Extrait du livre : CSS avancées - Vers HTML 5 et CSS 3 | R a p h a ë l G o e t t e r

- **commentaire**
- **Propriété, valeur et déclaration**
- **Sélecteur**
- **Sélecteur de classe**
- **Sélecteur d'identifiant**
- **Règle et bloc de déclaration**
- **Pseudo-classe et pseudo-élément**
- **Ancêtre, parent, frère**
- **Priorité des sélecteurs**
- **Sélecteurs avancés**

Commentaire

Les commentaires en CSS sont des instructions facultatives commençant par les caractères `/*` et se terminant par `*/` (figure 2-1). On peut les placer partout entre les règles, voire au sein des règles (mais pas entre une propriété et sa valeur), leur contenu n'ayant aucune influence sur le rendu. On ne peut pas les imbriquer.

Leur usage est principalement informationnel, dans le cadre de projets en commun et dans l'optique de justifier tel ou tel choix de déclaration CSS, ou encore de délimiter les différentes parties de la feuille de styles.

Figure 2-1

Illustration d'un commentaire CSS

```
header nav a:hover, header nav a:focus {
  background: #E4E9ED;
  background: rgba(255,255,255,0.3);
  text-decoration: none;
  -moz-border-radius: 10px;
  -wekbit-border-radius: 10px;
  border-radius: 10px;
  -moz-outline-radius: 10px;  /* a11y : on arrondit le focus, si un focus carré pose problème */
  -webkit-outline-radius: 10px;
  outline-radius: 10px;
  outline: none;
}
```

Propriété, valeur et déclaration

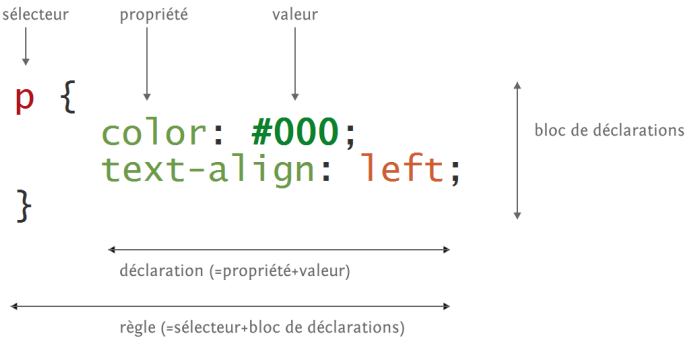
Une *propriété* est une syntaxe définie explicitement par les spécifications CSS et définissant la mise en forme de l'élément auquel elle s'applique (figure 2-2). On peut regrouper les propriétés par type (typographie, positionnement, modèle de boîte), mais aussi par type de média compatible (écran, imprimante, mobile, auditif, projection).

La version 1 de CSS comportait un total de 53 propriétés, tous types confondus. CSS 2.1 en propose 115 et CSS 3 en admet 246 à l'heure actuelle (source : http://meiert.com/en/indices/css-properties/).

Les normes CSS spécifient quelles sont les *valeurs* autorisées pour chaque propriété. Ces valeurs peuvent être construites à partir d'identificateurs, de chaînes de caractères, de nombres, de longueurs, de pourcentages, d'adresses URI, de couleurs, d'angles, de durées, etc.

Figure 2-2

Visuel de la syntaxe d'une règle CSS



Une propriété dont la syntaxe n'est pas homologuée ou dont la valeur n'est pas valide est a priori ignorée par les navigateurs. Cependant, certains d'entre eux offrent en réalité une tolérance qui peut être exploitée sous forme de *hack*, c'est-à-dire un détournement volontaire d'une propriété ou d'une valeur pour cibler une version de navigateur déficiente et lui appliquer des styles particuliers.

Le terme *déclaration* désigne un ensemble formé par la propriété et sa valeur associée, par exemple `font-style: italic`. On peut regrouper plusieurs déclarations en les séparant par des points-virgules, par exemple `font-style: italic; background-color: #ffc`.

> **BONNE PRATIQUE** **Dernière déclaration**
>
> La dernière déclaration d'un bloc ne nécessite pas de séparateur (`;`), mais pour des raisons de rigueur, je vous recommande de le signaler, ne serait-ce que pour éviter une erreur dans le cas où vous ajouteriez une déclaration par la suite.

Sélecteur

Le terme de sélecteur désigne la partie précédant le bloc de déclaration, c'est-à-dire à gauche de l'accolade ouvrante d'une règle CSS. Un sélecteur doit d'ailleurs toujours être accompagné d'un bloc de déclaration.

Le sélecteur indique l'élément sur lequel vont s'appliquer les déclarations CSS présentes dans le bloc qui lui est associé. Il peut revêtir plusieurs formes : sélecteur d'élément (exemple : `p`), sélecteur de classe (exemple : `.fruit`), sélecteur d'identifiant (exemple : `#kiwi`), sélecteur d'attribut (ex : `[href]`), etc.

Plusieurs sélecteurs peuvent précéder un bloc de déclaration. Il suffit de les séparer par une virgule. Le bloc sera alors appliqué à chacun des sélecteurs, comme l'illustre l'exemple suivant :

```
h1, h2, h3 { font-weight: bold;}.
```

BONNE PRATIQUE Attention à la syntaxe !

