Introduction à la mise en page

Désormais que nous avons vu le modèle de boîte CSS et l'agencement des pages HTML avec les balises structurantes, nous pouvons nous atteler à la mise en page proprement dite. Pour ce faire, nous devons d'abord comprendre comment se comportent et s'organisent les éléments par défaut : c'est ce qu'on appelle le flux naturel. Les différentes méthodes de positionnement nous permettront de modifier le flux naturel afin de réorganiser les éléments de notre page, mais cela n'aura d'intérêt que si les éléments ne tombent pas naturellement à l'endroit où l'on voudrait qu'ils apparaissent.

I. Mise en page et positionnement

A. Le flux naturel des éléments

Regardons ce que donne la succession de deux éléments de bloc qui ne sont pas positionnés et restent donc dans le flux normal.

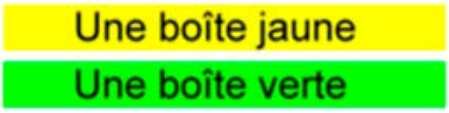

Fig. 1

Pour rappel, s'agissant des éléments de type **inline**, ils se succèdent de gauche à droite.

Fig. 2

Les principaux éléments créant des boîtes bloc sont :

- l'élément : **div** ;
- les titres **h1, h2, h3, h4, h5, h6** ;
- le paragraphe **p** ;
- les listes et éléments de liste **ul, ol, li, dl, dd** ;
- le bloc de citation **blockquote** ;
- le texte préformaté **pre** ;
- l'adresse **address** ;
- les balises structurantes **section, header, footer, aside, article, nav**, etc. Sachez, que pour les vieux navigateurs, on indique « display:bloc » dans le css pour les forcer à les considérer comme éléments de bloc (chose dispensable si on utilise un fichier reset.css ou modernize.css).

Dans un premier temps, il est pertinent de savoir de quelle manière les éléments de blocs vont se placer dans la page, c'est-à-dire par rapport à quel point, et quel est le système de coordonnées associé.

Le premier système de coordonnées est celui de la page web (balise <body>).

Il est appelé système de coordonnées globales.

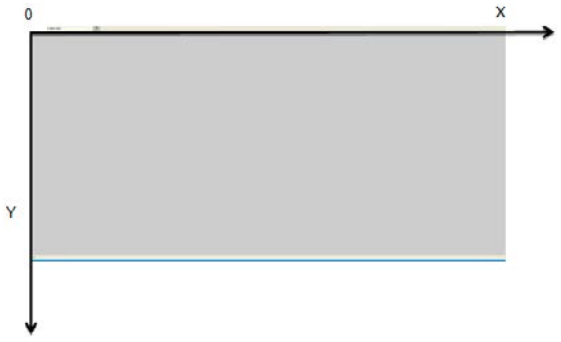

Fig. 3

L'axe horizontal appelé abscisses (X) est orienté de gauche à droite. L'axe vertical appelé ordonnées (Y) est orienté de haut en bas.

De plus, chaque élément en bloc a son propre système de coordonnées orienté de la même façon que le système global.

Il est appelé système de coordonnées locales.

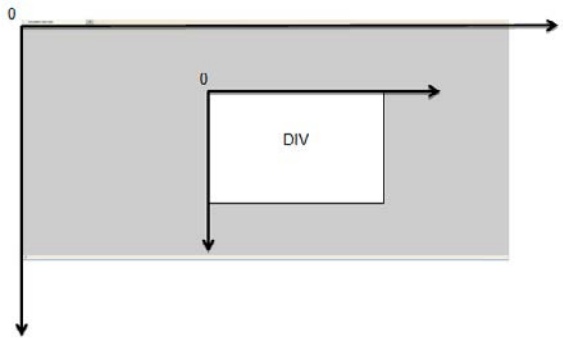

Fig. 4

En fait, c'est le point zéro de l'élément de bloc (ici **div**) qui se positionne par rapport au point zéro de son conteneur.

Par défaut, les positions des deux points 0, de l'élément div et de son conteneur sont identiques, sauf bien entendu si le conteneur possède une marge intérieure, ce qui est le cas par défaut sur la plupart des navigateurs pour la balise **body, p, ul, li** et les balises de titre (**h1, h2**) par exemple.

