L'objectif de ce poly est de survoler les principes de CSS : sélecteurs, priorités, propriétés.

1 Introduction

1.1 Pourquoi CSS ?

La vocation de CSS est de séparer le contenu (et son organisation logique) de sa mise en forme. Pour cette dernière, un certains nombres de *propriétés* sont définies, sur tout contenu du document. Certaines concernent la police, d'autres le texte, d'autres encore la boîte, *etc.* Il existe 5 façons de donner une valeur à ces propriétés :

- *indirectement*, par le biais d'attributs (*ex.* : utilisation de "frame" et "box" pour un tableau) ;
- *implicitement*, par *héritage* du contenant (*ex.* : un paragraphe "p" hérite des valeurs de propriétés de la section "div" qui le contient, et qui hérite elle-même de l'élément "body") ;
- *explicitement*, par utilisation de l'attribut "style", à l'ouverture d'un élément (ce qu'on appelle les styles "en ligne") ;
- par spécification d'instructions CSS de mise en forme au travers l'élément d'entête "style", en entête du fichier (x)html ;
- par spécification d'instructions CSS de mise en forme au travers un fichier CSS, extérieur (mais lié) au fichier (x)html.

Remarques :

- des valeurs par défaut sont définies par le navigateur client ;
- l'héritage n'est pas systématique (en fonction des priorités des diverses instructions de style concernant un élément donné), mais peut être spécifié par la valeur de propriété "inherit".

Illustration :

```
<div style="font-weight:normal;">
  <strong>épaisseur de trait "bold" définie par le CSS du navigateur</strong>
  <strong style="font-weight:inherit;">épaisseur de trait normale héritée du contenant "div"</strong>
</div>
```

De l'intérêt de CSS :

1. L'attribut "style", défini sur tout élément (dont les éléments de regroupement générique "div" et "span"), permet déjà de concentrer les instructions de mise en forme au sein d'un attribut unique.
2. L'utilisation de CSS via l'élément "style" permet de séparer clairement les instructions de mise en forme du contenu en les déportant dans l'entête du fichier (x)html ; cela permet également de factoriser ces instructions entre différentes instances d'éléments (plusieurs instances d'un même élément peuvent partager les mêmes instructions de mise en forme !).
3. Enfin, l'utilisation de CSS via un fichier "css", en externalisant physiquement ces instructions du contenu, parfait la séparation entre contenu et mise en forme (nul besoin de modifier le fichier (x)html pour en modifier la mise en forme... cas de la personalisation de l'affichage d'un même fichier par exemple !) ; cette fois, les instructions de mise en forme sont factorisées aussi entre différents fichiers (x)html (plusieurs fichiers html peuvent partager le même fichier css !).

1.2 Retour sur quelques éléments de style (x)html

Il s'agit ici de lister des entités logiques (éléments (x)html) auxquelles une mise en forme relativement complexe est attribuée par défaut. Ces entités permettent le plus souvent de marquer des morceaux de contenu qui jouent un rôle spécifique dans un document (hiérarchisation pour les titres, typage de contenu selon son importance, sa sémantique). Cette liste n'est pas exhaustive... d'autant que *tout élément*, d'une part, a une signification logique et, d'autre part, est dôté d'un *comportement par défaut* !

1.2.1 Niveau bloc

- éléments de titres hiérarchisés : h1,...,h6
- nature de contenu : blockquote (citation niveau bloc : tabulation), address (adresse)
- tableaux : cellules d'entête th

1.2.2 Niveau texte

- exergue : em (faible), strong (forte)
- nature de contenu : q (citation niveau texte : guillements), cite (citation d'une référence), code (code informatique), dfn (définition), abbr (abbréviation), acronym (acronyme), sup (exposant), sub (indice)

2 Syntaxe CSS

2.1 Pour mémoire : attribut "style"

L'attribut "style" permet de spécifier la valeur de toute propriété de style et est défini pour (quasi) tous les éléments de contenu (x)html. La valeur que l'on affecte à l'attribut "style" est un peu particulière, puisque l'on peut lister un ensemble de valeurs pour différentes propriétés de style. La syntaxe est la suivante :

