Le langage CSS

Sections

Fixe

Fixe

Introduction

Qu'est-ce qu'une feuille de style ?

Les **feuilles de styles en cascade** (**CSS**, pour *Cascading Style Sheets*) décrivent l'apparence des divers éléments d'une page web par le biais de couples *propriété* / *valeur*. Étant distinctes du code de la page (HTML ou XML), elles constituent un moyen pour séparer structure et mise en page d'un site web. En tant que spécification du W3C, elles obéissent à un ensemble de règles précises qui seront décrites dans les chapitres suivants et que les navigateurs web respectent progressivement.

Si l'on utilise le HTML pour déterminer la présentation dans un navigateur graphique, au lieu de se limiter à structurer le document, il faut alors intégrer les éléments et attributs de présentation au sein du code. Le code s'alourdit inutilement et devient beaucoup plus difficile à faire évoluer. Par exemple, si on veut changer la police (par exemple de type courrier), la couleur (par exemple rouge) et la taille de caractères (par exemple 5 fois la taille par défaut) de chaque paragraphe, en HTML de présentation, il faudrait écrire ceci dans chaque page Web et pour chaque paragraphe :

```
<p>
   <span style="font-family:monospace; color:red; font-size:5em;">
     texte du paragraphe
   </span>
</p>
```

Les feuilles de styles se proposent de résoudre ces deux problèmes par deux approches différentes :

1. En définissant une feuille de style interne au code HTML, on créé un style par page ; ceci est relativement lourd mais parfois intéressant.
2. En définissant une feuille de style externe qui peut alors être utilisée depuis n'importe quel document HTML ou XML.

Cette seconde méthode est la plus courante et la plus adaptée car elle exploite au mieux la faculté de *généralisation* des styles. Elle consiste à créer un fichier externe habituellement d'extension `.css` qui contient les règles de styles des pages. Une déclaration dans l'en-tête de chaque page web (par exemple la partie `head` d'un document HTML) permet ensuite d'appeler ces styles. Le fichier de la page web ne contiendra ainsi que la structure de la page et son style sera appliqué « par dessus » comme une sorte de masque. L'objectif de séparation de la présentation et de la structure est donc atteint.

Éléments et boîtes

En HTML, le code inclus entre une balise d'ouverture et une balise de fermeture est appelé « élément ». Cet élément peut être du texte — un paragraphe (balises `<p>…</p>`), du texte en emphase (balises `<em>…</em>`)… —, un lien (balises `<a href=…>…</a>`), une image (balises `<img src=…>…</img>`), un objet multimédia (balises `<object data=…>…</object>`), un tableau (balises `<table …>…</table>`), une ligne de tableau (balises `<tr …>…</tr>`), la case d'une ligne de tableau (balises `<td …>…</td>`), … Implicitement, chaque objet est inclus dans une boîte, même si, la plupart du temps, cette boîte est invisible.

Le CSS définit le style des boîtes : de leur contenu (alignement, police de caractère, effets…) mais aussi de la boîte en elle-même (couleur de fond, couleur de la bordure, espacements…).

Considérons le code suivant.

```
<p>
  Ceci est du texte comprenant des parties
  en <em>emphase normale</em>
  ainsi que des parties
  en <strong>emphase forte</strong>.
</p>

<p>
  L'exemple montre les boîtes
  qui englobent les éléments.
</p>
```

Nous avons mis en évidence les boîtes englobant les éléments HTML dans le rendu ci-dessous.

Ceci est du texte comprenant des parties en *emphase normale* ainsi que des parties en **emphase forte**

L'exemple montre les boîtes qui englobent les éléments.

Quand utiliser CSS ?

Les CSS permettent de gérer avec une grande efficacité le style des pages web. Il est donc rarement approprié de s'en passer dans une page. Cependant, certains effets CSS peuvent également être obtenus par d'autres biais, et en particulier via javascript. Quels sont alors les critères de choix entre les différentes techniques disponibles ?

Fixe

L'objectif majeur généralement admis est de permettre à tous les visiteurs du site d'accéder à son contenu et à chacun de tirer au mieux parti des effets de présentation et de dynamisme, selon ses contraintes, son matériel, ou son éventuel handicap. On

privilégiera donc le choix technique le plus accessible, profitant au plus grand nombre, en combinant si nécessaire les techniques appropriées.

Ainsi, certains contenus dynamiques peuvent certes être gérés uniquement en CSS, lorsqu'il s'agit notamment de menus déroulants ou de toutes autres boîtes apparaissant au survol de la souris. Cependant, le support des propriétés CSS nécessaires est insuffisant dans une partie des navigateurs, en particulier pour rendre ces contenus accessibles aux personnes qui ne peuvent utiliser une souris ou un dispositif de pointage similaire (utilisateurs handicapés moteurs, n'utilisant que le clavier ou un dispositif similaire). On optera donc plutôt pour des solutions mixtes, combinant javascript et CSS afin de garantir une accessibilité suffisante.

De même, on utilisera avec prudence les techniques purement CSS permettant d'ajouter à l'affichage un contenu qui n'y apparaît pas normalement : la génération de contenu via la propriété `content`, les pseudo-éléments `before` et `after`, ou encore les propriétés d'images d'arrière-plan `background` ne rendront en effet ces contenus accessibles qu'à une partie des visiteurs.

En revanche, les simples changements d'apparence des icônes, couleurs ou fonds au survol de la souris ne posent en général pas de problèmes d'accessibilité lorsqu'ils sont gérés uniquement en CSS : leur absence pour certains utilisateurs ne prive en effet pas d'une information nécessaire à la compréhension du contenu ou à la navigation.

De même, les boîtes dont la position est figée dans la fenêtre du navigateur indépendamment du défilement de la page (*scrolling*) peuvent être réalisées uniquement en CSS, si l'on accepte de limiter cet effet aux navigateurs supportant la propriété requise (`position:fixed`).

Le CSS, tout comme javascript, étant une surcouche technique optionnelle sur le document (X)HTML, tous deux sont susceptibles ne pas être supporté par un navigateur, d'être désactivé par des utilisateurs, ou encore de ne pas être correctement ou suffisamment implémenté pour obtenir l'effet recherché. On veillera donc, dans tous les cas, à :

- Réaliser avant tout un document (X)HTML auto-suffisant, donnant accès à la totalité du contenu et de la navigation
- Puis enrichir l'interface de présentation de ce document via CSS et/ou javascript, sans que cela n'en conditionne l'accès.

Historique

La création des CSS date de 1994. C'est à cette époque que l'essor du web commence avec en particulier l'avènement des publications électroniques. Mais il n'existe encore aucun moyen fiable et robuste pour appliquer un style à ces documents. Håkon Wium Lie, qui travaillait alors au CERN, berceau du web, comprend alors la nécessité de créer une feuille de style spécifique pour le web et en imagine une définition.

Certes, l'idée de séparer la structure de la présentation date de la création en 1990 par Tim Berners-Lee, d'un éditeur/navigateur dans lequel il est possible de définir une feuille de style mais Tim Berners-Lee ne définit pas de syntaxe laissant le soin aux navigateurs de le faire.

Au final, ont été publiés :

- CSS1 en 1996.
- CSS2 en 1998.
- CSS3 en 1999.
- CSS4 en 2010.

Fixe

Premier exemple

Nous allons commencer par vous présenter un exemple simple de feuille de style et un fichier HTML l'exploitant. Ce petit didacticiel permettra d'introduire les notions et le vocabulaire de base.