Par défaut, si aucune règle CSS n'est appliquée à un élément de bloc placé après la balise body, nous aurons :

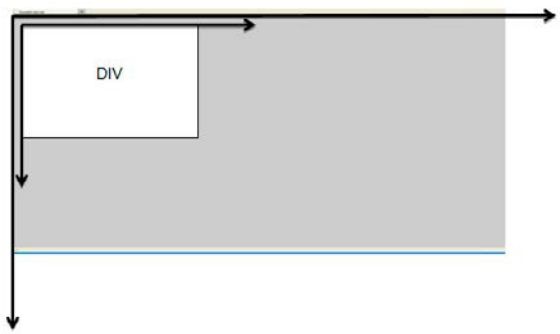

Fig. 5

Et si un élément de bloc contient un autre élément de bloc, la position des points zéro de chaque élément est exactement la même.

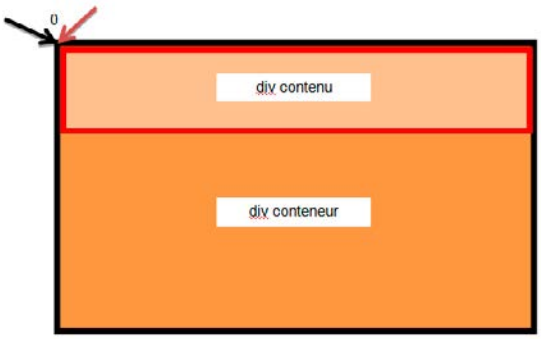

Fig. 6

Fichier HTML

```
<div id="conteneur">
    <div id="contenu"></div>
</div>
```

Fig. 7

En revanche, si plusieurs éléments de bloc se trouvent dans un élément de bloc, c'est uniquement le premier qui se positionne en haut à gauche, les suivants, rappelez-vous s'empilent les uns sous les autres.

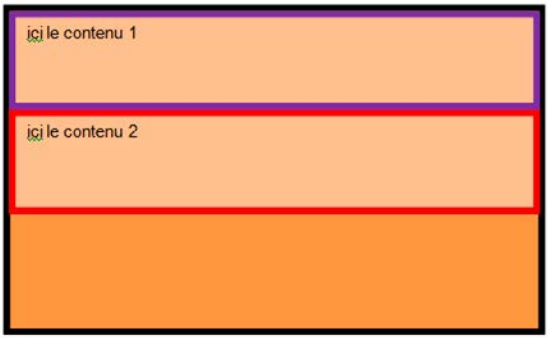

Fig. 8

Fichier HTML

```
<div id="conteneur">
    <div id="contenu_1">ici le contenu 1</div>
    <div id="contenu_2">ici le contenu 2</div>
</div>
```

Fig. 9

B. La mise en page

Les besoins de la mise en page vont nous amener à modifier le flux naturel des éléments de bloc, c'est-à-dire, le plus souvent, des balises structurantes. On a plusieurs besoins s'agissant de mise en page. Il faut d'abord distinguer ceux qui relèvent des règles que l'on a vues dans le chapitre sur modèle de boîte CSS et celles qui vont nous intéresser plus directement dans ce chapitre. Revoyons rapidement les premières avant de passer aux secondes.

D'une part, on a besoin de styliser les éléments de blocs (couleur, taille). Considérons par exemple les deux boîtes suivantes :

