CSS - RAPPELS - UNIV. MONTPELLIER - HMIN110 - M. HASCOET - 2018

FORME GÉNÉRALE

```
selecteur {
     attributCSS: valeur;
     attributCSS:valeur;
       ...
}
```

SÉLECTEURS

On rappelle ici les principaux éléments de syntaxe des sélecteurs css.

Les sélecteurs de base sont les sélecteurs par élément (nom de l'élément), classe (.), id (#) et le sélecteur universel (*)

- élément html, exemple : `h1`
- classe, exemple : `.titre`
- id, exemple : `#page`
- sélecteur universel : `*`

A ces sélecteurs de base s'ajoutent (1) les pseudo-classes, (2) les pseudo-éléments et (3) les sélecteurs composés par les opérateurs de composition.

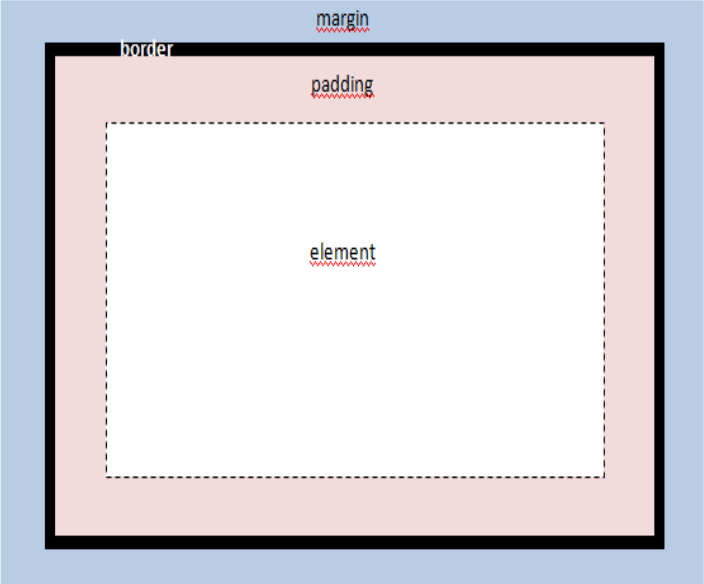

Figure 1: le modèle de boite avec margin, padding, border

PSEUDO-CLASSES

NB : la pseudo-classe agit comme un filtre sur une classe d'éléments sélectionnés alors que le pseudo élément crée un élément virtuel.

`:link`, `:visited`, `:hover`, `:active` lien non visité, visité, survolé, sélectionné

`:focus`, `:enabled`, `:disabled`, `:valid`, `:invalid`, `:required`, `:optional` susceptible de recevoir les entrées du clavier, actif, inactif, valide, invalide (au sens attributs de champs de formulaires html5 `required, type, pattern`).

`:first-child`, `:last-child`, `:nth-child(2), :nth-child(3n), nth-child(even), :only-child` premier enfant, dernier, 2ième, tous les trois enfants, tous les enfants pairs (`odd`, pour impairs).

`h2:first-of-type, h2:last-of-type, h2:nth-of-type, h2:only-of-type` premier élément h2 de son parent, dernier, nième, unique élément h2 de son parent.

`:target`, `:empty`, `:blank` contient un identifiant d'ancre, ne contient aucun élément ni texte, ne contient rien d'autres que des caractères d'espacement (espace, tab, nl, etc)

PSEUDO-ÉLÉMENTS

NB : Evolution de la syntaxe : devient :: pour les pseudo-éléments. Attention, pas de support pour la nouvelle syntaxe dans ie8.

::before,::after,::first-line,::first-letter,::selection - élément avant, après, première ligne, première lettre, en cours de sélection.

Exemples :

```
/* ajout d'une image devant les titres de niveau 1*/
h1::before {
     content:url(puce.png) ;
}

p::first-letter {
     color:rgb(128,64,64) ;
     font-size : 1.5em ;
}
```

COMPOSITION DES SÉLECTEURS

La composition de sélecteurs de base permet de créer de nouveaux sélecteurs.