Fichier HTML

Ce fichier HTML sera l'application de notre feuille de style. Enregistrez-le, par exemple sous le nom `didacticiel-css.html`, et affichez-le dans le navigateur. Regardez son apparence avant la création de la feuille de style.

```
<!DOCTYPE html>

<html lang="fr" xml:lang="fr">

  <head>

  <title>Didacticiel pour le CSS</title>

  <meta
    http-equiv="content-type"
    content="text/html; charset=UTF-8" />

  <link rel= "stylesheet"
    type = "text/css"
    href="essai.css" /> <!-- Inclusion de la feuille de style externe -->

  </head>

  <body>

    <div id="menu"> <!-- Menu -->
      <ul>
        <li><a href="#1">Premier point</a></li>
        <li><a href="#2">Deuxième point</a></li>
        <li><a href="#3">Troisième point</a></li>
      </ul>
    </div>

<!-- Corps du texte -->

    <p>
      Ceci est un fichier <abbr title="Hypertext Markup Language">HTML</abbr>
      utilisant une feuille de style.
    </p>

    <h1><a name="1">Premier point</a></h1>

    <p>
      L'objet de ce fichier est de fournir
      un support au didacticiel <abbr title="Cascading Style Sheets">CSS</abbr>.
    </p>

    <p class="navigation">
      | <a href="#menu">menu</a> |
    </p>

    <h1><a name="2">Deuxième point</a></h1>

    <p>
      Outre mesure, ce fichier ne présente strictement
      <em>aucun</em> intérêt.
    </p>

    <p class="navigation">
      | <a href="#menu">menu</a> |
    </p>

    <h1><a name="3">Troisième point</a></h1>

    <p>
      Nous vous souhaitons une bonne journée.
    </p>

    <p class="navigation">
      | <a href="#menu">menu</a> |
    </p>

  </body>

</html>
```

En l'absence de feuille de style, le navigateur devrait l'afficher comme suit.

- Premier point
- Deuxième point
- Troisième point

Ceci est un fichier HTML utilisant une feuille de style.

Premier point

L'objet de ce fichier est de fournir un support au didacticiel CSS.

| menu |

Deuxième point

Outre mesure, ce fichier ne présente strictement *aucun* intérêt.

| menu |

Troisième point

Nous vous souhaitons une bonne journée.

| menu |

Balises spécifiques

Par rapport à l'HTML sans feuille de style, on remarque :

- la balise de l'en-tête `<link rel= "stylesheet" …>` qui indique où trouver la feuille de style (qui n'existe pas encore) ;
- des références à des styles : `<div id="menu">`, `<p class="navigation">` (les styles « menu » et « navigation » sont décrits dans la feuille de style) ;
- qu'il y a des liens vers l'ancre `#menu` alors que l'on ne l'a pas définie explicitement par un `<a name="menu">…</a>` ; c'est l'attribut `id="menu"` qui joue le rôle de déclaration d'ancre.

Ceci est la syntaxe recommandée, mais au lieu de regrouper les styles dans un fichier .css il est aussi techniquement possible de :

- Les inclure dans le fichier HTML, dans une balise : `<style type="text/css">...</style>` ;
- Les inclure dans chaque balise (à la place des références "id" et "class"), dans un attribut "style". Exemple : `<div style="...">...</div>`.

Arbre du document

On peut construire l'**arbre du document** : chaque *élément*, c'est-à-dire portion du code comprise entre deux balises, se voit appliquer une mise en forme qui est soit la mise en forme par défaut, soit la mise en forme déclarée dans la feuille de style. Une balise est imbriquée dans une autre balise (elles sont toutes imbriquées dans la balise `<body>…</body>`) ; si l'on représente ces imbrications sous la forme d'une arborescence, on voit qu'un élément a un seul parent (est inclus dans une seule balise, qui elle-même peut être incluse dans…).

L'arbre du document est donc ici :

L'astérisque « * » (la « racine » de l'arbre) désigne le document en entier.

Remarquez que nous n'avons considéré ici que le contenu des balises `<body>…</body>`, puisque c'est ce qui va être influencé par la mise en forme. Au sens strict, l'arbre du document devrait commencer par :

Fichier CSS

Copiez le code suivant dans un éditeur de texte (voir la section *Avec quoi écrire un document HTML ?* du wikilivre sur le HTML) et enregistrez-le dans le fichier `essai.css`. Ce fichier doit se trouver dans le même répertoire que le fichier HTML ci-dessus.

```
/* Déclaration 1 : jeu de caractères utilisable */

@charset "UTF-8"; /* ISO Latin-1 */

/* Déclaration 2 : style du menu */

div#menu {
  float:left;                    /* objet flottant à gauche (le reste du texte en fait le tour) */
  width: 10em;                   /* largeur : dix cadratins */
  margin: 0.5em; /* marge de 1/2 cadratin de chaque côté */
  background-color: #f0f8fc;     /* fond bleu clair */
  border: solid blue 2px         /* bordure bleue de 2 pixels tout le tour */
}

/* Déclaration 3 : style de abbr */

abbr {
  border-bottom: dotted red 1px /* bordure de 1 pixel en pointillé en-dessous */
}

/* Déclaration 4 : style de navigation */

p.navigation {
  border-top: solid gray 1px /* bordure grise de 1 pixel au dessus */
}
```

Une fois enregistré, mettez à jour l'affichage de la page HTML dans le navigateur, et regardez la différence.

Les portion de code comprises entre les chaînes `/*…*/` sont des commentaires qui ne sont pas interprétés.

L'exemple devrait ressembler à ce qui suit.

- Premier point
- Deuxième point
- Troisième point

Ceci est un fichier HTML utilisant une feuille de style.

Premier point

L'objet de ce fichier est de fournir un support au didacticiel CSS.

| menu |

Deuxième point

Outre mesure, ce fichier ne présente strictement *aucun* intérêt.

| menu |

Troisième point

Nous vous souhaitons une bonne journée.

| menu |

> **Attention !**
>
> Si les modifications du fichier CSS ne sont pas prises en compte, c'est qu'il contient une erreur de compilation.

Déclarations, règles, sélecteurs et attributs

Le fichier CSS ci-dessus est constitué de quatre **déclarations**. Une déclaration est de la forme :

```
sélecteur { règle-1; règle-2;… }
```

Les espaces et retours de ligne n'ont pas d'importance en CSS, on peut donc écrire de même :

```
sélecteur {
  règle-1;
  règle-2;
  …
}
```

La première déclaration, celle définissant le jeu de caractères disponibles dans le fichier CSS, n'obéit pas à cette syntaxe. On parle de « pseudo-règle ».

Le **sélecteur** est ce qui désigne l'élément du fichier HTML auquel s'applique la déclaration. Par exemple :

- le sélecteur `abbr` désigne tous les éléments `abbr` du fichier HTML (abréviations, comprises entre les balises `<abbr>…</abbr>`) ;
- le sélecteur `p` — qui n'est pas utilisé ici — désigne tous les éléments `p` du fichier HTML (paragraphes, compris entre les balises `<p>…</p>`) ;
- le sélecteur `p.navigation` désigne les éléments `p` de classe « navigation » (paragraphes ouverts par une balise `<p class="navigation">`) ; on peut ainsi définir plusieurs types de paragraphes ;
- le sélecteur `div#menu` désigne l'élément dont l'identifiant est « menu » (groupe d'éléments compris entre les balises `<div id="menu">…</div>`) ; **cet identificateur est unique dans un fichier HTML**, on ne *doit pas* l'utiliser sur plusieurs éléments de cette manière.