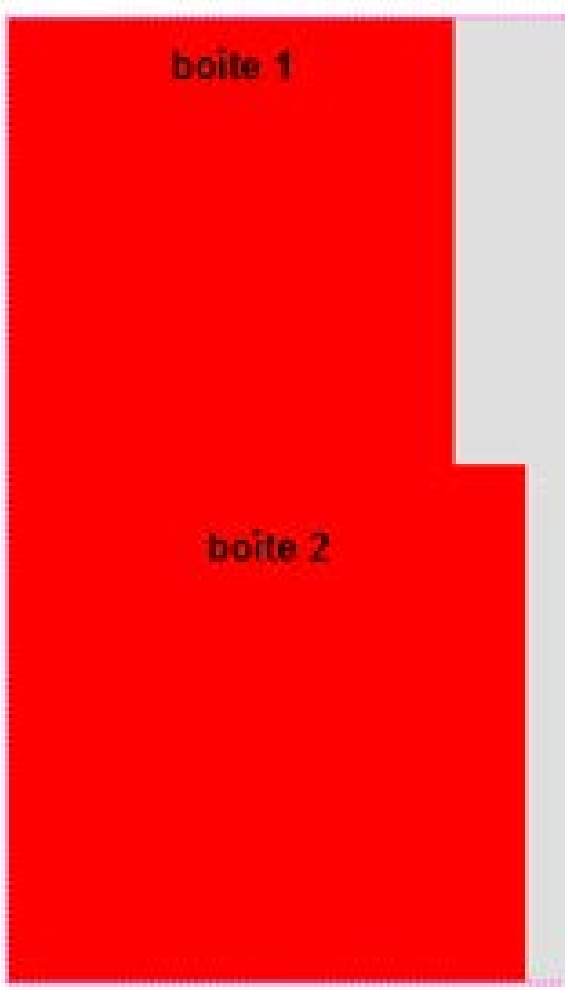

Fig. 10

On peut également espacer les blocs entre eux (ici on ajoute une marge basse sur le premier bloc).

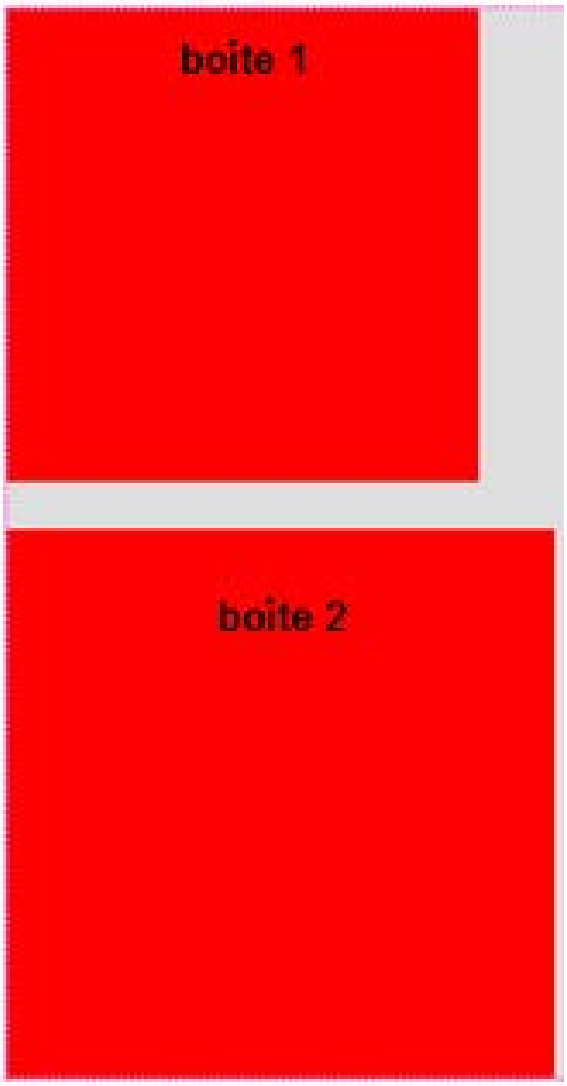

Fig. 11

À l'intérieur du bloc, on doit également savoir aérer le contenu. Exemple :

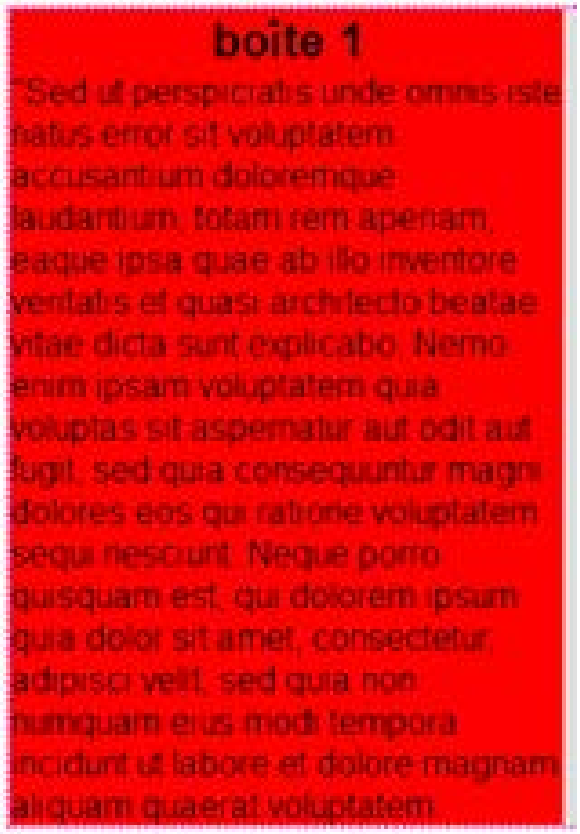

Fig. 12

Un simple **padding** permet de décongestionner tout cela :