Un sélecteur ne peut être composé que de caractères alphanumériques, de tirets (-) et de caractères *souligné* (_). Les caractères spéciaux ne sont pas valides et – excepté en HTML 5 – il n'est pas autorisé de commencer un sélecteur par un chiffre ou un tiret.

Sélecteur de classe

Une *classe* est un nom que l'on choisit librement (en se limitant aux caractères alphanumériques classiques) et dont on baptise les éléments concernés via l'attribut HTML `class`. Un sélecteur de classe reprend ce nom en le préfixant d'un point (par exemple : `.error`, `.bloc`, etc.) :

Partie HTML

```
<span class="error">formulaire incomplet</span>
```

Partie CSS

```
.error {color: red;}
```

Sélecteur d'identifiant

Un *identifiant* (ou `id`) est lui aussi un nom choisi librement (en se limitant aux caractères alphanumériques classiques). Il se distingue de la classe en ce qu'il ne peut porter au plus que sur un objet du document.

Les sélecteurs CSS s'y réfèrent par l'emploi d'un caractère dièse (`#`) suivi de ce nom (exemples : `#header`, `#nav`, `#contact`, etc.).

Partie HTML

```
<ul id="nav">...</ul>
```

Partie CSS

```
#nav {margin: 0; padding: 0;}
```

L'identifiant désigne un élément unique dans le document (par exemple, `#kiwi`), contrairement aux classes pouvant symboliser plusieurs types d'éléments ou des groupes d'éléments (par exemple, `.fruit`) : l'élément dont l'`id` est `kiwi` sera unique, mais plusieurs éléments peuvent appartenir à la classe `fruit`.

Prenez donc le réflexe d'attribuer un identifiant aux objets uniques de votre code ou aux grands blocs qui structurent la page. On trouve souvent un bloc d'en-tête (`#header`), le pied de page (`#footer`), la partie latérale (`#aside`), la partie principale (`#main`), le menu de navigation (`#nav`), etc. Les éléments susceptibles d'apparaître plusieurs fois dans un document HTML seront quant à eux qualifiés à l'aide de classes.

Règle et bloc de déclaration

Une règle CSS se compose d'un sélecteur suivi de son bloc de déclaration, ce dernier débutant par une accolade gauche ({) et se terminant par l'accolade droite (}) correspondante et contenant une ou plusieurs déclarations.

Au final, une feuille de styles est un fichier de type textuel contenant l'ensemble des règles CSS.

Pseudo-classe et pseudo-élément

Dans l'optique de mettre en forme des éléments selon un état contextuel (survol, premier enfant, première ligne…) qui ne serait pas défini dans l'arbre du document, CSS introduit les concepts de pseudo-classe et de pseudo-élément.

La différence entre les pseudo-classes et les pseudo-éléments étant plutôt ténue, d'autant plus qu'un certain nombre d'exceptions en compliquent encore la compréhension, je vais par facilité regrouper ces deux termes sous la même dénomination de « pseudo-élément » tout au long de cet ouvrage, sauf dans les cas où la distinction sera pertinente.

EXEMPLES Pseudo-classes et pseudo-éléments

Parmi les pseudo-classes les plus couramment usitées, retenons les différents états des liens hypertextes (`:link`, `:visited`, `:hover`, `:focus` et `:active`) et le premier enfant d'un élément (`:first-child`). Du côté des pseudo-éléments, mentionnons plus particulièrement la première lettre d'un bloc (`:first-letter`), sa première ligne (`:first-line`) ou le contenu créé avant (`:before`) ou après (`:after`) un élément.

L'exception :visited

Les dernières générations de navigateurs (à partir de Firefox 3.6, Chrome 5 et Safari 4.0.5) viennent subitement de restreindre considérablement l'éventail des propriétés CSS applicables à la pseudo-classe `:visited`, vieille comme le Web et désignant un lien que l'on a déjà suivi. Les seules propriétés dorénavant tolérées sur cet élément se limitent à la définition des couleurs (`color`, `background-color`, `border-color`, `outline-color`, `column-rule-color`, `fill` et `stroke`).

Cette restriction imposée par les navigateurs est due à une vulnérabilité de `:visited` découverte très récemment et permettant d'exploiter l'historique de navigation d'un visiteur. Le détail de cette faille est expliqué sur le blog du développeur principal de Mozilla, David Baron, à l'adresse :

▸ http://dbaron.org/mozilla/visited-privacy

En traitant ces données à l'aide des technologies dynamiques actuelles, il devient possible de récupérer un grand nombre d'informations véhiculées via les URL visitées et de traiter un vaste champ de données personnelles telles que :

- les visites de liens dits « sensibles » (banques, sites politiques, religieux ou pour adultes) ;

- les recherches effectuées sur les moteurs (exemple : http://www.google.fr/search?q=ma+recherche) ;
- votre réseau social dans sa globalité (Qui sont vos contacts sur Facebook, LinkedIn ou Twitter ? Où habitent-ils ? Quel est leur profil ?) ;
- des informations relatives à votre vie privée, votre nom, vos coordonnées, éventuellement votre état de santé, votre orientation politique, etc.