On note que les éléments `abbr` disposent déjà d'une mise en forme par défaut, et que l'on redéfinit celle-ci. À l'inverse, les éléments de classe « navigation » ou d'identifiant « menu » sont spécifique à ce fichier HTML.

La description de la mise en forme est comprise entre une accolade ouvrante « { » et une accolade fermante « } ». Elle est constituée d'une série de règles.

Chaque **règle** est de la forme *`attribut : valeur ;`* :

- *`attribut`* est un mot-clef désignant la propriété à modifier ;
- les deux-points « `:` » séparent l'attribut de la valeur qui lui est affectée ;
- un point-virgule « `;` » sépare les règles.

Les attributs que l'on a utilisés ici sont des attributs de boîte : l'objet (le texte ici) est compris dans une boîte, qui peut être invisible (cas général) ou être visible (cas du menu) :

- `float` signifie que l'objet va être calé à un endroit de la page, et que les autres objets (ici le texte) contournent la boîte, l'objet est dit « flottant » ; la règle « `float: left` » crée un objet flottant calé à gauche ;
- `width` désigne la largeur de la boîte ; la valeur utilisée ici, `10em`, se compose d'un nombre (`10`) et d'une unité (`em`, qui désigne le cadratin c'est-à-dire la largeur de la lettre « M », et varie donc selon la police utilisée). Elle peut aussi être définie en %, en px, ou à "auto" pour s'adapter à son conteneur. Par contre, pour adapter la boîte à ton contenu, ne pas utiliser cet attribut mais `display: inline-block` ;
- `margin` désigne la marge (le blanc) laissé entre la boîte et les objets alentours ; on indique successivement la marge en haut, à droite, en bas et à gauche (elles sont ici toutes identiques) ;
- `background-color` désigne la couleur du fond (pour le codage, voir les articles de Wikipédia *Rouge vert bleu* et *Codage informatique des couleurs*) ;
- `border` désigne la bordure ;
  les valeurs `solid blue 2px` (de la déclaration de `div#menu`) indiquent un trait continu (`solid`) de couleur bleue (`blue`) et de deux pixels d'épaisseur (`2px`) ;
- `border-bottom` désigne uniquement la bordure du bas (cela produit une sorte de soulignement, mais le « vrai » soulignement est obtenu autrement, voir ci-après) ; les valeurs « `dotted red 1px` » indiquent un trait en pointillés (`dotted`) de couleur rouge (`red`) et d'un pixel d'épaisseur (`1px`) ;
- `border-top` désigne uniquement la bordure du haut ; les valeurs « `solid gray 1px` » indiquent un trait continu (`solid`) de couleur grise (`gray`) et d'un pixel d'épaisseur (`1px`).

Imaginons que l'on veuille redéfinir le formatage des éléments `em` (texte en emphase, compris entre les balises `<em>…</em>`), pour avoir du texte souligné au lieu d'en italique. On aurait écrit une déclaration

```
em {
  font-style: normal;        /* pas d'italique */
  text-decoration: underline /* soulignement */
}
```

Ajoutez cette ligne au fichier CSS, enregistrez-le et mettez à jour l'affichage de la page HTML pour voir l'effet.

Priorités, cascade et sélecteurs

Comme indiqué, les éléments sont imbriqués les uns dans les autres. Chaque élément fils hérite des propriétés de son parent (l'élément dans lequel il est imbriqué), et ses propriétés propres s'y substituent. Les styles s'appliquent donc en cascade (d'où le nom de CSS, *cascading style sheets*).

La définition du sélecteur est donc capitale : c'est elle qui indique dans quelles conditions un style s'applique ou pas. Ceci est largement évoqué dans le chapitre *Les sélecteurs*, voici toutefois quelques considérations de base.

Élément seul

Si le sélecteur est simplement le nom de l'élément (« `p` », « `em` », « `abbr` »…), ils s'applique à tous les éléments concernés, sauf cas plus précis.

Élément et classe

Si le nom de l'élément est suivi d'un point et du nom d'une classe (par exemple « `p.navigation` »), il ne s'applique qu'aux éléments de la classe du style (`<p class="navigation">…</p>`).

Classe seule

Si le sélecteur ne se compose que du point et du nom de la classe (par exemple « `.navigation` ») il s'applique alors à tous les éléments de cette classe (`<p class="navigation">…</p>`, mais aussi `<h1 class="navigation">…</h1>`, `<ul class="navigation">…</ul>`…).

Élément et identifiant

De même, un sélecteur « `#menu` » s'appliquerait à l'unique élément ayant l'identifiant `menu` du fichier HTML, quel qu'il soit (`<p id="menu>…</p>`, mais aussi `<h1 id="menu">…</h1>`, `<ul id="menu">…</ul>`…) tandis que « `div#menu` » impose que l'élément soit de type `div` (donc `<div id="menu">…</div>`).

Éléments imbriqués

On peut aussi construire un sélecteur à partir de deux autres sélecteurs, séparés d'un signe supérieur « `>` », ou simplement d'un espace. Le sélecteur « *`sélecteur-1 > sélecteur-2`* » ne s'applique que pour les éléments désignés par *sélecteur-2* dont le père est un élément désigné par *sélecteur-1*. Par exemple, « `q > em` » désigne un élément contenu dans des balises `em` et `q` imbriquées comme suit : `<q>…<em>…</em>…</q>` — ainsi, si par exemple une citation est en italiques (avec une déclaration du type `q {font-style: italic; …}`) une mise en emphase dans cette citation sera en lettres romaines (avec une déclaration du type `q > em {font-style: normal; …}`), comme c'est la norme en typographie française.

Exemple

Lorsque trois citations sont imbriquées, la troisième est en italiques, et l'emphase est alors en lettre droites. On a donc :

```
q {quotes: '\00ab\00a0' '\00a0\00bb'} /* guillemets français et espace insécable : s'applique à tous */
q > q > q {font-style: italic}        /* 3e niveau en italiques */
q > q > q > em {font-style: normal}   /* emphase en lettre droites */
```

Application : dans un fichier HTML

```
<p>
  <q> premier niveau, l'<em>emphase</em> est classique, puis
    <q> deuxième niveau, et
      <q> troisième niveau <em>avec emphase</em> au milieu.
  </q></q></q>
</p>
```

ce qui donne

« premier niveau, l'*emphase* est classique, puis « deuxième niveau, et « *troisième niveau* avec emphase *au milieu.* » » »

Voir aussi

- Les styles par défaut du W3C :
  - Chocolate (http://www.w3.org/StyleSheets/Core/Chocolate),
  - Midnight (http://www.w3.org/StyleSheets/Core/Midnight),
  - Modernist (http://www.w3.org/StyleSheets/Core/Modernist),

Fixe

    - Oldstyle (http://www.w3.org/StyleSheets/Core/Oldstyle),
    - Steely (http://www.w3.org/StyleSheets/Core/Steely),
    - Swiss (http://www.w3.org/StyleSheets/Core/Swiss),
    - Traditional (http://www.w3.org/StyleSheets/Core/Traditional),
    - Ultramarine (http://www.w3.org/StyleSheets/Core/Ultramarine).
- Comment déboguer ses CSS :
    - Firefox/Outils aux développeurs

Fixe

Structure et syntaxe

Ce chapitre a pour but de poser les bases de la structure d'une feuille de styles CSS ainsi que les premiers éléments de syntaxe. Nous aborderons également la liaison entre les documents web et les CSS, la propriété de *cascade* des styles et l'adéquation des styles à l'appareil restituant la page web.