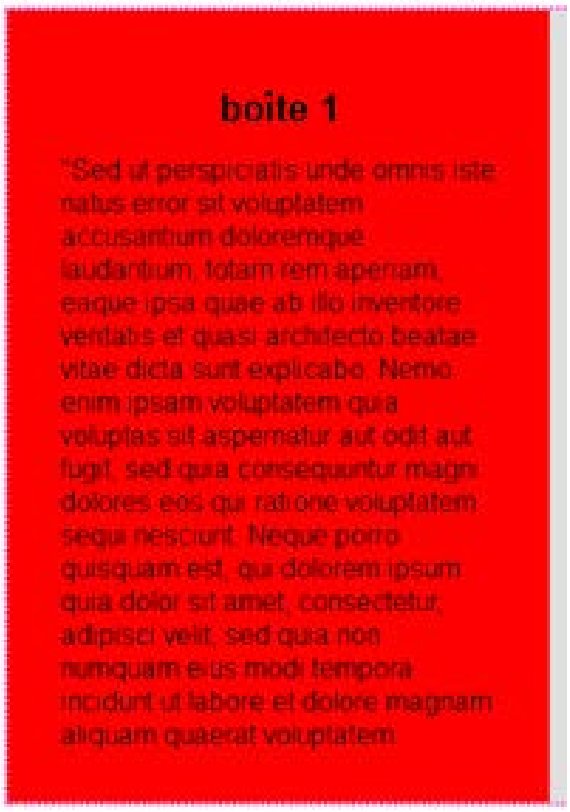

Fig. 13

C'est une simple révision et nous n'allons donc pas nous y attarder. Plus compliqué serait le fait d'avoir deux blocs côte à côte, comme ceci :

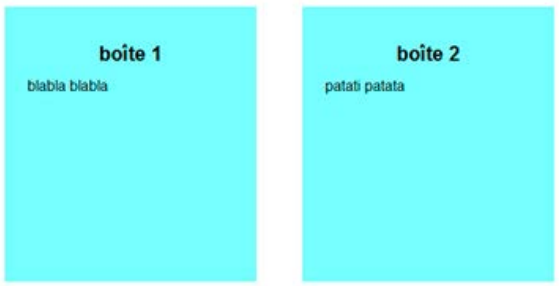

Fig. 14

En effet, pour ce genre de présentation, il faut toucher au flux naturel des éléments de blocs afin de le modifier.

C. Comment modifier le flux naturel

Il existe plusieurs méthodes pour modifier le flux naturel des éléments de blocs. Faisons un point rapide sur celles-ci en décrivant leurs avantages et leurs inconvénients.

1. Les flottements (float: left ; float: right)

C'est la méthode historiquement la plus utilisée. Elle est en effet très robuste, très souple et fonctionne depuis la nuit des temps, c'est-à-dire que même les vieux navigateurs la comprennent. Elle a également connu un regain lorsque le Responsive Design a fait son apparition.

En revanche, elle est plus compliquée à mettre en place et la compréhension de son fonctionnement est plus ardue. Elle demande l'ajout d'éléments HTML pour remettre les éléments qui suivent les flottements dans le flux naturel. Mal construite, elle génère des débordements et autres comportements visuels « absurdes ». Il faut savoir qu'elle n'a pas été conçue pour l'utilisation qui en est faite la plupart du temps. Néanmoins, il est essentiel de la maîtriser correctement.

Dans le cas vu plus haut, on aurait pour le HTML :

```
<div class="wrapper-boites">
    <div class="boite">
        <h3>boîte 1</h3>
        <p>blabla blabla</p>
    </div>
    <div class="boite" >
        <h3>boîte 2</h3>
        <p>patati patata</p>
    </div>
    <div class="clear"><div>
</div>
```

Fig. 15

Explication : une première **div** (**div.wrapper-boites**) enveloppe les deux boîtes HTML que l'on va faire flotter, après la seconde on stoppe la flottaison pour remettre les éléments suivant dans le flux naturel.