Si vous n'avez pas la chance de disposer d'une très récente version de navigateur, sachez que vous êtes exposé à ces risques de confidentialité.

EN SAVOIR PLUS **Faille de confidentialité**

Un site Internet d'information anglophone sur ce sujet vous permettra d'en savoir plus à propos de cette faille et son exploitation en pratique :

▸ http://whattheinternetknowsaboutyou.com

Généalogie

Les différents éléments composant un fichier HTML (titres, paragraphes, listes, liens, images…) s'organisent sous la forme d'un « arbre généalogique », que l'on nomme également « arbre du document » (ou DOM pour *Document Object Model*). Cet arbre généalogique est composé de fratries et de degrés de parenté qui sont exploités par les sélecteurs CSS pour cibler les différents éléments du document.

Ancêtre, parent, frère

Choisissons arbitrairement un élément situé au sein de l'arborescence. Cet élément, que nous nommerons X pour préserver son anonymat, est susceptible d'être entouré d'éléments ancêtres, descendants, parents, enfants et frères.

Au sein de cette arborescence, le terme d'*ancêtre* désigne tout élément situé dans la même branche que l'élément X, mais à un niveau supérieur dans la hiérarchie. Le *parent* est un élément unique qui fait référence à l'élément placé directement au dessus de X.

À l'inverse, le concept de *descendant* fait référence à tout élément qui est situé dans la même branche que X, mais à un niveau plus bas dans la hiérarchie. Les *enfants* sont les éléments placés directement au dessous de X.

Tous les éléments situés au même niveau que X et partageant le même parent sont appelés *frères*.

Influence sur les sélecteurs

Pour des raisons de compatibilité avec les navigateurs, nous avions jusque-là coutume de n'employer que des sélecteurs très basiques se limitant généralement aux noms de classes, d'identifiants ou aux ancêtres et descendants.

Cependant, CSS 2.1 et CSS 3 proposent un vaste panel de sélecteurs avancés ciblant bien plus spécifiquement les éléments selon leur hiérarchie, ce qui évite d'attribuer à outrance des noms de classes sur de nombreux éléments. Grâce à ces « nouveaux » sélecteurs enfin utilisables depuis Internet Explorer 7, nous allons par exemple pouvoir atteindre les éléments frères ou les enfants directs.

Priorité des sélecteurs

Il est reconnu que lorsque deux règles CSS différentes ciblent le même élément, c'est la dernière déclarée (la plus basse dans le code source) qui s'applique et qui écrase la précédente. En pratique, cette loi ne vaut que dans le cas où les deux sélecteurs incriminés sont exactement du même type et déclarés de la même manière, car il existe une priorité entre les différents types de sélecteurs.

Mode de déclaration

Je ne vais pas vous apprendre qu'il existe plusieurs façons d'appliquer des styles CSS à un élément HTML. En revanche, vous ne saviez peut-être pas que chaque méthode a une priorité d'application différente :

1. Style en ligne : on applique le style directement au sein de la balise HTML, via l'attribut `style=`. Ce mode est prioritaire sur tous les autres.
2. Déclaration dans l'en-tête du document : on emploie la balise HTML `<style>` que l'on place au sein de l'élément `<head>` de la page HTML. Ce mode est prioritaire sur les feuilles de styles externes.
3. Feuille de styles auteur : la mise en forme est complètement externalisée dans un fichier de styles CSS lié à l'aide de `<link>` ou `@import`.

> **À RETENIR Feuille de styles utilisateur**
>
> Aux styles définis par l'auteur du site web s'ajoute la possibilité, pour chaque internaute, de créer des styles personnalisés dans la feuille de styles « utilisateur », qui toutefois demeurera en retrait par rapport à celle proposée par le concepteur du site.

Poids des sélecteurs

Au sein du même mode de déclaration, les spécifications CSS proposent une classification des sélecteurs selon un barème de poids sous forme de notation à quatre chiffres :

1. Poids « a » : règle CSS déclarée à l'aide de l'attribut HTML `style=` au sein de l'élément.
2. Poids « b » : sélecteur d'identifiant (`id`).
3. Poids « c » : sélecteur de classe, d'attribut (`[]`) ou de pseudo-classe (`:hover`, `:focus`, `:first-child`).

4. Poids « d » : sélecteur d'élément (`p`, `div`, `a`,…) ou de pseudo-élément (`:first-letter`, `:before`, `:after`, `:first-line`).
5. Poids nul : sélecteur joker (`*`), de parenté (`>`) ou d'adjacence (`+`).

L'explication de ce barème est simple : une déclaration qui s'applique à un sélecteur d'`id` (exemple : `#kiwi {color: green;}`) aura toujours un poids supérieur à la même déclaration appliquée à un sélecteur de classe (exemple : `.kiwi {color: lime;}`), qui elle-même sera prioritaire sur une déclaration visant un élément en particulier (ex : `h1 {color: red;}`).