Si vous êtes pressé consultez en priorité :

- Règles syntaxiques de base
- Structure générale
- Déclarations de styles

Règles syntaxiques de base

Casse
Les feuilles de styles CSS ne sont pas sensibles à la casse : elles ne tiennent pas compte des majuscules et minuscules. Exception faite pour les éléments n'obéissant pas directement aux règles de syntaxe CSS, notamment les attributs *id* et *class* (dont le nommage est assuré par le rédacteur : vous), les noms des polices de caractères (exemple : "Trebuchet MS"), et les suffixes d'URL ne répondant pas à ces règles.

Mise en forme du code
Les feuilles de styles CSS ne tiennent pas compte des espaces et retours à la ligne.

Identifiants
Les identifiants (nom, *id* et *class*) ne peuvent contenir que des caractères A-Z, a-z, 0-9 plus le tiret (-) et le caractère de soulignement (_). Il ne peuvent pas commencer par un nombre.

Chaînes de caractère
Les chaînes de caractères affichables (par exemple pour les pseudo-éléments `:before` et `:after`, ou pour la propriété `quote`) sont entre des guillemets simples « ' » (« apostrophe ») ou doubles « " ».

- Pour mettre un guillemet simple ou double dans la chaîne affichable, on fait précéder le caractère d'une barre de fraction inversée, respectivement « \' » et « \" ».
- Pour mettre un retour de ligne, on utilise le caractère « \000a » (ou « \a ») ; si l'on veut revenir à la ligne dans le code, on place une barre de fraction inversée seule en fin de ligne.
- En absence de la définition du jeu de caractères *(charset)*, elles ne peuvent contenir que des caractères ASCII ; les caractères Unicode sont obtenus en mettant le code hexadécimal précédé d'une barre de fraction inversée, par exemple « \00a0 » pour une espace insécable, « \0152 » pour « œ » (on peut ignorer les zéros de tête)… Une feuille incluse dans un fichier HTML (entre les balises `<style>…</style>`) utilise le même jeu de caractères que la page HTML. Si la feuille de style est dans un fichier à part, on définit la feuille de code par la règle `@charset` (par exemple `@charset "ISO-8859-1";`).

Commentaires
Les commentaires commencent par une barre de fraction suivie d'un astérique « `/*` », et se concluent par la succession de caractères inverse « `*/` ». Ils sont facultatifs, voire inutiles, pour les modifications mineures d'affichage (inutile d'indiquer que l'on souligne, cela se lit facilement), mais indispensables pour les mises en pages importantes (inscrire par exemple la taille minimale d'une marge pour avoir la place d'insérer le menu permet de ne pas commettre de maladresse lors d'une future modification du fichier).

Structure générale

Syntaxe des règles de style

Une feuille de styles CSS fonctionne sous forme de *déclarations*.

```
selecteur{propriété:valeur;}
```

Une déclaration est composée au minimum de deux éléments : l'élément de la page auquel on souhaite appliquer un style (le **sélecteur**), et le groupe de règles définissant le style (**propriété** et **valeur**). Analysons cette déclaration :

```
h1 { color: red }
```

Ici, l'élément à mettre en forme est h1 (titre de niveau 1) et le groupe de règles, délimité par les accolades, contient la règle « mettre cet élément en rouge ». Une règle consiste en une propriété (ici *color*), suivie par deux points (:), suivie enfin par la valeur associée à la propriété (ici *rouge*).

Il est bien évidemment possible de spécifier plusieurs règles, pour un même élément, en les séparant par des point-virgules (;) de cette façon :

```
h1 {
   color: red;
   font-weight: bold
}
```

On peut aussi spécifier le même jeu de règles pour plusieurs identifiants en les séparant par des virgules (,) :

```
h1, h2 {
   color: red;
   font-weight: bold
}
```

Remarque : la dernière règle du groupe ne comporte pas obligatoirement de point-virgule terminal. Toutefois le mettre systématiquement évite de l'oublier. Il faut savoir que les erreurs de syntaxes CSS ont pour effet d'interrompre dans le navigateur web l'interprétation des styles et donc la mise en forme. Contrairement au moteur d'interprétation du code HTML des navigateurs, l'interprétation des CSS par les navigateurs ne corrige habituellement pas d'erreur de syntaxe.

Modularisation des styles

Il est possible d'importer les styles contenus dans des fichiers de styles différents afin de les organiser de façon modulaire. Parmi les pratiques possibles, on rencontre notamment :

- la scission des styles relatifs à la mise en page, c'est-à-dire le placement des éléments de la page, et des styles relatifs à la typographie (couleurs, bordures, polices, etc.).
- la cascade d'une feuille globale pour un groupe de pages et d'une feuille spécifique à la page concernée (voire une cascade plus complexe prenant en compte les styles par rubriques de pages)

Pour ce faire, on peut notamment utiliser la syntaxe suivante :

```
@import "fichier.css";
```

où *fichier.css* est le nom du fichier contenant les styles à importer. Cette mention doit être spécifiée *au tout début* de la feuille de style, avant d'éventuels styles de la feuille (*déclarations*). Elle doit aussi impérativement comporter un point-virgule (;) final. Elle peut également, si ce n'est pas déjà le cas de la feuille parente où elle se trouve, être complétée par la mention des médias cibles de ces importations.

Remarque : si le chemin du fichier à importer est relatif (pas de barre / devant ni de `http://`), il sera relatif au fichier contenant cette importation. Ici on a donc supposé qu'ils étaient dans le même répertoire.

Utiliser les styles CSS dans une page web

Dans une page HTML

NB: l'annexe Le langage CSS/Interface HTML offre une vue synthétique des exemples exposés ci-après.

Déclaration de styles

Une première méthode pour utiliser des styles CSS consiste à intégrer les styles dans l'entête du fichier HTML à l'aide d'une balise *style*. Le code CSS est alors simplement écrit entre la balise ouvrante et la balise fermante :

```
<html>
  <head>
    <style type="text/css">
      Fixe
```

```
    p {
      font-family: Bitstream Vera Sans;
      color: #666;
      line-height: 1.6em;
    }
  </style>
 </head>
 <body>
  <p>
   Exemple de page HTML avec CSS intégrés
  </p>
 </body>
</html>
```

Les styles CSS ainsi définis ne sont évidemment valides que pour la page en question, on réservera donc en général cette méthode à l'expérimentation ou à des styles propres à une page unique. Pour plus de souplesse, on peut donc transférer le code CSS dans un fichier texte, par exemple *styles.css*, et appeler ces styles dans l'entête HTML à l'aide d'une balise *link*. Ce fichier CSS peut alors être inclus dans toute page web de cette manière :

```
<html>
 <head>
  <link rel="stylesheet" type="text/css" href="styles.css">
 </head>
 <body>
  <p>
   Exemple de page HTML avec CSS externes
  </p>
 </body>
</html>
```

Le fichier *styles.css* contient alors simplement :

```
p {
  font-family: Bitstream Vera Sans;
  color: #666;
  line-height: 1.6em;
}
```

