. </p> </div> <div class="aGauche"></div> <div class="aDroite"></div></pre> CSS <pre>.auCentre{ width: 60%; margin: auto; height: 600px; border: 1px solid black; box-sizing: border-box; padding: 6px; }</pre>	<pre>.aGauche{ width: 19%; position: absolute; background-color: #ffccff; height: 600px; border: 1px solid black; top: 8px; box-sizing: border-box; } .aDroite{ width: 19%; position: absolute; background-color: #009933; height: 600px; border: 1px solid black; top: 8px; right: 8px; box-sizing: border-box; }</pre>
---	--

LE POSITIONNEMENT FIXE

Comme le positionnement **absolute**, « **fixed** » crée aussi un élément **indépendant** dont on peut définir la position. La différence avec **absolute** est le calcul du positionnement qui se fait par rapport au début de la page. Ici l'élément reste fixe même lorsque l'on fait défiler le document. Ce positionnement est idéal pour les zones de menu qui restent fixes, quel que soit le document, ou bien encore un pied de page qui est visible, quelle que soit la taille du document.

Exemple : ici une div avec beaucoup de texte centré sur la largeur. Le bloc « fixe » reste fixe même quand on scroll la page.

<pre><div class="auCentre2"> <p>Lorem ipsum [. . .] laborum.</p> <p>Lorem ipsum [. . .] laborum.</p> <p>Lorem ipsum [. . .] laborum.</p> <p>Lorem ipsum [. . .] laborum.</p> <div class="fixe">Bloc fixe à 50px left 50px top du début de la page</div> </div></pre>	<pre>.fixe{ position: fixed; width: 100px; height: 100px; border:solid 1px black; background-color: #ffff99; left :50px; top :50px; }</pre>
<pre>.auCentre2{ width: 40%; margin: auto; border: 1px solid black; box-sizing: border-box; padding: 6px; }</pre>	

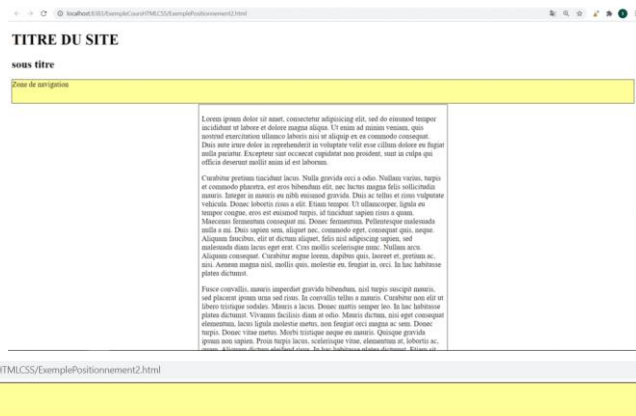
LE POSITIONNEMENT STICKY

Ce type de positionnement offre un mixte entre relative et **fixed**. Au départ l'élément est positionné en relative dans le flux normal. Puis en cas de défilement (scroll vertical), sa position va passer à **fixed** le moment choisi.

Dans l'exemple nous avons une zone de navigation (un menu). Au départ celui-ci est placé dans le flux normal en dessous d'un titre. Quand on va scroller et que la zone de navigation sera à 0px du haut de la page, cette zone va se fixer.

```
.sticky{
  width: auto;
  height: 50px;
  border:solid 1px black;
  background-color: #ffff99;
  left :50px;
  top :0px;
  position: sticky;
}
```

```
<h1>TITRE DU SITE</h1>
<h2>sous titre</h2>
<div class="sticky"> Zone de navigation </div>
<div class="auCentre2">
  <p>Lorem ipsum [ . . . ] laborum.</p>
  <p>Lorem ipsum [ . . . ] laborum.</p>
</div>
```



LA PROPRIÉTÉ FLOAT :

La propriété **float** retire un élément boîte du flux normal pour la placer le plus à droite (float:right) ou le plus gauche (float: left) possible dans son élément parent. Le texte et les autres éléments en ligne (inline) entoureront alors l'élément flottant. Il n'y a pas ici de recouvrement. Exemple :

```
.blocfloat{
  width: 300px;
  padding: 0;
  margin-right: 20px;
  float: left;
  border : solid 1px;
}
<div class="blocfloat"> contenu1</div>
<div class="blocfloat"> contenu2</div>
<div class="blocfloat"> contenu3</div>
```

Exemple avec une image : L'image est positionnée à droite elle n'est pas dans le flux normal, et le texte, quant à lui, s'affiche alors à gauche autour de l'image positionnée dans le flux normal. Si l'image avait été positionnée en absolu, le texte aurait recouvert l'image. Ce n'est pas le cas ici.