En cas de conflit, il est alors parfois nécessaire « d'ajouter du poids » à votre sélecteur en y incluant un type de sélecteur supplémentaire.

Prenons un exemple concret en observant la priorité des styles dans le cas du lien hypertexte du code suivant.

Partie HTML

```
<div id="warning">
  <a class="error" href="url">source de l'erreur</a>
</div>
```

Supposons que les règles CSS appliquées soient les suivantes.

Partie CSS

```
a {color: green;} /* 1 sélecteur d'élément (poids « d ») */
#warning a {color: blue;} /* sélecteur d'identifiant (poids « b ») + 1 sélecteur de
➥ poids « d » */
a.error {color: red;} /* 1 sélecteur de classe (poids « c ») + 1 sélecteur de poids
➥ « d » */
```

La couleur du lien indiquant une erreur sera bleue car la règle appliquée est `#warning a {color: blue;}`. Bien que la règle `a.error {color: red;}` lui succède dans l'ordre du code CSS, le sélecteur d'identifiant demeure prioritaire.

Si je souhaite que la couleur du lien soit rouge, je dois ajouter un poids au moins équivalent à celui du sélecteur `a.error`. Concrètement, j'écrirai `#warning a.error`.

> **BONNE PRATIQUE** **Sélecteurs « à rallonge »**
>
> Attention à ne pas verser dans l'extrême dans l'apport de poids, afin de ne pas vous retrouver avec des accumulations de multiples types de sélecteurs. Identifiez au plus court les éléments à mettre en forme, sans sélecteurs intermédiaires inutiles, et ce, en amont de votre projet de conception CSS.

!important

La déclaration `!important` a été introduite par CSS dans le but d'outrepasser volontairement la priorité conférée par défaut aux modes de déclaration que sont les feuilles de styles auteur et utilisateur. Dans la pratique, une déclaration suivie du mot-clé `!important` devient préférentielle,

quel que soit le poids du sélecteur qui l'accompagne : les styles marqués ainsi écrasent d'office tous les styles similaires antérieurs.

Pour reprendre notre exemple précédent, nous pouvons forcer l'application de la couleur rouge à nos liens de classe `error` en agissant de la sorte :

```
a.error {color: red !important;}
```

Quelle que soit la place d'apparition de cette règle dans la feuille de styles et quel que soit le poids de son sélecteur, les styles seront affectés en priorité… sauf si une autre règle équivalente revendique elle aussi sa préséance à l'aide d'une déclaration `!important`, auquel cas, la règle habituelle du poids des sélecteurs s'applique.

APPLICATION PRATIQUE **!important**

L'usage du mot-clé `!important` est courant dans les scripts JavaScript, Ajax ou jQuery que vous pouvez rencontrer sur Internet, tout simplement parce que ces scripts doivent pouvoir s'appliquer sur n'importe quelle page web sans en connaître la teneur et les styles déjà présents. Cependant, je ne vous recommande guère de l'employer sans en maîtriser toutes les conséquences : sa présence est généralement un indicateur d'une feuille de styles brouillonne avec une gestion hasardeuse des poids des sélecteurs.

Sélecteurs et pseudo-éléments CSS 2.1

Les versions CSS 2 et CSS 2.1 élaborent de nouveaux types, mal connus, de sélecteurs et de pseudo-éléments. La grande majorité de ces concepts est acquise par tous les navigateurs actuels et, du côté de Microsoft, depuis la version Internet Explorer 7 ; autant dire que nous pouvons d'ores et déjà les manipuler, à condition de maîtriser un tant soit peu leur alternative sur le vénérable ancêtre IE6. Seules exceptions de taille dans ce tableau idyllique, les pseudo-éléments `:before` et `:after` autorisant la création de contenu via CSS ne sont pris en compte qu'à partir d'Internet Explorer 8.

Sélecteur d'enfant

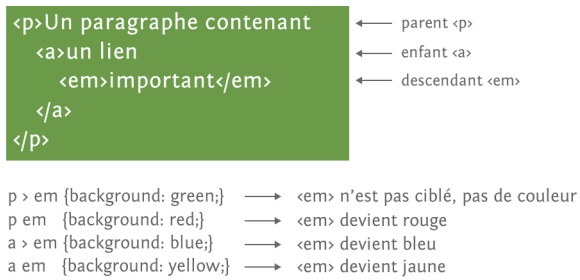
Le sélecteur d'enfant s'applique, comme son nom le laisse présager, à l'enfant ou aux enfants d'un élément désigné (figure 2-3). Sa syntaxe est la suivante :

```
E > F {propriété: valeur;}
```

À l'instar des autres sélecteurs, celui-ci se lit de la droite vers la gauche. Ainsi notre exemple va-t-il appliquer la déclaration à l'élément `F` s'il est enfant d'un élément `E`. À la différence du sélecteur classique de descendance, symbolisé par un espace entre les éléments (`E F`), le sélecteur d'enfant (`E > F`) ne s'applique qu'en cas de parenté directe et n'aura aucune prise si un élément se trouve hiérarchiquement entre `E` et `F`.