<élément style="propriété$_1$:valeur$_1$; propriété$_2$:valeur$_2$; ... propriété$_\ell$:valeur$_\ell$; ">...</élément>

Exemple :

```
<body style="text-align:justify; color:black; background-color:#EEEEFF;">
<h4 style="font-variant: small-caps; background-color: #CCCCCC;">blab bla</h4>
```

2.2 Élément "style"

Tant qu'à définir des styles, autant les définir une fois pour toutes. Par exemple, pour afficher tous les titres de niveau 4 en petites capitales sur fond gris, et plutôt que d'affecter les deux propriétés "font-variant" et "background-color" à chaque ouverture de l'élément "h4", on peut le faire une fois pour toutes dans la zone d'entête, à l'aide de l'élément "style" :

```
<head>
  ...
  <style>
    h4 {font-variant: small-caps; background-color: #CCCCCC;}
  </style>
</head>
```

Si l'on écrit la ligne précédente dans la zone d'entête, alors à chaque ouverture de l'élément "h4", le navigateur saura qu'il doit écrire des petites capitales sur fond gris. On peut ainsi spécifier autant de styles d'élément que l'on veut dans l'entête, que ce soit à l'intérieur du même élément "style" ou dans des éléments "style" séparés :

utilisation d'un seul élément

```
<head>
  ...
  <style>
    body {text-align:justify; color:black; background-color:#EEEEFF;}
    h4 {font-variant: small-caps; background-color: #CCCCCC;}
  </style>
</head>
```

utilisation de deux éléments

```
<head>
  ...
  <style>
    body {text-align:justify; color:black; background-color:#EEEEFF;}
  </style>
  <style>
    h4 {font-variant: small-caps; background-color: #CCCCCC;}
  </style>
</head>
```

La syntaxe est quasiment la même que pour définir la valeur d'un attribut "style". Au lieu d'écrire *à l'ouverture de chaque instance d'élément concernée* :

<élément style="propiete_1:valeur_1;propiete_2:valeur_2;...;propiete_k:valeur_k;">...</élément>

On écrit *en entête de fichier*, entre deux balises ouvrante et fermante <style> et </style> :

élément {propiete_1:valeur_1;propiete_2:valeur_2;...;propiete_k:valeur_k;}

2.3 Fichier css

2.3.1 Le fichier lui-même

On peut déporter les instructions de mise ne forme de l'entête des fichiers (x)html vers un fichier externe d'extension ".css". Puisque la syntaxe de définition est exactement la même qu'à l'intérieur de l'élément "style", le fichier "css" n'est qu'une succession d'instructions de mise en forme. Un exemple de fichier css est proposé en figure 1.

2.4 Lier le fichier css au fichier (x)html

Pour indiquer qu'un fichier (x)html utilise un fichier, il suffit d'ajouter dans la zone d'entête l'élément de lien "link" avec les bonnes valeurs d'attributs :

```
<!-- pour associer un fichier de style au fichier html -->
<link rel="stylesheet" type="text/css" href="myDirectory/css/myCssFile.css">
```

L'attribut "rel" précise la nature de la relation (rôle que joue le document lié pour le document (x)html courant), l'attribut "type" indique le type de fichier (type MIME), tandis l'attribut "href" donne l'url du fichier à attacher. En gros (pas tout à fait, mais bon), cela revient à faire un "#include" en C.

3 Sélecteurs

Une chose est de définir une mise en forme (liste de valeurs de propriétés) :

{propiete_1:valeur_1;propiete_2:valeur_2;...;propiete_k:valeur_k;}

Figure 1: Fichier css

```
body
{
    background-image: url('../img/logoLIPN.jpg');
    color: black;
    font-style: normal;
    text-align: left;
}
h1
{
    background-color: grey;
    color: blue;
    text-align:center;
}
h2
{
    background-color: grey;
    color: green;
    text-align:left;
}
p
{
    background-color: grey;
    text-align: left;
    font-style: italic;
}
strong
{
    color: red;
    font-style: normal;
    font-weight: bolder;
}
```

Une autre est de déterminer l'ensemble des *éléments* ou des *instances d'éléments* auquel cette mise en forme s'applique ! Pour faire cela, on dispose en CSS d'un ensemble de sélecteurs, qui permettent précisément de décrire des éléments, des instances d'éléments, éventuellement en fonction de la valeur de certains attributs ("class", "id"), éventuellement encore en fonction de leur contexte (imbrication / succession d'éléments).