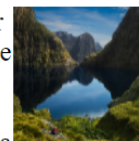
```
<div style="float: right;">
  
</div>
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed non risus. Suspendisse lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed, dolor. Cras elementum ultrices diam. Maecenas ligula massa, varius a, semper congue, euismod non, mi.</p>
```

La propriété **clear** permet d'annuler le flottement introduit par la propriété float. Les valeurs possibles sont right, left, both, none.

L'exemple précédent pourrait alors s'écrire :

```
<div style="float: right;">
  
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed non risus. Suspendisse lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed, dolor. Cras elementum ultrices diam. Maecenas ligula massa, varius a, semper congue, euismod non, mi.



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed non risus



Suspendisse lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed, dolor. Cras elementum ultrices diam. Maecenas ligula massa, varius a, semper congue, euismod non, mi.

```

    </div>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed non risus </p>
    <p style="clear: right">Suspendisse lectus tortor, dignissim sit amet, adipiscing nec, ultricies
    sed, dolor. Cras elementum ultrices diam. Maecenas ligula massa, varius a, semper congue, euismod non,
    mi. </p>

```

Autre exemple avec un menu : ici le contenu est en position flottante à gauche, le menu est dans le flux normal, mais avec une marge de 80% sur la gauche. Si le contenu prend une largeur de 100%, pour qu'il n'y ait pas de recouvrement, la zone de menu se placera en dessous de la zone de contenu.

```

.content {
  float: left;
  width: 70%;
  background-color: #ffecce;
  border: 1px solid #666666;
}
.menu {
  margin-left: 80%;
  background-color: #ffecce;
  border: 1px solid #666666;
  padding: 1em;
}
<div class="content"><br/><br/><br/><br/><br/></div>
<div class="menu"></div>

```



REDIMENSIONNEMENT

La propriété CSS3 **resize** permet de redimensionner un élément avec les valeurs : **none**, **horizontal**, **vertical** et **both** (les 2: Vertical et Horizontal).

Cette propriété CSS est souvent utilisée avec la propriété **overflow** qui détermine ce que le navigateur doit faire lorsqu'un élément est plus grand que l'élément parent qui le contient.

Elle utilise les valeurs :

- **visible** : le contenu qui dépasse n'est pas rogné
- **hidden** : le contenu qui dépasse est rogné
- **scroll** : le contenu qui dépasse est rogné, et ajout de scroll barre après redimensionnement.
- **Auto** : le contenu qui dépasse est rogné, et ajout de scroll barre verticale et horizontale supprimée après redimensionnement.

Michaelmas term lately over, and the Lord Chancellor sitting in Lincoln's Inn Hall. Implacable November weather. As much mud in the streets as if the waters had but newly retired from the face of the earth.

Michaelmas term lately over, and the Lord Chancellor sitting in Lincoln's Inn Hall. Implacable November weather. As much mud in the

Michaelmas term lately over, and the Lord Chancellor sitting in Lincoln's Inn Hall. Implacable November

verticale et horizontale toujours présente même

Exemple :