Figure 2-3

Exemple de sélecteur d'enfant



Au premier abord, l'intérêt de cette syntaxe peut paraître mineur. Il est pourtant très pratique dans le cas de documents emboîtant plusieurs données du même type, par exemple des listes imbriquées.

Partie HTML

```
<ul id="menu">
  <li><a href="url1">accueil</a></li>
  <li><a href="url2">société</a></li>
  <li><a href="url3">contact</a>
    <ul class="submenu">
      <li><a href="url3-1">plan d'accès</a></li>
      <li><a href="url3-2">formulaire de contact</a></li>
      <li><a href="url3-3">réseaux sociaux</a></li>
    </ul>
  </li>
</ul>
```

Partie CSS

```
#menu > li {list-style-type: square;}
```

Notre exemple illustre un menu de navigation à deux niveaux. Afin de ne cibler que la première génération de la liste (les enfants directs de l'élément identifié comme `menu`), j'ai choisi le sélecteur d'enfant. Ainsi, les éléments `<li>` contenus dans `submenu` ne seront pas affectés et conserveront leurs puces par défaut.

Il y a fort à parier que ce type de sélecteur prenne de l'importance avec la promotion de HTML 5, car ce dernier envisage de larges possibilités d'imbrications d'éléments, notamment des balises `<h1>` pour chaque section ou article, eux-mêmes pouvant être incorporés dans d'autres sections.

COMPATIBILITÉ Sélecteur d'enfant

Le sélecteur d'enfant est pris en charge par tous les navigateurs, à partir d'Internet Explorer 7.

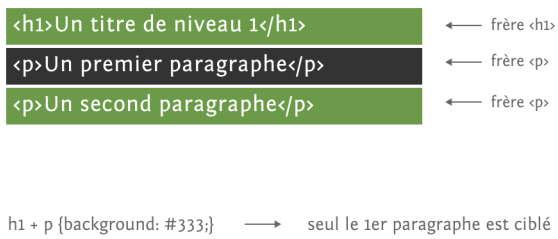
Sélecteur de frère adjacent

Le sélecteur de frère adjacent (ou sélecteur d'adjacence) affecte un élément à condition qu'il soit immédiatement précédé d'un autre élément spécifié (figure 2-4). Sa syntaxe est la suivante :

```
E + F {propriété: valeur;}
```

Figure 2-4

Illustration du sélecteur de frère adjacent



Cette règle s'applique à l'élément `F` s'il est frère de `E` et s'il lui succède dans l'ordre de déclaration HTML. Illustrons ce concept par un exemple simple constitué d'un titre de niveau 1 suivi par deux paragraphes.

Partie HTML

```
<h1>Le kiwi</h1>
<p>Le kiwi est un fruit, mais c'est aussi un animal</p>
<p>Le kiwi le plus répandu est le fruit de Actinidia deliciosa</p>
```

Partie CSS

```
h1 + p {font-style: italic;}
```

Dans notre exemple, seul le premier paragraphe `<p>`, directement adjacent au titre `<h1>`, se distinguera en italique. Les paragraphes suivants demeureront inchangés.

Visualiser le résultat en ligne

▸ http://www.ie7nomore.com/fun/columns/

Seul le paragraphe directement adjacent au titre principal diffère des suivants.

Ce type de sélecteur a pour avantage de cibler un groupe d'éléments identiques tout en excluant le premier. Par exemple, pour afficher une bordure à gauche de tous les éléments d'une liste sauf le premier, il suffit d'indiquer la règle suivante :

```
li+li {border-left: 1px solid black;}
```

Nous verrons un peu plus loin que CSS 3 étend la portée de ce sélecteur en l'appliquant à la fratrie indirecte.

COMPATIBILITÉ Sélecteur de frère adjacent

Le sélecteur de frère adjacent est compris par tous les navigateurs à partir d'Internet Explorer 7.

Sélecteur d'attribut

Le sélecteur d'attribut apparaît avec la spécification CSS 2.1. Il consiste en une extension du sélecteur d'élément, avec la possibilité de cibler un attribut HTML, la valeur d'un attribut, voire une partie de cette valeur.

Sa syntaxe est la suivante :

```
[attribut] {propriété: valeur;}
```

On peut éventuellement préciser l'élément concerné à l'aide de la syntaxe suivante :

```
E[attribut] {propriété: valeur;}
```

Il est ainsi possible de cibler, par exemple, uniquement les images disposant d'un attribut `alt`, à l'aide du sélecteur CSS `img[alt]`, et de les styler comme bon nous semble.

Le sélecteur d'attribut offre également l'opportunité de préciser la valeur attendue de l'attribut ciblé : le sélecteur `a[hreflang="fr"]` va s'appliquer à tous les liens dont l'attribut `hreflang` contient exactement la valeur `fr`. Pour ne cibler que les cellules de tableaux dont l'attribut `colspan` vaut `3`, il suffit d'écrire `td[colspan="3"]`.

Notez qu'il est parfaitement envisageable de cumuler deux sélecteurs d'attribut, comme le décrit l'exemple suivant :

```
img[class="vignette"][width="250"] {border: 1px solid green;}
```

Cette règle aura pour but d'ajouter une bordure verte autour des éléments d'image à condition que leur attribut `class` ait la valeur `vignette` et que leur attribut `width` vaille `250`.