Remarque : le chemin des fichiers CSS spécifiés peut être absolu ou relatif. Dans ce deuxième cas, il est relatif au fichier important la feuille de styles, en l'occurrence le document web. Exemple :

- la page web est `http://mon.site.org/index.html`
- la feuille de styles est `http://mon.site.org/css/styles.css`

On peut alors utiliser une des règles équivalentes suivantes :

```
<link rel="stylesheet" type="text/css" href="css/styles.css">
<link rel="stylesheet" type="text/css" href="./css/styles.css">
<link rel="stylesheet" type="text/css" href="http://mon.site.org/css/styles.css">
<link rel="stylesheet" type="text/css" href="/css/styles.css">
```

Pour être exhaustif, signalons qu'il existe une autre méthode pour inclure du code CSS externe. Elle consiste à utiliser la règle CSS `@import` qui permet d'importer une feuille de style dans un code CSS :

```
<html>
 <head>
  <style type="text/css">
   @import "styles.css";
  </style>
 </head>
 <body>
  <p>
   Exemple de page HTML avec CSS externes
  </p>
 </body>
</html>
```

Cette règle permet, via ses variantes syntaxiques, de filtrer les navigateurs de génération 4.x et 5.0, par exemple Netscape 4 et Internet Explorer 5.0 Mac et/ou Windows. Il faut donc l'utiliser en toute connaissance de cause. Son avantage est que l'on peut facilement déclarer plusieurs feuilles de styles à importer sans multiplier les balises HTML.

Adapter les styles au périphérique de sortie

Le périphérique de sortie de la page web n'est pas forcément l'écran de l'ordinateur. Il peut être également l'imprimante pour faire un tirage papier, un projecteur pour une présentation grand format, ou un navigateur mobile. Les styles définis pour l'écran ne conviennent généralement pas pour d'autres médias. Pour prendre deux exemples fréquents :

- le menu latéral de gauche : est-il vraiment utile à l'impression ? Généralement non, il faudrait donc le faire disparaître dans ce cas.
- la disposition en colonne : est-elle appropriée à un écran réduit tel que celui d'un mobile ? Dans ce cas, il convient d'opter pour une présentation linéaire.

Ceci est possible dans les feuilles de styles grâce à la mention du *media* de sortie. On trouvera la liste des média possibles à la page Interface HTML. Il existe à nouveau deux techniques : la mention directement dans la feuille ou la mention à l'importation d'une feuille externe. Dans le premier cas, les styles CSS contiennent un passage de ce type :

```
@media screen {
  /* règles de styles */
}
```

Ceci définit des styles spécifiques à la sortie écran. Dans le second cas, le plus courant, on définit des fichiers séparés par type de media. Il faut alors ajouter la mention du media dans la page web à l'aide d'un attribut de la balise *link* :

```
<html>
 <head>
  <link rel="stylesheet" type="text/css" href="styles.css" media="screen" />
  <link rel="stylesheet" type="text/css" href="print.css" media="print" />
 </head>
 <body>
  <p>
   Exemple de page HTML avec CSS intégrés
  </p>
 </body>
</html>
```

Dans cet exemple, la feuille *styles.css* contient le style destiné à l'affichage à l'écran dans un navigateur courant, et une seconde feuille *print.css* spécifique à l'impression.

Dans un document XML

On peut importer une feuille de styles CSS dans un fichier XML. Il faut utiliser la règle suivante :

Fixe

```
<?xml-stylesheet type="text/css" href="styles.css"?>
```

L'inconvénient par rapport à l'utilisation d'une feuille de transformation XSL permettant de générer du HTML à partir du XML, est que tous les éléments XML ont la même apparence par défaut. Il faut ainsi redéfinir le style de quasiment chaque élément. Au contraire une transformation en HTML permet d'utiliser les éléments standards du HTML comme les paragraphes, les titres, les listes, etc.

Ordre d'interprétation des styles et cascade

Cascade de styles

Dans le cas où plusieurs feuilles de styles sont utilisées dans la même page web, les règles CSS sont construites au fur et à mesure de la lecture de chacune d'entre elles, dans l'ordre mentionné des feuilles de styles. Ainsi si des règles contradictoires apparaissent dans les styles, c'est la dernière règle lue qui prime.

Considérons par exemple les scripts CSS *styles.css* et *couleurs.css* suivants :

```
/* styles.css */
p {
  color: black;
}
```

```
/* couleurs.css */
p {
  color: gray;
}
```

Les textes des paragraphes seront de couleur grise si ces styles sont importés de cette manière :

```
@import "styles.css";
@import "couleurs.css";
```

alors qu'ils seraient noir en inversant l'ordre d'importation des feuilles de styles. Il faut donc veiller à ne pas dupliquer des règles dans des fichiers CSS différents sous risque de casse-tête par la suite. L'intérêt de cette spécificité de cascade est que les styles par défaut du navigateur sont toujours écrasés par les styles de la page web, lesquels peuvent à nouveau être écrasés par les styles de l'utilisateur lorsque le navigateur le permet. Ainsi si l'utilisateur visite régulièrement un site écrit trop petit, il peut définir une règle agrandissant la police pour ce site.

Remarques :

- toutes les méthodes de déclaration des CSS interviennent dans la cascade lorsqu'elles sont mélangées dans la même page HTML
- la cascade ne s'applique que lorsque c'est exactement le même sélecteur qui est employé ou, bien sûr, un sélecteur de priorité supérieure (lire la suite).

Priorité des règles

Afin d'éviter que le style d'une partie précise du document (par exemple le menu) ne soit facilement modifié par une règle assez générale comme celle du paragraphe précédent, il existe la notion de priorité des règles. Ainsi dans la cascade, une règle apparaissant postérieurement ne peut modifier un style que si sa priorité est égale ou supérieure.

Considérons la feuille de styles suivante dans laquelle nous souhaitons que les titres du menu n'aient pas la même couleur que les titres du texte :

```
h1, h2, h3 {
  color: darkblue;
}
div#menu h1 {
  color: navy;
}
```

Le code HTML correspondant serait :

```
<div id="page">
   <h1>Titre de la page</h1>
   <p>...</p>
</div>
<div id="menu">
   <h1>Section de menu</h1>
   <ul>
     <li>Entrée de menu 1</li>
     ...
   </ul>
</div>
```

Nous supposons ici que les éléments du menu sont tous contenus dans un unique bloc parent, un élément *div* d'identifiant *menu*. Que se passerait-il si une feuille de styles importée par la suite modifiait à nouveau la couleur des éléments de titre HTML par un « `h1 {color: black}` » ? Rien !