3.1 Sélecteurs d'éléments

1. de type : ELEMENT
   syntaxe css : ELEMENT {}
   instances sélectionnées : toute instance de l'élément ELEMENT
   exemple : body{background-image:url(../img/logoLIPN.jpg);background-attachment:scroll;}

2. universel : *
   syntaxe css : * {}
   instances sélectionnées : toute instance de tout élément
   exemple : *{font-style:normal;}

3. de classe : .
   syntaxe css : ELEMENT.classe {}
   syntaxe (x)html : <ELEMENT class="classe">...</ELEMENT>
   instances sélectionnées : toute instance de l'élément ELEMENT qui est de la classe spécifiée
   exemple :

.red{color:red;}	... <h1 class="red">...</h1>
div.red{color:red;}	...

4. d'identifiant (instance unique) : #
   syntaxe css : ELEMENT#identifiant {}
   syntaxe (x)html : <ELEMENT id="identifiant">...</ELEMENT>
   instances sélectionnées : l'unique instance de l'élément ELEMENT ayant l'identifiant spécifié
   (remarque : puisque les identifiants sont uniques indifféremment des éléments, la mention de l'élément est inutile)
   exemple :

ul#menu > li{display: block; position: fixed; left: 5px;}	<ul id="menu"> ...
#parcours{font-size:smaller;font-style:italic;}	...

5. de descendants : ELEMENT1 ELEMENT2
   syntaxe css : ELEMENT1 ELEMENT2 {}
   instances sélectionnées : toute instance de ELEMENT2 qui **descend** *directement ou indirectement* d'une instance de ELEMENT1
   exemple : #parcours p {color:red;}

6. de fils : >
   syntaxe css : ELEMENT1 > ELEMENT2 {}
   instances sélectionnées : toute instance de ELEMENT2 qui **descend** *directement* d'une instance de ELEMENT1
   (remarque : "ELEMENT1 > ELEMENT2" est donc plus restrictif - sélectionne un ensemble plus restreint d'instances - que "ELEMENT1 ELEMENT2")
   exemple : #parcours > p {text-indent:10px;}

7. d'adjacent (premier *suivant*) : +
   (premier *suivant* : + (syntaxe css : ELEMENT1 + ELEMENT2 {})
   instances sélectionnées : toute instance de ELEMENT2 qui **suit** *directement* une instance de ELEMENT1 (petit frère direct)
   exemple : h1+p{text-indent:10px;}

3.2 Pseudo classes ":"

Les pseudo classes sont des classes contextuelles (contexte classique d'imbrication des objets (x)html ou contexte dynamique de navigation) qui sont prédéfinies sur tout élément ou sur des éléments spécifiques.

1. premier et dernier fils : first-child, last-child (CCS2 uniquement pour ce dernier)
   syntaxe css : ELEMENT:first-child{}, ELEMENT:last-child{}
   instances sélectionnées : toute instance de l'élément ELEMENT qui est le *premier fils* (*resp.*, le *dernier fils*) de son père (quel que soit l'élément de celui-ci !).
   exemple :

p:first-child{text-indent:10px;} p:last-child:after{content:"EOD";}	<**p**>**...**</**p**> ... ... <**p**>**...**</**p**>

2. élément "a" et état dynamique des liens : link, visited, active, hover, focus
   syntaxe css : a:link{}, a:visited{}, a:active{}, a:hover{}, a:focus{}
   instances sélectionnées : toute instance de l'élément "a" qui est (respectivement) en état non visité, visité d'une part, actif, survolé par la souris, focus, d'autre part.
   (remarque : un lien peut conjontement avoir été visité et subir un passage de la souris !)
   exemple :

```
a{color: #9999DD; text-decoration:none;}
a:visited{color: #444488;}    a:link{color: #9999DD;}
a:hover{color: #FFAC00; font-style:italic;}    a:active{color: #FFAC00;}
```

3.3 Pseudo éléments ":"

Les pseudo éléments permettent d'agir sur le contenu des éléments.