Les guillemets entourant la valeur de l'attribut ne sont pas nécessaires, ni demandés par les spécifications, mais l'expérience a démontré une meilleure reconnaissance du sélecteur lorsqu'il en dispose.

Nous verrons dans le chapitre 8 que la version 3 de CSS ajoute encore de nouvelles options à ce sélecteur, permettant de cibler les valeurs débutant ou se terminant par une chaîne de caractères définie.

COMPATIBILITÉ Sélecteur d'attribut

Le sélecteur d'attribut est supporté par tous les navigateurs à partir d'Internet Explorer 7, mais avec des lacunes selon les syntaxes. En effet, dans certaines situations, IE7 ne reconnaît pas certains sélecteurs d'attributs si le nom de l'élément n'y est pas accolé : `input[type="text"]` sera globalement mieux interprété que `[type="text"]`.

:first-letter et :first-line

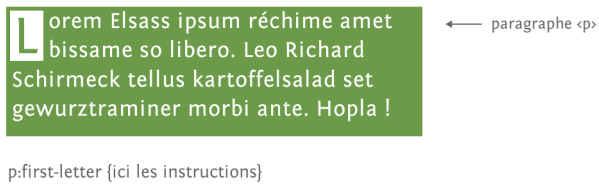
Les pseudo-éléments `:first-letter` et `:first-line` correspondent respectivement au premier caractère et à la première ligne de texte au sein d'un élément de type bloc ou équivalent.

Créés dès CSS 1, ces pseudo-éléments comptent encore aujourd'hui un certain nombre de différences d'appréciation minimes entre les navigateurs, bien qu'ils soient globalement bien reconnus.

`:first-letter` est traditionnellement utilisé pour mettre en exergue une lettrine, à l'instar de certains anciens ouvrages typographiques, comme le montre la figure 2-5.

Figure 2-5

Création d'une lettrine



Code CSS correspondant à l'élaboration d'une lettrine

```
p:first-letter {
  float: left; /* positionnement de la lettrine dans le conteneur */
  font: bold 3em Georgia, "Times New Roman", Times, serif;
  color: #900;
  padding: 3px 8px;
  margin: 5px 5px 0 0;
  border: 1px solid #900;
}
```

La spécification CSS 2 liste les propriétés qui peuvent être appliquées à `:first-letter`. Il s'agit des propriétés de police, marges externes (`margin`), marges internes (`padding`), bordures, couleur, arrière-plan, ainsi que les propriétés `text-decoration`, `text-transform`, `letter-spacing`, `word-spacing`, `line-height`, `float` et `vertical-align` (seulement si `float` a pour valeur `none`).

La norme indique que les agents utilisateurs peuvent prendre en charge d'autres propriétés, notamment de positionnement, s'ils le souhaitent.

COMPATIBILITÉ :first-letter et :first-line

Pris en charge depuis longtemps, ces deux pseudo-éléments présentent un certain nombre d'erreurs célèbres sur l'ensemble des navigateurs. La première est qu'Internet Explorer 6 ne va pas interpréter la règle lorsque le bloc de déclaration est collé au séparateur : `p:first-letter{propriété: valeur}` sera invisible pour IE6, contrairement à `p:first-letter {propriété: valeur}`.

Il est également connu, toujours au chapitre Internet Explorer, que ces pseudo-éléments ne vont pas s'appliquer à tous les éléments (voir partie consacrée à la résolution de bogues au chapitre 6).

:first-child

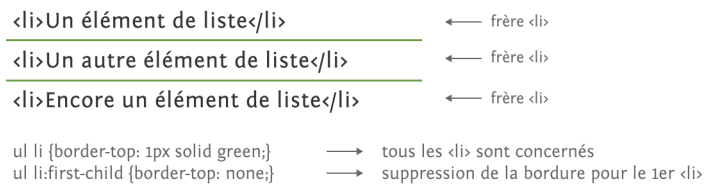
La pseudo-classe `:first-child`, définie depuis CSS 1, désigne le premier élément enfant au sein d'un élément (figure 2-6). Par exemple, `li:first-child` qualifie le premier élément `<li>` d'une liste `<ul>` ou `<ol>`, *et non pas le premier élément contenu dans* `<li>` *comme beaucoup sont tentés de le croire.*

Ce sélecteur rend très commode l'identification du premier élément parmi des frères semblables, comme le montre cet exemple qui a pour but d'appliquer une bordure sous les éléments d'une liste, sauf le premier :

```
ul li {border-top: 1px solid green;}
ul li:first-child {border-top: none;}
```

Figure 2-6

Illustration de first-child



COMPATIBILITÉ :first-child

`:first-child` n'est reconnue par Internet Explorer qu'à partir de la version 7. C'est pourquoi cette pseudo-classe est encore sous-exploitée malgré les nombreux bénéfices qu'elle apporte.

Par expérience personnelle, j'ai également parfois pu constater que certains navigateurs n'appliquent pas les styles lorsque le parent n'est pas spécifié : je recommande donc d'employer plutôt la syntaxe `ul li:first-child` que `li:first-child`.