```

.boiteresize{
  resize:both;
  overflow:auto;
  width:270px; height:60px;
  border:1px solid silver;
}
<div class=" boiteresize ">Lorem ipsum dolor ... </div>

```

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed non risus. Suspendisse lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed, dolor. Cras elementum ultrices diam. Maecenas ligula massa,

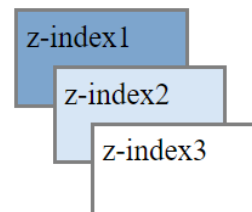
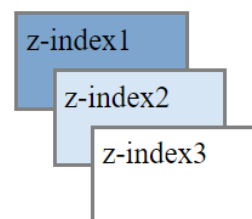
Z-INDEX

La propriété z-index ajoute un axe en profondeur permettant de positionner des éléments au-dessus ou en dessous d'un autre élément. Ainsi, l'élément avec comme propriété z-index: 2 apparaîtra avant ou au-dessus de l'élément ayant comme propriété z-index: 0. Cette propriété ne fonctionne qu'avec un positionnement absolu des éléments.

Exemple :

```
.bloc{
  width: 80px; height: 40px;
  padding: 4px;
  border: 2px solid gray;
}
#bloc1 {
  position: absolute;
  left: 20px; top: 20px;
  background-color: rgb(125,165,205);
  z-index: 1;
}
#bloc2 {
  position: absolute;
  left: 40px; top: 50px;
  background-color: rgb(215,230,245);
  z-index: 2;
}
#bloc3 {
  position: absolute;
  left: 60px; top: 80px;
  background-color: white;
  z-index: 3;
}

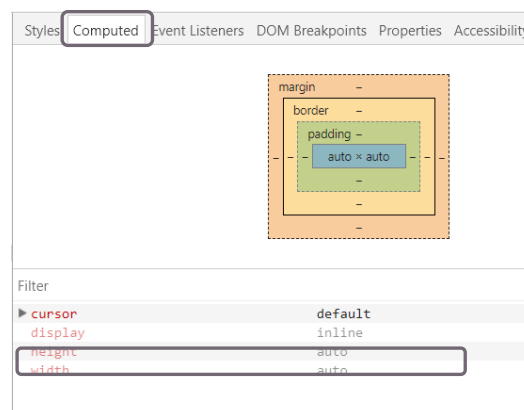
<div id="bloc1" class="bloc"> z-index1 </div>
<div id="bloc2" class="bloc"> z-index2 </div>
<div id="bloc3" class="bloc"> z-index3 </div>
```

**LA PROPRIÉTÉ DISPLAY**

Rappel : Chaque élément a une valeur **display** par défaut dépendant du type de l'élément. Les valeurs par défaut de la plupart des éléments sont **block** ou **inline**. Les navigateurs permettent via l'inspecteur d'élément d'obtenir cette information.

Exemple avec une balise `<label>` →

La propriété de style **display** permet de redéfinir un élément **inline** comme étant un élément **bloc** et inversement. Ainsi une balise `` qui est du type en ligne par défaut peut devenir une balise en **block** et inversement, une balise normalement en bloc peut devenir une balise **inline**.



Les valeurs possibles pour cette propriété sont :

inline, block, list-item, inline-block, table, inline-table, table-row-group, table-header-group, table-footer-group, table-row, table-column-group, table-column, table-cell, table-caption, none.

Exemple avec inline : transformer les éléments de liste (**list-item**) en type en **inline** : idéal pour un menu.

```
li {
  display: inline;
  border: 1px solid gray;
  background-color: rgb(215,230,245);
  text-align: center;
  margin-right: 3px;
  padding-right: 15px;
  padding-left: 15px;
} ...
<ul>
  <li> Accueil </li>
  <li> BTS SIO </li>
  <li> BTS SN </li>
  <li> BTS AM </li>
</ul>
```



Exemple 1 avec inline-block : les éléments de type **inline** se placent les uns à côté des autres et leur largeur s'adapte à son contenu. Dans l'exemple précédent, les éléments du menu semblent être tous à la même dimension, mais en réalité non. L'ajout d'une propriété `width:100px` dans le CSS n'aura aucun effet.

Inline-block permet de spécifier un élément inline tout en préservant leurs capacités d'éléments block, tels que la possibilité de définir une largeur et une hauteur, des marges, etc...

Faisons les modifications suivantes :

```
li {
  display: inline-block;
  border: 1px solid gray;
  background-color: rgb(215,230,245);
  text-align: center;
  margin-right: 3px;
  padding-right: 15px;
  padding-left: 15px;
  width: 120px;
  height: 30px; }
```



Exemple 2 : soit le formulaire suivant :