	Signification	Exemple de l'usage du sélecteur dans une règle css
,	union de sélecteurs.	`h1, h2, h3 { font-family: verdana; }`
.	intersection de sélecteurs.	div.menu *un élément div et de classe menu*
esp	descendant	`.menu li { margin-top: -5mm; }` *dans tout item de liste contenu dans un élément de classe menu*
>	enfant direct	`.button > a {width: 20px; }` *Largeur de l'ancre lorsqu'elle se trouve comme enfant direct d'un élément de classe button*
+	adjacent direct	`h1 + h2 { margin-top: -5mm; }` *Si un h2 suit immédiatement un h1, remonter la marge haute du h2.*
~	tous les éléments adjacents	`input:invalid ~ [type= "submit"]{opacity : .3 ; pointer-event:none;}` *Si un champs input est invalide, désactive le bouton submit adjacent.*
[]	spécialisation par valeur sur attribut (attention, peut augmenter les temps de réponse)	`a [href ~ "http"]{ color = rgb(64,128,128) ; }` *Si un lien pointe vers un site externe, la couleur du lien peut différer de celle des autres liens.*

HÉRITAGE DES RÈGLES – DIFFUSION DES STYLES DANS LE DOM

L'héritage correspond à l'application implicite de règles du fait de l'imbrication des éléments html. Ainsi, une règle écrite pour certains éléments a de grandes chances d'être héritée par les enfants de ces éléments dans le dom. Nombreuses propriétés CSS s'héritent (par exemple, `font-style, color`), mais attention, pas toutes. Quelques propriétés CSS très utilisées ne s'héritent pas : c'est le cas, par exemple, de `border, margin, padding`.

CASCADES – FUSION DE DIFFÉRENTS STYLES POUR UN MÊME ÉLÉMENT HTML

En CSS, plusieurs règles peuvent être applicables aux mêmes éléments. Lorsque ces règles ne sont pas conflictuelles, elles s'appliquent toutes. Lorsqu'elles sont conflictuelles, c'est à dire lorsque plusieurs règles fixent des valeurs différentes du même attribut css, le principe de cascade définit laquelle des règles est appliquée si aucune règle ne comporte de mention `!important` (sinon le principe de cascade est désactivé et c'est la règle qui comporte la mention `!important` qui est appliquée).

Lorsque la cascade est appliquée dans un cas conflictuelle, la règle appliquée est cette qui est la plus importante dans l'ordre construit à partir des ordres partiels suivants :

1. D'abord, la source de la règle

   Il existe trois principales sources pour une spécification css : « css externe » < « css "inline" « < « css attribut d'élements html ». Ainsi, un style fixé directement sur l'élément l'emporte sur un style fixé dans la css.

2. Ensuite, la spécificité du sélecteur

   La spécificité du sélecteur peut faire l'objet d'un calcul en cas d'ambiguïté. La spécificité est alors calculée comme le nombre

   $$s = a*100+b*10+c \text{ où}$$

   a est le nombre d'id,

   b est le nombre de classes et

c est le nombre d'éléments html.

Plus la spécifité du sélecteur est grande plus la règle associée est prioritaire.

3. Enfin, l'ordre de la spécification

   Lorsque deux règles ont le même poids, la dernière règle spécifiée l'emporte.

Exemple:

```
body{
     color : black ! important
     margin-top: 2em;
}
.menu li {
     color : blue ;
     margin-top: -1em;
}
```

Avec le mot clé `!important`, le principe de cascade est désactivé et dans l'exemple, la couleur d'avant plan noire est appliquée même pour les éléments sélectionnés par .menu li.

En revanche, l'attribut de margin-top à -1em est prioritaire sur la valeur de 2em pour les éléments sélectionnés par .menu li (s=11), car leur sélecteur est plus sélectif que body (s= 1).

UNITÉS

em	equivalent m width, unité relative à la police du parent. Par exemple, margin : 3 em est une marge de la largeur de trois caractère m dans la police du parent.
rem	*root em* idem em mais en rapport à l'élément html. Pallie problèmes liés à l'usage de em dans éléments imbriqués. Attention, pas supportée par Ie8.
pt	point, ~ 0,35 mm, unité héritée de la typographie, 1 pt correspondait à la taille du plus petit espace blanc en typographie
pc	pica, ~ 12 pt ~ 4,22mm
in	inch, ~ 2,54 cm.
%	mesure en pourcentage de celle du parent
vh,vw	vh et vw sont comme % mais en rapport respectivement à la largeur et à la hauteur du viewport, partie visible de la page dans le navigateur.