Ordre des spécificités des règles

Il est évident dans l'exemple précédent que les titres du menu ne doivent pas être traités comme les autres titres. Aussi, plus le sélecteur d'élément est en quelque sorte *précis* dans sa cible, plus sa priorité est importante. La précision du sélecteur est appelé *spécificité* dans le jargon CSS. L'ordre des spécificités est le suivant, de la moins prioritaire à la plus prioritaire :

1. élément HTML, par exemple « `h1 {color: black}` » (vaut 1 point de priorité).
2. imbrication d'éléments, par exemple « `div h1 {color: black}` » (vaut 2 points de priorité, un par élément présent dans le sélecteur)
3. utilisation d'une classe, par exemple « `h1.noir {color: black}` » (vaut 11 points de priorité : 1 pour l'élément, 10 pour la classe)
4. utilisation d'un identifiant, par exemple « `h1#nom-du-site {color: black}` » (vaut 101 points de priorité : 1 pour l'élément, 100 pour l'identifiant)
5. style dans la balise HTML, par exemple « `<h1 style="color: black">...</h1>` » (vaut 1000 points de priorité)

Un style de moindre priorité ne peut jamais modifier un style de priorité plus élevée. Il existe cependant un moyen de déroger à cette règle : la mention `!important`. Cette mention s'insère juste derrière une valeur de propriété CSS et indique au navigateur qu'il ne faut pas tenir compte de styles contradictoires par la suite. Elle s'utilise de cette manière :

```
p {
  color: gray !important;
}
```

Calcul de spécificité

Le calcul de la spécificité d'une règle se fait en allant des éléments les plus spécifiques vers les moins spécifiques. À chaque pas, le décompte du nombre de sélecteurs correspondant à la spécificité indique le niveau de priorité. En effet, il est normal que :

```
div#page p em
```

soit plus spécifique que :

```
div#page em
```

mais moins que :

```
div#page p em.rouge
```

Ainsi lorsque l'on parcourt les différentes spécificités dans l'ordre décroissant, on obtient :

- « `div#page` » qui met tout le monde à égalité
- « `em.rouge` » qui indique que la dernière règle est prioritaire
- « `p em` » qui place la première en priorité 2
- « `em` » qui place la deuxième en dernière priorité

En corollaire, ceci signifie qu'il vaut toujours mieux **utiliser le moins possible de sélecteurs** pour simplifier l'ajout de règles plus spécifiques. Par exemple dans le code html suivant :

```
<div id="page">
   <p> ...
      <ul>
        <li> <a>...</li>
        ...
      </ul>
   </p>
</div>
```

le style des liens *A* doit vraisemblablement être spécifié en partant d'une règle simple comme « `div#page a` » plutôt que « `div#page p ul li a` » d'emblée. C'est seulement si d'autres liens apparaissent dans la page en dehors des listes que l'on utilisera « `div#page li a` », et ainsi de suite.

Déclaration de variable

Pour éviter de modifier en masse une valeur utilisée à plusieurs endroits, il est possible de la stocker dans une variable, qui sera appelée avec la fonction `var()`.

Exemple :

```
:root {
  --my-super-custom-color: #f6f6f6;
}


#page1 {
  background-color: var(--my-super-custom-color);
}

#page2 {
  background-color: var(--my-super-custom-color);
}
```

Fixe

Les sélecteurs

Introduction

Si vous êtes pressé consultez en priorité :

- Sélecteurs généraux
- Sélecteur de classe
- Sélecteur d'identifiant
- Le sélecteur de descendant
- Les bonnes recettes

Nous avons vu qu'il est facile avec les CSS de changer le style de tous les éléments d'une page web, par exemple mettre en gras tous les titres de niveau 2. Néanmoins, il arrive souvent que le style d'un élément dépende de son contexte. Par exemple un titre de niveau 2 dans le texte de la page ne sera certainement pas mis en forme de la même manière qu'un titre de même niveau dans une section de menu latéral (comme le menu sur la gauche de cette page). Il faut donc un outil pour restreindre le champ d'application d'une règle CSS à une catégorie particulière d'un ou plusieurs élément(s) donné(s) : ce sont les *sélecteurs CSS*.

Un **sélecteur** CSS est un mot-clef qui permet de désigner une catégorie d'éléments de la page éventuellement de nature différente ou une relation entre deux éléments. On pourra par exemple sélectionner tous les titres de niveau 2 dans le menu, ou encore tous les éléments que l'on a marqués comme étant en rouge. Pour ce faire, les sélecteurs CSS trient les éléments de la page selon leur type, certains de leurs attributs, selon leur imbrication dans le code de la page ou encore selon l'interaction avec l'utilisateur pour les liens par exemple (liste non exhaustive).

Il est possible d'utiliser autant de sélecteurs que nécessaire pour spécifier l'élément dont on veut changer le style. On peut donc combiner comme on veut les différents types de sélecteurs qui sont décrits ci-après. Cette caractéristique confère une grande polyvalence aux règles CSS.

Sélecteurs généraux

Sélecteur universel

Le sélecteur étoile (*) sélectionne toutes les balises du document web. Si on souhaite modifier la police de tout le document, on pourra donc employer :

```
* {
  font-family: serif;
}
```

Le sélecteur universel peut trouver d'autres applications dans l'imbrication d'éléments (voir Le sélecteur de descendant plus loin). Il est en effet optionnel lorsqu'il est accolé à un autre sélecteur, ce qui le rend généralement peu utile :

```
*.nom_classe {
  font-family: serif;
}
```

est en effet équivalent à :

```
.nom_classe {
  font-family: serif;
}
```

Sélecteur de type

D'un point de vue de la sémantique employée par le W3C, lorsqu'on utilise *h2* pour changer le style de tous les titres de niveau 2, on utilise un sélecteur de type. Un sélecteur de type est donc le nom d'une balise dans la page web. Exemple :

```
h2 {
  font-size: 150%;
}
```

fixe la taille de la police pour les titres de niveau 2 à 150 % de la police normale. Il s'agit bien évidemment d'un sélecteur extrêmement courant d'utilisation.

Sélecteurs d'attributs

Un sélecteur CSS peut faire référence à un attribut d'un élément HTML. Les deux attributs les plus couramment utilisés sont **class** et **id** mais il est possible de se référer à n'importe quel attribut. La classe *class* permet d'attribuer des styles génériques (par exemple une classe *petit* pour mettre du texte plus petit) alors que l'identifiant *id* sert à repérer un élément différent des autres dans la page (par exemple la zone de menu).

Sélecteur de classe

Le sélecteur de classe s'applique typiquement à des éléments redondants dans la page. Il est spécifié en CSS par un point (.) et peut concerner tous les éléments HTML utilisant cette classe ou seulement l'un d'entre eux. La syntaxe CSS est la suivante :

```
.nom_de_classe {
    /* déclaration(s) */
}
élément.nom_de_classe {
    /* déclaration(s) */
}
```

Dans le document HTML, on se réfère à cette classe de la sorte (exemple pour l'élément P) :

```
<p class="nom_de_classe">...</p>
```

N'importe quel élément HTML de la page peut utiliser cette classe :

```
<p class="nom_de_classe">...</p>
<ul class="nom_de_classe">
  <li>...</li>
  <li>...</li>
</ul>
```

Ce qui n'empêche pas par la suite de définir des règles communes à tous ces éléments et d'autres spécifiques à certains d'entre eux :

```
.nom_de_classe {
    color: gray;
}
p.nom_de_classe {
    font-style: italic;
}
```

Ici les éléments de classe *nom_de_classe* sont affichés en gris et les paragraphes de cette classe ont en plus leur texte en italique.

Sélecteur d'identifiant

Le sélecteur d'identifiant (ID) ne peut être appliqué qu'à un seul élément dans le code HTML, par exemple un seul paragraphe. Il concerne donc les éléments uniques de structuration du document, comme les blocs principaux (logo, en-tête, colonne(s), pied de page...).