1. insertion avant / après : pseudo éléments "before" et "after"
   syntaxe css : ELEMENT:after{content:*contenu*;}, ELEMENT:before{content:*contenu*;}
   instances sélectionnées : toute instance de l'élément ELEMENT
   but : permet d'insérer (du texte, une image,...) avant / après le contenu de l'instance
   exemple : .commentaire{font-style:italic;}
   .commentaire:before{content:"/*";} .commentaire:after{content:"*/";}

2. première lettre / première ligne : first-line, first-letter
   syntaxe css : ELEMENT:first-line{}, ELEMENT:first-letter{}
   instances sélectionnées : toute instance de l'élément ELEMENT
   but : permet d'attribuer une mise en forme spécifique à chaque première lettre (*premier ligne*) du contenu de l'instance (pour un élément de niveau bloc, et si celui-ci contient du texte).
   exemple : div:first-letter{color: #FF0000; font-size: xx-large;}
   div:first-line{color: #FF0000; texte-decoration: underline; font-variant: small-caps;}

3.4 On peut tout mélanger...

Évidemment, les sélecteurs se composent ! Quelques exemples de sélecteurs :

sélecteur	**désgine**
*.red	toute instance de tout élément ouvert avec "class="red""
ul *	toute instance de tout élément qui descend d'une instance d'un élément "ul"
ul > li	toute instance de "li" qui est fille d'une instance "ul"
ul#menu > li	toute instance de "li" qui est fille de l'instance "ul" d'identifiant "menu"
p:first-letter	toute instance de "p"
tfoot > td, th	toute instance de "td" ou de "th" qui est fille d'une instance "tfoot"

4 Priorités des instructions de style

1. Selon le lieu : plus l'instruction est proche de la définition du contenu, plus elle est prioritaire.

   style "en-ligne" avec l'attribut style > élément style > fichier css > navigateur

2. Selon l'imbrication des éléments : si le contenant hérite du contenu, les instructions définies au niveau du contenu supplantent celles définies au niveau du contenant. Par exemple, si un paragraphe est contenu dans un tableau (via une cellule, via une ligne), lui-même contenu dans une division fille de "body" :

   p > td > tr > table > div > body

3. Selon la généricité : plus une instruction est spécialisée (désigne un ensemble restreint d'instances), plus elle est prioritaire. Soit :

   (ELT#id) > (ELT.class) > (ELT1 + ELT) > (ELT1 > ELT) > (ELT1 ELT) > (ELT) > (*)

5 Propriétés

5.1 Liste (non exhaustive !) de propriétés

s'applique à	propriété	exemples
contenu	background-color	background-color: blue; background-color: #0000FF;
contenu	background-image	background-image: url('../img/logoLIPN.jpg');
contenu	background-repeat	background-repeat: repeat;
contenu	background-attachment	background-attachment: fixed;
contenu	background-position	background-position: top-center;
contenu	color	color: blue; color: #0000FF;
contenu	text-align	text-align: justify;
contenu	text-indent	text-indent: 1cm; text-indent: 10px;
contenu	text-transform	text-transform: uppercase;
contenu	vertical-align	vertical-align: middle;
contenu	font-family	font-family: Lucida, sans-serif;
contenu	font-style	font-style: italic;
contenu	font-variant	font-variant: small-caps;
contenu	font-weight	font-weight: bold; font-weight: bolder;
contenu	font-size	font-size:x-small; font-size: larger;
boîte	position	position: relative;
boîte	left, right, bottom, top	left: 2px; left: 5%;
boîte	float	float: right;
boîte	z-index	z-index: 2;
boîte	display	display: inline;
boîte	width	width: auto; width: 280px; width: 80%;
boîte	height	height: auto; height: 280px; height: 80%;
boîte	border-width	border-width: thin; border-width: 5px;
border-bottom-width, border-left-width, border-right-width, border-top-width		
boîte	border-style	border-style: groove;
border-bottom-style, border-left-style, border-right-style, border-top-style		
boîte	border-color	border-color: #CCCCFF; border-color: red;
border-bottom-color, border-left-color, border-right-color, border-top-color		
boîte	margin	margin: auto; margin: 15px; margin: 2%;
margin-bottom, margin-top, margin-left, margin-right		
boîte	padding	padding: auto; padding: 15px; padding: 2%;
padding-bottom, padding-top, padding-left, padding-right		
boîte	outline-width	outline-width: thin; outline-width: 5px;
boîte	outline-style	outline-style: groove;
boîte	outline-color	outline-color: #CCCCFF; outline-color: invert;
contenu	list-style-type	lower-roman;
contenu	list-style-position	list-style-position: inside;