MISE EN PAGE - GÉNÉRALITÉS

Les principaux attributs génériques css concernés par la mise en page sont : display (environ 42 valeurs possibles), position (environ 8 valeurs), margin*, padding*, et border. De nombreux autres attributs css peuvent être utilisés mais ils sont spécifiques à chaque stratégie de mise en page.

Tout élément html peut-être considéré comme englobé par un rectangle. Ce rectangle, appelé boite le plus souvent, a des marges intérieures (attributs padding), des marges extérieures (attributs margin), une bordure (attribut border) et éventuellement une scrollbar. L'épaisseur et la visibilité des marges et des bordures peuvent être modifiées. Cette boite est représentée par la Figure 1.

La mesure de la taille de la boite est sujet à instabilités et controverses depuis l'existence du modèle de boite. Il en résulte toutes sortes de choix qui peuvent paraître contre-intuitifs. Par exemple, en CSS, aujourd'hui, dans ce qu'on appellera le mode "normal", les attributs `width` et `height` ou `max-width` et `max-height` spécifient la taille des *éléments, et ne spécifient* pas, a priori, la taille de la boîte qui reste à calculer. Même si le calcul est simple - taille de la boite = taille de l'élément + taille des marges intérieures + marges extérieures + taille des bordures. Il reste néanmoins indéterminé sur les marges automatiques relatives et d'autres espaces implicites qui interviennent dans le calcul comme, par exemple, l'espace alloué aux potentielles scrollbars.

Une alternative à cette mesure est toutefois possible. En utilisant l'attribut `box-sizing:border-box` à partir de CSS3, la taille de la boite est alors ce qui est défini par `width` et `height`, et c'est la taille de l'élément qui est à calculer en retirant les tailles des marges intérieures et bordures. Une manière de l'utiliser sur tout un document est de l'écrire ainsi :

```
html{
     box-sizing:border-box ;
```

```
}
```

La règle est alors héritée à tous les éléments. Les autres valeurs possible de box-sizing sont `content-box` (retour au mode "normal") et `padding-box` (très peu supporté).

Une autre évolution importante avec css3 et html5 est la modification du type des boites. Anciennement, les éléments html relevaient d'un type de boite par défaut et les principaux types de boite étaient **`inline`**, **`block`** ou **`inline-block`**. Avec html5, les types de boites sont progressivement remplacés par des types plus sémantiques spécifiés par l'attribut display.

MISE EN PAGE - LE MODÈLE DE FLUX - ATTRIBUT POSITION

Le modèle de flux positionne les élements dans l'ordre de lecture en les plaçant de haut en bas et de gauche à droite sauf mention explicite d'un positionnement différent. Quatre modes de positionnement existent pour les éléments : **`static`**, **`absolute`**, **`fixed`**, **`relative`**, **`sticky`**.

Le mode **`static`** est le mode par défaut suivant les spécifications de mise en page plus globales.

Pour les autres modes - `absolute`, `fixed`, `relative`, `sticky`, les éléments seront placés par rapport à un cadre de référence en fonction des valeurs des attributs `top`, `bottom`, `left`, `right` qu'il convient donc de spécifier par ailleurs.

A chaque mode - `absolute`, `fixed`, `relative`, `sticky` correspond un cadre de référence différent :

Mode/Valeur de l'attribut position	Cadre de référence
`relative`	La boite correspondant à la position « normale » de l'élément
`absolute`	La boite du plus proche ancêtre *placé* de l'élément
`fixed`	Le viewport, c'est à dire la zone visible du navigateur. Position indépendante du défilement possible lors de la lecture.
`sticky`	Combine `fixed` et `relative` : le cadre de référence est la boite correspondant à la position normale de l'élément, tant que cette boite est dans le `viewport`, et devient le viewport dès que cette boite sort du viewport.