Et le css :

```
.wrapper-boites {
    width:700px;
    margin:0px auto;
}

.boite{
    width:250px;
    min-height: 250px;
    background-color: #aff;
    padding: 25px;
    margin:15px;
    float: left;
}
.clear {
  clear:both;
}
```

Fig. 16

Explications : **.wrapper-boites** fait 700px de large (c'est important pour la suite), les valeurs de margin permettent de centrer notre conteneur sur l'horizontal. La règle qui fait flotter nos deux boîtes (dans **.boite**) est **float: left** . La largeur des boîtes est égale à **width** (250px) + **paddinggauche** (25px) + **padding droite** (25px) + **margin gauche** (15px) + **margin droite** (15px), soit 330px par boîtes ce qui nous fait 660px pour les deux. Si d'aventure la largeur totale des éléments flottants excède celui du conteneur, les éléments qui ne tiennent pas sont rejetés sur une seconde ligne. Enfin. clear permet de stopper le flottement. Il est indispensable.

On aurait pu également utiliser **box-sizing: border-box** pour simplifier le calcul de la largeur de boîte (on calcule la largeur avec **width** (250px) + **margin gauche** (15px) + **margin droite** (15px), soit 280px par boîte ou 560px pour les deux).

2. Le modèle flexbox (display: flex)

Le modèle flexbox est plus qu'une méthode de positionnement, c'est un véritable modèle de mise en page proposant de nombreuses combinaisons s'agissant d'alignement, de distribution, d'agencement et de centrage des éléments. Appréhender l'ensemble du modèle flexbox et toutes les règles qui le composent demande de la pratique. Mais à présent que les navigateurs récents le comprennent, nous pouvons en utiliser très simplement des rudiments pour modifier le flux naturel.

Reprenons notre exemple, le HTML devient :

```
<div class="wrapper-boites">
    <div class="boite">
        <h3>boîte 1</h3>
        <p>blabla blabla</p>
    </div>
    <div class="boite" >
        <h3>boîte 2</h3>
        <p>patati patata</p>
    </div>
</div>
```

Fig. 17

Explications : Nous avons simplement supprimé la **div.clear** du code précédent car nous n'en avons plus besoin.

Le CSS :

```
.wrapper-boites {
    width:750px;
    margin:0px auto;
    display:flex;
}
.boite{
    width:250px;
    min-height: 250px;
    background-color: #aff;
    padding: 25px;
    margin:15px;
}
```

Fig. 18

Explications : Nous ajoutons au parent des éléments une règle simple : **display: flex** , nous supprimons la règle **float: left** sur **.boite**. Et c'est tout.

3. Les méthodes avec display (display: inline-block ; display: table-cell ; display: table ; display: table-row)

Il existe de nombreuses possibilités en utilisant **display** (nous avons traité flexbox à part, car c'est un sujet vaste et très particulier) qui est essentiellement un moyen de modifier des balises qui sont par défaut inline ou de block. En effet, avec **display** on peut, par exemple, demander à un élément de block de devenir inline et vice-versa, de se comporter comme un tableau, une cellule de tableau, etc.

Le sujet est également conséquent et nous allons uniquement nous concentrer sur **inline-block** qui est un mélange des comportements de block (l'élément possède une hauteur et une largeur) et inline (il répond aux instructions typographiques : alignement, hauteurs de lignes, etc.).

Reprenons notre exemple, le HTML devient :

```
<div class="wrapper-boites">
    <div class="boite">
        <h3>boîte 1</h3>
        <p>blabla blabla</p>
    </div>
    <div class="boite" >
        <h3>boîte 2</h3>
        <p>patati patata</p>
    </div>
</div>
```

Fig. 19

Explications : Même chose que précédemment avec le modèle flexbox.

Le CSS :

```css
.wrapper-boites {
    margin: 0 auto;
    width: 750px;
}

.boite {
    width:250px;
    height: 250px;
    background-color: #aff;
    padding: 25px;
    margin:15px;
    display: inline-block;
}
```

Fig. 20

Explications : Ici, on supprime le **display: flex** sur le parent. C'est sur les boîtes que l'on va indiquer qu'elles sont en **inline-block**.