:focus

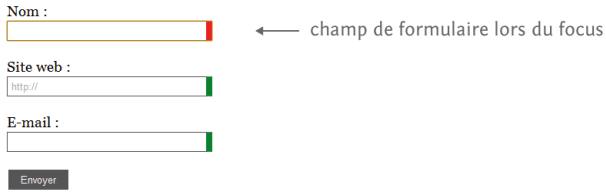
La pseudo-classe `:focus` se rapporte à un élément lorsqu'il est atteint via le clavier ou autre dispositif de pointage (figure 2-7). Selon la spécification, seuls les éléments acceptant l'attribut HTML `tabindex` (qui définit l'ordre de navigation avec la touche tabulation) sont concernés, c'est-à-dire `<a>`, `<area>`, `<button>`, `<input>`, `<object>`, `<select>` et `<textarea>`.

Pour rappel, l'ordre de déclaration des pseudo-classes est important dans le cas des liens hypertextes, pour faire en sorte que toutes s'appliquent :

1. `:link`
2. `:visited`
3. `:hover / :focus`
4. `:active`

Figure 2-7

Focus d'un champ de formulaire sur Chrome



COMPATIBILITÉ :focus

`:focus` est universellement reconnu par l'ensemble des navigateurs à l'exception d'Internet Explorer (seulement à partir de IE8, sauf pour les éléments de formulaires). Cependant, Internet Explorer 6 applique (incorrectement) la pseudo-classe `:active` en lieu et place de `:focus`. Je vous invite par conséquent à systématiquement doubler, voire tripler, vos états de survol en cumulant les sélecteurs `:hover`, `:focus` et `:active` pour le même bloc de règles, ceci pour une plus grande accessibilité de vos liens.

:before et :after

Les pseudo-éléments `:before` et `:after` ouvrent la voie à la création automatique de contenu via CSS, sans que celui-ci ne soit présent dans le document HTML (figure 2-8).

Attachés à un sélecteur, ces pseudo-éléments couplés à des propriétés spécifiques telles que `content` permettent d'afficher une chaîne de caractères, une image d'arrière-plan, des guillemets de citation ou encore des compteurs avant ou après l'élément. Pour être très précis, ce contenu apparaîtra dans la boîte de l'élément, entre les marges internes (`padding`) et le contenu de celui-ci.

La propriété `content` accepte différentes valeurs : toute chaîne de caractères placée entre guillemets simples ou doubles, une image de fond via `url(`*`chemin_de_l'image`*`)`, la valeur d'un attribut récupéré via `attr(`*`nom_de_l'attribut`*`)`, ou encore des mots-clés tels que `counter` (compteur), `open-quote` (guillemets de citation ouvrants), `close-quote` (guillemets de citation fermants), `no-open-quote` ou `no-close-quote` (pas de guillemets ouvrants ou fermants).

L'élément créé via `:before` ou `:after` n'existe pas dans l'arbre de document et a un mode de rendu par défaut de type `inline`, mais peut être mis en forme, dimensionné et positionné à l'aide des propriétés CSS classiques existantes.

ATTENTION Différence de syntaxe entre CSS 2.1 et CSS 3

Depuis CSS 3, la syntaxe de ces pseudo-éléments s'écrit à l'aide d'un double deux-points « : », c'est-à-dire `::before` et `::after`. Cependant, pour des raisons de compatibilité ascendante (l'écriture CSS 3 n'est pas reconnue par Internet Explorer 8 notamment) et parce que les deux syntaxes sont valides, j'ai choisi dans ce livre la syntaxe CSS 2.1 qui est `:before` et `:after`.

Dans les spécifications CSS 3, il est par ailleurs prévu que le contenu puisse être créé sans nécessiter la présence des pseudo-éléments `:before` ou `:after`.

Voici quelques exemples d'usage de ce couple de pseudo-éléments.

Libellé s'affichant automatiquement devant le fil d'Ariane d'un site web

```
#ariane:before {content: "Vous êtes ici : ";} /* à appliquer sur l'élément <ul
id="ariane"> */
```

Séparateurs « > » entre chaque élément du fil d'Ariane

```
#ariane li {display: inline}
#ariane li + li:before { content: "> "; }
```

Guillemets ouvrants au début d'un bloc de citation

```
blockquote:before {content: open-quote;}
```

Guillemets fermants à la fin d'un bloc de citation

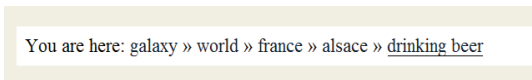
```
blockquote:after {content: close-quote;}
```

Images de fond affichées avant et après chaque paragraphe

```
p:before {
  content: url(images/arrondi_haut.png);
}
p:after {
  content: url(images/arrondi_bas.png);
}
```

Figure 2-8

Fil d'Ariane avec création de contenu



COMPATIBILITÉ :before et :after

Les pseudo-éléments `:before` et `:after` ne sont pas pris en charge par les versions d'Internet Explorer inférieures à IE8. Il est encore quelque peu prématuré de s'en servir, sauf à maîtriser parfaitement les conséquences d'une non-reconnaissance.