```
<form name="my_form" action="#result" method="post">
  <p> <label for="nom">Nom</label> <input type="text" id="nom"> </p>
  <p> <label for="email">E-mail</label> <input type="email" id="email"> </p>
  <p> <label for="sujet">Sujet</label> <input type="text" id="sujet"> </p>
  <p> <label for="message">Message</label> <textarea id="message"
rows="5"></textarea> </p>
</form>
```

Nom

E-mail

Sujet

Message

Ici les labels (inline) entraînent des décalages. Pour y remédier, nous ajoutons une propriété `width`, sans effet puisque les éléments **inline** sont prévus pour s'ajuster à leurs contenus. Modifions le CSS:

```
label{
  display: inline-block;
  width: 70px;
  vertical-align: top;
}
```

Nom

E-mail

Sujet

Message

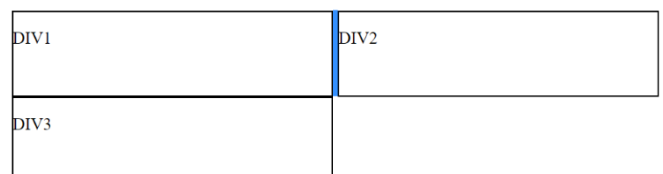
Notons au passage qu'une balise `<input>` est de type `inline-block`.

Exemple 3 : 3 div côte à côte.

```
.conteneur {
  width: 900px;
}
.conteneur div {
  display: inline-block;
  width: 300px;
}
...
<div class="conteneur">
  <div><p>DIV1</p></div>
  <div><p>DIV2</p></div>
  <div><p>DIV3</p></div>
</div>
```



Dans cet exemple les 3 div font 300px de large soit une largeur totale de 900px, ce qui correspond à la largeur du conteneur. Les 3 div devraient être côte à côte. Pourtant ce n'est pas le cas. On obtient le résultat suivant. On note en **bleu** un white-space (espace blanc) qui correspond au retour chariot. Ce caractère (de 2 ou 4px) fausse l'affichage.



Une solution consiste à ajouter des commentaires sur la fin de code pour éviter ce caractère blanc

```
<div class="conteneur">
  <div><p>DIV1</p></div><!--
  --><div><p>DIV2</p></div><!--
  --><div><p>DIV3</p></div><!--
--></div>
```

Si on souhaite disposer d'un espace entre les balises div une solution consiste à modifier la largeur, et ajouter une marge à droite.

```
.conteneur div {
  display: inline-block;
  width: 299px;
  height: 80px;
  border: solid 1px;
  box-sizing: border-box;
  margin-right: 1px;
}
```



Exemple 4: Dans cet exemple, nous utilisons un conteneur en affichage table, de 980 pixels de large et centrée dans la fenêtre du navigateur. Les deux colonnes de la mise en page sont deux cellules de tableau (display: table-cell) qui occupent respectivement 30 % et 70 % de la largeur disponible. Chacune des colonnes a un fond coloré qui est parfaitement affiché, quelle que soit la quantité de texte s'y trouvant. Contrairement à inline-block aucun white-space n'est ajouté entre les 2 balises div.

```
#conteneur {
  display: table;
  width: 980px;
  margin: 0 auto;
}
#boite-1, #boite-2 {
  display: table-cell;
  padding: 5px;
}
#boite-1 {
  background-color: #eee;
  width: 30%;
}
#boite-2 {
  background-color: #bbb;
  width: 70%;
}
p, h3 {
  margin: 0;
}
```

Sollicitudin Euismod
Donec id elit non mi porta...
Quam Cras
Nullam id dolor id nibh...

Ligula Fermentum
Cras mattis consectetur purus...

```
<div id="conteneur">
  <div id="boite-1">
    <h3>Sollicitudin Euismod</h3>
    <p>Donec id elit non mi porta...</p>
    <h3>Quam Cras</h3>
    <p>Nullam id dolor id nibh...</p>
  </div>
  <div id="boite-2">
    <h3>Ligula Fermentum</h3>
    <p>Cras mattis consectetur purus...</p>
  </div>
</div>
```