Inconvénients : Entre deux blocs placés en **inline-block**, on a un petit espace incompressible. En effet, si l'on supprime le **margin** de la classe **.boite** (ici, on l'a placé en commentaire pour le neutraliser) :

```css
.boite {
    width:250px;
    height: 250px;
    background-color: #aff;
    padding: 25px;
/*  margin:15px;*/
    display: inline-block;
}
```

Fig. 21

On obtient :

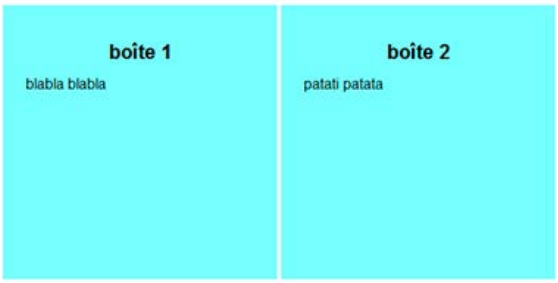

Fig. 22

Cela pourrait donc être un inconvénient dans certains cas même si l'on peut résoudre ce problème par l'intermédiaire d'un **hack** (un détournement ou contournement) (cf. cours avancé).

4. La règle position (position: relative ; position: absolute ; position: fixed ; position: sticky).

La règle **position** possède plusieurs valeurs qui modifient le comportement de l'élément sur lequel elle s'applique. Le principe fondamental est que cet élément va être positionné par rapport à un repère. À chaque valeur de position correspond un repère différent. En relatif, le repère est le positionnement naturel de l'élément ; en absolu, c'est le premier parent positionné ; en position fixe, c'est l'écran.

Adaptons notre exemple au cas d'un positionnement absolu.

Notre HTML de départ :

```html
<div class="wrapper-boites">
    <div class="boite">
        <h3>boîte 1</h3>
        <p>blabla blabla</p>
    </div>
    <div class="boite absolute">
        <h3>boîte 2</h3>
        <p>patati patata</p>
    </div>
</div>
```

Fig. 23

Explications : Presque identique aux exemples précédents, si ce n'est qu'on a ajouté une classe **.absolute** à la seconde boîte. En effet, seule celle-ci a besoin de sortir du flux naturel.

Le CSS :

```css
.wrapper-boites {
    margin: 0 auto;
    width: 750px;
    position: relative
}
.boite {
    width:250px;
    height: 250px;
    background-color: #aff;
    padding: 25px;
}

.absolute {
    position: absolute;
    top: 0;
    right: 0
}
```

Fig. 24

Explications : On ajoute **position: relative** au parent qui va servir de repère au positionnement. Notez qu'on a également supprimé la marge de l'élément boite, en effet nous n'en avons pas besoin. Enfin, la classe « absolute » nous permet d'indiquer le comportement de la deuxième boîte et de la placer en haut (**top**) à droite (**right**) du parent, soit le résultat suivant.

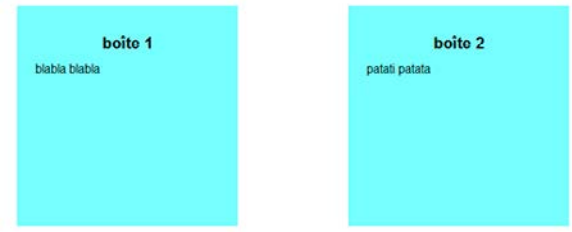

Fig. 25

La règle **position** est donc très puissante mais n'est utilisée que dans certains cas, en particulier pour les **superpositions d'éléments**, les **apparitions-disparitions** (avec ou sans superposition), les **menus**, **header** ou **footer** fixes. Il s'agit donc d'une méthode à n'utiliser que dans des cas très précis.

II. Conclusion

Une mise en page web précise demande avant tout de bien avoir assimilé le modèle de boîte CSS. Celui-ci permet d'éviter de nombreuses règles redondantes, comme l'ajout de largeurs identiques chez les enfants quand le parent possède déjà cette largeur.

Ensuite, il faut également maîtriser les différentes possibilités de modification du flux naturel. On utilisera dès lors sans complexe **flexbox**, le flottement, les méthodes avec **display** et même la règle **position**. En général, les trois premiers sont souvent interchangeables tandis que la règle **position** répond à des besoins plus spécifiques. Nous allons étudier leurs fonctionnements et leurs applications plus en détail dans les chapitres suivant.