À ce propos, l'élément créé n'existant pas dans l'arbre du document, ni sur un agent utilisateur dont les CSS seraient désactivées, ce genre de techniques de création de contenu à la volée est vivement déconseillé pour des raisons évidentes d'accessibilité, à moins qu'il ne soit purement décoratif ou insignifiant.

Valeur de l'attribut attr()

`attr()` est une valeur particulière de la propriété `content` – et donc actuellement liée aux pseudo-éléments `:before` ou `:after` – qui renvoie la chaîne de contenu d'un attribut de l'élément ciblé. Cette valeur dynamique offre la possibilité d'afficher un contenu servant de méta-donnée normalement réservée aux agents utilisateurs.

Parmi les applications courantes, citons l'affichage des URL des liens hypertextes, très pratique au sein d'une feuille de styles d'impression :

```
a:after {
  content: " (" attr(href) ")";
}
```

Pour appliquer la règle précédente exclusivement à l'impression, il est possible de prévoir une feuille de styles dédiée et appelée via l'attribut `media`, ou simplement de placer une *règle* @ (`@media`) en début de feuille de styles classique.

```
@media print
{
  a:after {
    content: " (" attr(href) ")";
  }
}
```

Il est également possible d'employer cette fonction pour afficher la valeur d'un attribut `title` d'un lien, afin de signaler son ouverture dans une nouvelle fenêtre.

Partie HTML

```
<a href="http://www.alsacreations.com" title="s'ouvre dans une nouvelle fenêtre"
target="_blank">Le site d'Alsacréations</a>
```

Partie CSS

```
a[title]:after { /* on ne cible que les liens disposant d'un attribut title */
  content: " (" attr(title) ")"; /* on affiche la valeur de title à la suite du lien */
}
```

Une méthode plus simple consiste à considérer systématiquement les liens munis d'un attribut `target` comme hors de la page actuelle, ce qui nous dispense de l'usage de l'attribut `title`.

Partie HTML

```
<a href="http://www.alsacreations.com" target="_blank">Le site d'Alsacréations</a>
```

Partie CSS

```
a[target]:after { /* on ne cible que les liens disposant d'un attribut target */
  content: " (nouvelle fenêtre)";
}
```

DÉVELOPPEMENTS FUTURS **Possibilités étendues avec CSS 3**

Notez qu'en CSS 3, la valeur dynamique `attr()` a pour vocation de devenir une véritable fonction renvoyant d'autres types de valeurs et qui bénéficiera d'une autre syntaxe plus évoluée. Il sera par exemple possible d'utiliser `attr()` directement comme valeur d'une propriété CSS et pas forcément couplée à `::before`, `::after` ou `content`.

Exemple : `p {float: attr(class)}` sur un élément tel que `<p class="left">` ; dans ce cas, le paragraphe sera flottant à gauche.

Exercice pratique : langue de destination d'un lien

Notre premier cas d'étude concret consiste à agrémenter graphiquement les liens dont la destination vise des pages web dans une autre langue.

HTML propose un attribut destiné à indiquer la langue de destination d'un lien hypertexte : `hreflang`. La pseudo-classe `:after` permet d'afficher la valeur de cet attribut à la suite du lien.

Partie HTML

```
<a href="http://www.alsacreations.com" hreflang="fr">Le site d'Alsacréations</a>
```

Partie CSS

```
a[hreflang]:after { /* on ne cible que les liens disposant d'un attribut hreflang */
  content: " (" attr(hreflang) ")"; /* on affiche la valeur de hreflang à la suite
  ➥ du lien */
}
```

Cette solution demeure quelque peu austère, mais nous pouvons la remplacer avantageusement par une méthode plus graphique, en affichant un petit drapeau de langue :

```
a[hreflang="fr"]:after { /* on ne cible que les liens disposant d'un attribut hreflang
➥ dont la valeur est « fr » */
  content: url(img/flag_fr.png); /* on affiche une image de drapeau */
}
```

Règles @

Les règles @ (ou *règles-at*) sont des instructions pour la plupart définies dès CSS 1 et qui se présentent généralement sous forme de « méta-sélecteurs » permettant de regrouper des blocs de règles au sein d'un bloc de niveau supérieur.

De nombreuses règles @ sont proposées dans les spécifications, mais n'ont pour la plupart qu'un intérêt limité hors cas spécifiques. Je n'ai souhaité développer que les plus intéressantes, à savoir `@import`, `@media` et `@page`. La plus séduisante de toutes, `@font-face`, bénéficiera d'une section de chapitre entière dans la partie du livre dédiée à CSS 3 (chapitre 8).

@import

La règle `@import` inclut une feuille de styles CSS au sein d'une autre feuille de styles. Cette règle doit absolument être placée en tête du fichier, sous peine d'être invalide. Le chemin vers la feuille de styles est indiqué via la valeur `url(styles.css)`. Il est possible de préciser les types de médias concernés par cette feuille de styles au moyen d'un mot-clé placé après l'adresse du fichier externe.