Fixe

Le sélecteur d'identifiant est spécifié en CSS par un dièse (#), la syntaxe est la suivante :

```
élément#nom_id {
    /* déclarations */
}
```

ou :

```
#nom_id {
    /* déclarations */
}
```

Théoriquement seule cette dernière syntaxe devrait être utilisée puisqu'un seul élément peut se voir attribué l'identifiant *nom_id* dans la page. Il est toutefois courant de voir la première syntaxe utilisée pour des raisons de lisibilité du code.

Dans un document HTML, si l'identifiant *nom_id* se rapporte à un élément de type *div*, on ne doit écrire qu'une seule et unique fois dans la page :

```
<div id="nom_id">...</div>
```

Sélecteur d'attribut

À compléter...

Les autres attributs sont référençables en ajoutant [nom_attribut] à un sélecteur de type.
Exemple avec une balise input avec attribut "name" : <input **name**="saisie">
... La règle CSS peut s'écrire: input[**name**]{border:1px solid silver}.
La syntaxe complète des sélecteurs d'attributs (http://www.yoyodesign.org/doc/w3c/css2/selector.html#attribute-selectors)

Les sélecteurs hiérarchiques

Une caractéristique fondamentale des règles CSS est que l'on peut spécifier des règles pour les éléments contenus à l'intérieur de certains autres éléments sans ajouter de classe ni d'identifiant. Concrètement, on peut par exemple changer le style d'un lien selon qu'il est dans un paragraphe ou dans une liste. Ceci est rendu possible par la notion de hiérarchie des éléments du document web, conformément à l'imbrication des balises dans le code de la page.

Le document HTML peut en effet être vu comme un arbre dont le tronc est l'élément `html`, les deux principales branches les éléments `head` et `body`, desquelles partent en ramification toutes les autres branches. Un sélecteur suit ces ramifications, comme une branche de lierre, en partant du tronc, pour désigner une ou des branches — qui sont des éléments HTML. Ainsi, chaque élément a un élément *parent* — sauf l'élément racine `html` — et un ou plusieurs *élément(s) enfant(s)* — sauf les éléments terminant une ramification. Ceci définit une hiérarchie dans laquelle il faut pouvoir s'orienter facilement et précisément.

La grammaire CSS définit plusieurs types de sélecteurs pour parcourir l'arbre qu'est le document HTML.

Le sélecteur de descendant

Le **sélecteur de descendant**, noté par un ou plusieurs espace(s), permet de désigner les éléments qui sont des descendants d'un autre élément (ou de plusieurs), c'est-à-dire liés les uns aux autres par une relation de parenté quelconque. Le dernier élément est inclus directement ou non dans celui qui le précède et ainsi de suite. Exemple :

```
#wikipedia h1 {... }
```

désigne tous les éléments *h1* contenus, à quelque niveau que ce soit, dans les éléments d'identifiant *wikipedia*. Cet exemple illustre pourquoi ce sélecteur est d'utilisation courante : il permet de modifier le style d'éléments enfants sans avoir à les marquer d'une classe ou d'un identifiant.

On peut vouloir aussi désigner des éléments descendants mais au minimum petits-enfants dans la hiérarchie mais il peut être laborieux d'expliciter tout le ou les chemin(s) jusqu'aux éléments concernés. Il est possible d'utiliser alors `*` comme un sélecteur descendant particulier :

```
#wikipedia * h1 {... }
```

Ce sélecteur désigne tous les éléments *h1* petits-enfants d'éléments d'identifiant *wikipedia*, ou situés plus loin dans la hiérarchie du document (par exemple `#wikipedia > #contenu > .introduction > h1`). Ainsi dans le code HTML suivant :

```
<div id="wikipedia">
    <h1> Nom du site </h1>

    <div id="contenu">
      <h1> Titre 1 </h1>
      <p>Paragraphe...</p>

      <h2> Titre 1.1 </h2>
      <p>Paragraphe...</p>
    </div>
</div>
```

les deux titres *h1* seraient affectés par la première règle mais seul le deuxième titre de texte « Titre 1 » serait affecté par la seconde.

Tous ces sélecteurs sont supportés par les principaux navigateurs modernes (Internet Explorer, Mozilla, Firefox, Opera...)

Les sélecteurs d'enfant et de frère adjacent

Le **sélecteur d'enfant**, noté >, permet de désigner un élément par filiation directe à un autre élément. La relation ne concerne donc que les enfants et non les petits-enfants ou plus. L'élément parent est suivi de **>** puis vient l'élément enfant. Exemple :

```
#wikipedia > h1 {...}
```

désigne tous les éléments *h1* enfants d'éléments d'identifiant *wikipedia*. Les espaces ne sont pas obligatoires bien qu'ils rendent la syntaxe plus claire.

Le **sélecteur de frère adjacent** 'ou sélecteur d'adjacence directe), noté +, permet de définir un style pour un élément enfant suivant immédiatement un autre élément enfant du même élément parent. La relation de filiation est donc à nouveau directe. Exemple :

```
h1 + h2 {...}
```

applique un style à l'élément *h2* qui suivrait immédiatement un élément *h1*. Ainsi dans le code HTML suivant :

```
<div class="wikipedia">
   <h1> Titre 1 </h1>

   <h2> Titre 1.1 </h2>
   <p>Paragraphe...</p>

   <h2> Titre 1.2 </h2>
   <p>Paragraphe...</p>
</div>
```

seul le titre de texte « Titre 1.1 » serait affecté par les règles de style.

Attention : *ces sélecteurs sont supportés par les principaux navigateurs modernes (y compris IE5/Mac)* ***sauf Internet Explorer 5.0 à 6.0*** *pour Windows. Ceci limite beaucoup malheureusement l'intérêt de ces sélecteurs qu'on utilisera donc avec parcimonie.*

Enfin, le sélecteur d'adjacence indirecte est noté ~.

Les pseudo-classes et les pseudo-éléments

Fixe
Les sélecteurs présentés précédemment suffisent pour mettre en œuvre des pages web déjà complexes. Pour aller plus loin, d'autres sélecteurs permettent de modifier le style d'éléments dans un état spécifique comme les liens visités ou non, les états

correspondant à l'interaction de l'utilisateur ou encore des éléments répondants à des critères particuliers.

Malheureusement leur support par les Internet Explorer 5.0, 5.5 et 6.0 de Microsoft est extrêmement réduit, ce qui limite beaucoup la portée de ces sélecteurs étant donné que la part de marché des Internet Explorer début 2007 environne les 80%. Il n'y a cependant aucune contre-indication à réserver certaines règles aux (nombreux) autres navigateurs qui les supportent, dans un souci de perfectionnisme. Il faut simplement que les pages du site ne reposent pas sur ces utilisations avancées des normes.

Ressources :

- Les pseudos-classes (fr) (http://www.yoyodesign.org/doc/w3c/css2/selector.html#q15)
- Les pseudos-éléments (fr) (http://www.yoyodesign.org/doc/w3c/css2/selector.html#q20)

Pseudo-classes des liens

Les pseudo-classes `:link` et `:visited` s'adressent uniquement à des liens, donc des balises HTML `<a href="…">…</a>`. Le premier permet de sélectionner les liens qui n'ont pas encore été visités et le second ceux qui l'ont déjà été. Ainsi le code CSS suivant :

```
a:link {color: blue;}
a:visited {color: gray;}
```

fait en sorte que les liens des pages non visitées soient bleus alors que ceux des pages visitées sont gris. On peut bien évidemment jouer sur toute autre propriété du lien. Un traitement mettant un peu d'originalité consiste à barrer les liens visités :

```
a:link {text-decoration: none;}
a:visited {text-decoration: line-through;}
```

Cet effet n'est cependant pas toujours bien apprécié des lecteurs car les liens visités deviennent moins lisibles.