MISE EN PAGE - LE MODÈLE MULTI-COLONNES - ATTRIBUTS COLUMN*

A partir de CSS3, il est possible de réaliser une mise en page en multi-colonnes beaucoup plus simplement qu'avant. Les calculs sont pris en charge par le navigateur qui calcule des colonnes de même largeur et, par défaut, même hauteur. Il reste possible pour le concepteur de spécifier explicitement :

- le nombre de colonnes - `column-count`,
- la largeur des colonnes - `column-width`,
- la taille de la gouttière - espace entre les colonnes - `column-gap`,
- la forme d'une éventuelle bordure de séparation entre les colonnes - `column-rule-color`, `column-rule-width` et `column-rule-style` ainsi que
- le mode de remplissage des colonnes dont découle la hauteur des colonnes - `column-fill`.

En outre, l'usage de l'attribut `columns` est possible pour condenser les spécifications de `columm-width` et `column-count` en une seule spécification. Les deux règles ci-dessous sont équivalentes et l'ordre des valeurs dans `column` est sans importance a priori car les unités permettent de discriminer entre nombre et largeur.

```
.multic{
    column-count:auto;
    column-width:20em ;
    column-gap:2em ;
}
```

```
.multic{
    columns:auto 20em;
    column-gap:2em ;
}
```

Le modèle multi-colonnes permet également de positionner des éléments sur plusieurs colonnes avec l'attribut `column-span` à deux valeurs possibles `none` et `all`.

LE MODÈLE DES BOITES FLEXIBLES AVEC FLEXBOX, ATTRIBUT DISPLAY : FLEX

Css3 définit également le modèle de boites flexibles qui introduit une douzaine de propriétés css supplémentaires pour définir le comportement des « enfants » (dans le dom) d'un élément dont l'attribut display est à flex. Par exemple, le plus simple usage de flex permet un centrage parfait d'un élément de classe .image dans un élément de classe .conteneur:

```
.conteneur{
    display: flex;
    width: 300px;
    height: 300px;
}
.image{
    width: 100px;
    height: 100px;
    margin: auto;
}
```

Les autres attributs utiles au paramétrage d'une mise en page selon le modèle des boites flexibles s'appliquent soit aux éléments conteneur (parent dans le dom), soit aux éléments contenus (enfants dans le dom).

1. Attributs d'un conteneur

a) `display: flex;`
b) `flex-direction: row | row-reverse | column | column-reverse;`
c) `flex-wrap: nowrap | wrap | wrap-reverse;`
d) `flex-flow: <'flex-direction'> || <'flex-wrap'>`
e) `justify-content: flex-start | flex-end | center | space-between | space-around | space-evenly;`
f) `align-items: flex-start | flex-end | center | baseline | stretch;`
g) `align-content: flex-start | flex-end | center | space-between | space-around | stretch;`

La direction et l'orientation des éléments contenus dans la boite flexible sont spécifiés par flex-direction (de gauche à droite, de droite à gauche, de haut en bas, de bas en haut). Le passage à la ligne est réglé par flex-wrap. Flex-flow permet de spécifier direction, orientation, passage à la ligne.

Ensuite, les attributs justify-content, align-items, align-content pour un conteneur, utilisent les notions d'axes principal/orthogonal. L'axe principal est spécifié explicitement par flex-direction : row spécifie l'axe principal comme horizontal, et l'axe orthogonal est donc, dans le cas où flex-wrap n'est pas à nowrap, l'axe vertical.

justify-content, align-items, align-content permettent alors de spécifier respectivement l'alignement selon l'axe principal (en mono-ligne ou mono-colonne),l'alignement selon l'axe orthogonal (en mono-ligne ou mono-colonne), l'alignement des lignes/colonnes selon l'axe orthogonal lorsqu'il y a plusieurs lignes/plusieurs colonnes.

2. Attributs d'un contenu

a) `order: <integer>;`
b) `flex-grow: <number>;`
c) `flex-shrink: <number>;`
d) `flex-basis: <length> | auto;`
e) `flex: none | [ <'flex-grow'> | <'flex-shrink'>? | <'flex-basis'> ]`
f) `align-self: auto | flex-start | flex-end | center | baseline | stretch;`

LE MODÈLE DE GRILLES, ATTRIBUT DISPLAY : GRID

Le modèle de grille repose sur des conteneurs ayant un attribut css `display : grid` et des contenus dont le placement dépend principalement des attributs `grid-column` et `grid-row`.