5.2 Quelques éclaircissements

Image de fond

background-image	url de l'image url('chemin_image')	
background-repeat	comment l'image est-elle (ou non !) répétée	{repeat, repeat-x, repeat-y, no-repeat}
background-attachment	l'image suit-elle le texte	{scroll, fixed}
background-position	position de l'image	{top, bottom} × {left, center, right}

Texte

text-indent	indentation de la première ligne d'un bloc de texte (*ex.* : text-indent: 1cm;)
text-transform	casse des mots d'un bloc de texte, de valeur ∈ {uppercase, lowercase, capitalize}
	"capitalize" met la première lettre de chaque mot en caractère majuscule

Police

font-family	nom de la famille de police (*ex.* arial, times, courier, times,...) + famille générique (*ex.* serif, sans-serif, cursive, fantasy, monospace,...), séparés par une virgule
exemples :	font-family:arial; font-family: arial, "lucida console", sans-serif;
font-variant	normal, small-caps
font-style	normal, italic, oblique

Bordure (= contour de la boîte de contenu)

border-width	largeur de la bordure	{thin, medium, thick} ∪ {*longueur absolue*}
border-style	style de la bordure	{none, dashed, solid, groove, ridge, inset, outset}
border-color	couleur de la bordure	{red, black, ...} ∪ {#RRVVBB}
exemples :	border-top-width: 2px; border-top-color: red; border-top-style: groove;	
	border-top: red groove 2px; (opérateur compact)	

Marge (= espace entre le *bord de l'élément* et le *bord du contenu de son contenant*)

margin-bottom	la marge du bas	{auto} ∪ {*longueur absolue*} ∪ {*longueur relative*}
margin-left	marge de gauche	*ex.* : margin-left: auto;
margin-right	marge de droite	*ex.* : margin-right: 10px;
margin-top	marge du haut	*ex.* : margin-top: 5%;

Espacement (= espace entre le *bord de l'élément* et le *bord de son contenu*)

padding-bottom	espace du bas	{auto} ∪ {*longueur absolue*} ∪ {*longueur relative*}
padding-left	espace à gauche	*ex.* : padding-left: 10px;
padding-right	espace à droite	*ex.* : margin-right: 5%;
padding-top	espace en haut	

Relief (= contour *extérieur* à la bordure de la boîte de contenu)

outline-width	largeur du trait	{thin, medium, thick} ∪ {*longueur absolue*}
outline-style	style du trait	{none, dashed, solid, groove, ridge, inset, outset}
outline-color	couleur du trait	{invert} ∪ {red, black, ...} ∪ {#RRVVBB}
exemples :	outline-width: 1px; outline-color: invert; outline-style: solid;	
	outline: invert solid 1px; (opérateur compact)	

Liste

list-style-type	style de puce : ∈{none, disc, circle, square} ∪ {decimal, lower-roman, upper-roman, lower-alpha, upper-alpha} ∪ {lower-greek, upper-greek, lower-latin, upper-latin}
list-style-position	situation du texte par rapport à la puce : ∈{inside, outside}

Dimensions

width, height	largeur, hauteur	{auto} ∪ {*longueur absolue*} ∪ {*longueur relative*}
line-height	distance entre deux lignes	{normal} ∪ {nombre de lignes} ∪ {*longueur absolue*} ∪ {*longueur relative à la taille de police*}

6 Positionnement

6.1 Type de boîte

On a déjà vu qu'il existait différents types de boîtes (et de valeurs de propriété d'affichage associée), essentiellement :

1. Bloc (propriété display == block) : l'élément se place *en-dessous* de l'élément précedent et *au-dessus* de l'élément suivant.

2. Texte ou en-ligne (propriété display == inline) : l'élément se place *à droite* de l'élément précedent et *à gauche* de l'élément suivant.

6.2 Référence

Les propriétés "left", "right", "top", "bottom" permettent de préciser la position d'un élément. Par exemple, si l'on écrit {left: 5px; top: 10px;}, cela signifie que l'objet doit être placé à 5 pixels du bord qui est à sa gauche, à 10 pixels du bord qui est au-dessus de lui. La question du *positionnement* est de savoir... de quel bords parle-t-on ! C'est ce que permet de définir la propriété "position".

1. Position fixe – {postion: fixed;} : on parle des bords *de la fenêtre*. L'élément ne défile donc pas avec la page, il est *figé* sur *la fenêtre*. Dans l'exemple proposé, le bord gauche de l'élément sera distant de 5 pixels du bord gauche de la fenêtre, et de 10 pixels du bord supérieur de la fenêtre.

2. Position absolue – {postion: absolute;} : on parle des bords *de la page*. L'élément défile avec la page, il est *figé* sur *la page*. Dans l'exemple proposé, le bord gauche de l'élément sera distant de 5 pixels du bord gauche de la page, et de 10 pixels du bord supérieur de la page.

3. Position static – {postion: static;} : positionnement "*automatique*", dans le flux (x)html, ne tenant pas compte d'éventuelles coordonnées. La position est alors essentiellement définie par la propriété "display" et les valeurs d'espacement. Dans l'exemple proposé, comme dans tout positionnement statique, les valeurs de "left" et de 'top" ne sont pas prises en compte.

4. Position relative – {postion: relative;} : le positionnement relatif tient compte du contexte, *i.e.*, des *des éléments voisins*. L'élément défile avec la page, il est *placè* en-haut/en-bas, à gauche/à droite de ce qui le précède et ce qui le suit. Plus précisément, le positionnement relatif apporte *une correction* du positionnement *statique*. Dans l'exemple proposé, le bord gauche de l'élément sera placé 5 pixels *plus à droite* et 10 pixels *plus en bas* qu'il ne l'aurait été en positionnement *statique*.

6.3 Profondeur

Lorsqu'un élément a un positionnement fixe ou absolu, il peut se retrouver superposé avec d'autres éléments (par exemple, une image de fond dans le "body" est en superposition avec tout élément de la page). Il faut alors déterminer la *profondeur* de l'élément, via la propriété "z-index" (pour reprendre l'exemple, l'image de fond du "body" est toujours en arrière plan, soit de profondeur supérieure à tout élément de contenu !). La profondeur est gérée par "z-index" de façon ordinale, de la façon suivante :

1. z-index $\in \mathbb{Z}$

2. valeur par défaut z-index == 0 ;

3. plus le z-index est *grand*, plus l'élément concerné sera de *premier plan*.

6.4 Habillage

L'habillage définit comment une image ou un texte se situe à l'intérieur de son élément parent (vis-à-vis des autres éléments de son parent). Prenons l'exemple d'une image (une illustration par exemple) entourée de texte : ou peut vouloir que le texte de situe *au-dessus* et *en-dessous* de l'illustration (affichage de type bloc), comme on peut vouloir que l'image et le texte s'affichent *au même niveau*, par exemple, que l'image soit *à gauche* (valeur de float à "left") ou *à droite* (valeur de float à "right") du texte.

Plus précisément, donner la valeur "left" ou "right" à la propriété "float" d'un élément (au lieu de "none" par défaut) fait que l'élément génère une boîte de bloc qui flotte à gauche ou à droite (de son contenant) ; le reste du contenu de son élément parent (le flux (x)html) s'écoule alors à sa droite ou à sa gauche.

Illustration : on souhaite afficher les titres à gauche de la page, dans une boîte flottante (au lieu d'une boîte de type bloc). il suffit d'écire dans le css :

```
h2 {float: left; margin-right: 20px;}
```

Si le fichier (x)html est le code suivant :

```
<h2>Cursus</h2>
<p>bla bla bla ... bla bla bla</p>
```

Il s'affichera :

Cursus bla bla bla ...
bla bla bla ...bla bla bla ...

Au lieu de l'affichage "classique" (par défaut, les éléments "h2" et "p" générant des boîtes de type bloc) :

Cursus
bla bla bla ...
bla bla bla ...bla bla bla ...