Pseudo-classes d'interaction avec l'utilisateur

Les pseudo-classes `:hover`, `:active` et `:focus` permettent de sélectionner des éléments suivant l'interaction qu'ils ont avec l'utilisateur. Les définitions du W3C sont les suivantes :

- La pseudo-classe *:hover* est effective pendant que l'utilisateur désigne un élément (avec un outil quelconque), mais ne l'active pas. Par exemple, un navigateur pourra appliquer cette pseudo-classe quand le curseur (de la souris) passe au-dessus de la boîte délimitant l'élément.
- La pseudo-class *:active* est effective lorsqu'un élément est activé par l'utilisateur. Par exemple pendant la durée entre laquelle l'utilisateur appuie sur le bouton de la souris puis le relâche.
- La pseudo-classe *:focus* est effective lorsqu'un élément a le focus (ceci inclut les événements du clavier ou tout autre forme de saisie de texte).

Typiquement un champ de formulaire dans lequel le curseur clignote a le focus. On peut utiliser cet effet pour changer l'apparence d'un champ de formulaire et notifier son état d'interaction. Par exemple ce code CSS :

```
input {color: #444; background-color: #fafafa;}
input:focus {color: #400; background-color: #fff;}
```

change l'apparence d'un champ de texte de gris foncé sur fond gris clair lorsqu'il est inactif, à marron foncé sur fond blanc lorsqu'il récupère le focus.

La pseudo-classe *:hover* offre de nombreuses possibilités qui vont du simple changement de couleur au survol de la souris au menu déroulant en pur CSS ! L'utilisation actuellement classique sur le web consiste à supprimer le soulignage par défaut des liens et de le faire apparaître uniquement au survol de la souris :

```
a {text-decoration: none;}
a:hover {text-decoration: underline;}
```

Enfin la pseudo-classe *:active* est d'une utilisation assez marginale, son effet étant extrêmement bref. On peut par exemple l'utiliser lors du clic sur un bouton dans un formulaire pour simuler l'enfoncement du bouton par un changement de couleur.

Remarques :

- La pseudo-classes *:hover* s'applique théoriquement à n'importe quel élément de la page. Malheureusement les **Internet Explorer 5.0, 5.5 et 6.0 de Microsoft ne la reconnaissent que sur les liens**, donc les balises HTML *A*. En pratique on ne l'utilise donc quasiment que pour les liens.
- La pseudo-classe *:focus* n'est pas disponible dans les Internet Explorer de Microsoft, version Windows.
- La pseudo-classes *:hover* peut ne pas correspondre à une interaction possible de l'utilisateur, par exemple lorsque le périphérique de saisie est un stylo.

Autres pseudo-classes

La pseudo-classe `:first-child` désigne le premier enfant d'un élément, c'est-à-dire le premier élément inclus dedans. On peut l'utiliser pour modifier le premier paragraphe d'une page ou encore la première entrée d'une liste. À nouveau l'utilisation de cette pseudo-classe est fortement réduite par le fait que les **Internet Explorer 5.0, 5.5 et 6.0 de Microsoft ne la reconnaissent pas**.

La pseudo-classe `:lang` permet de sélectionner des éléments correspondant à certaines langues. Il faut pour cela que les éléments concernés ou un de leur parent aie(nt) une langue spécifiée, ou encore que la page comporte une mention de langue dans l'entête. Cette pseudo-classe est d'une utilisation extrêmement marginale. Citons tout de même la possibilité de changer le caractère des guillemets utilisés pour les citations dans la balise HTML `<q>…</q>`. Nous utilisons en effet « et » en français alors que les anglo-saxons utilisent plutôt “ et ”.

Les pseudo-éléments

Les pseudo-éléments `:first-line` et `:first-letter` désigne respectivement la première ligne et le premier caractère d'un texte, typiquement un paragraphe. Par exemple le code CSS suivant :

```
div#page p:first-child:first-letter {font-size: 200%;}
```

affiche une première lettre plus grosse pour le premier paragraphe de la page.

Les pseudo-éléments `:before` et `:after` se réfèrent au contenu de l'élément. Ils servent à ajouter du texte avant ou après le texte contenu dans l'élément grâce à la propriété CSS `content`. On s'en sert par exemple pour symboliser les pages précédente et suivante à l'aide de liens (les encodages \00AB et \00BB désignant respectivement les guillemets doubles ouvrants et fermants français):

```
a.prec:before {content: "\00AB ";}
a.suiv:after {content: " \00BB";}
```

Remarques :

- Ces pseudo-éléments `:before` et `:after` ne sont eux aussi **pas reconnus par les Internet Explorer de Microsoft**, toute version.
- Une utilisation plus poussée de `:before` permet de numéroter n'importe quel élément de la page automatiquement mais, bien sûr, pas dans les navigateurs de Microsoft.

Les bonnes recettes

Inutile de mettre des classes partout

L'erreur classique consiste à attribuer des classes à tous les éléments de la page que l'on souhaite modifier. Le sélecteur de descendant associé soit au sélecteur de classe, soit au sélecteur d'identifiant permet dans la très grande majorité des cas de s'en passer.

Prenons l'exemple d'un menu constitué de plusieurs listes de liens. On peut coder un tel menu en HTML de cette façon :

```
<!-- exemple surchargé en classes -->
 <h1 class="menu"> Navigation </h1>
 <ul class="menu">
   <li class="menu"> <a href="..."> Lien 1 </a> </li>
   ...
 </ul>

 <h1 class="menu"> Boîte à outils </h1>
 <ul class="menu">
   <li class="menu"> <a href="..."> Lien A </a> </li>
   ...
```

Fixe

```
</ul>
```

Noter qu'on aurait aussi pu utiliser un nom de classe différent par élément différent (par ex. *menu_title* pour les titres *h1*). En fait il n'est pas nécessaire d'ajouter une classe à chaque élément fonctionnel du menu (le titre, la liste, l'entrée, le lien) sous prétexte que ce sont des éléments HTML différents. Ainsi il suffit de placer le menu dans un élément parent de type *div* et d'identifiant *menu* :

```
<!-- solution habituellement rencontrée -->
<div id="menu">
  <h1> Navigation </h1>
  <ul>
    <li> <a href="..."> Lien 1 </a> </li>
    ...
  </ul>

  <h1> Boite à outils </h1>
  <ul>
    <li> <a href="..."> Lien A </a> </li>
    ...
  </ul>
</div>
```

D'un point de vue de la structure de la page web, il est en plus tout-à-fait logique de placer le menu dans un élément conteneur pour le séparer du reste de la page. Ceci facilite son placement par la suite. On peut alors accéder à chaque élément fonctionnel en CSS de cette manière :

```
div#menu h1 {...}
div#menu ul {...}
div#menu li {...}
div#menu a {...}
```

Cette méthode à l'avantage d'alléger le code HTML — qui en devient donc plus lisible, plus maintenable et moins consommateur de bande passante — tout en gardant un code CSS des plus clairs également.

Utiliser de multiples classes

Une autre erreur courante consiste à multiplier des classes qui sont en fait des combinaisons de classes plus simples. Pour clarifier notre propos, supposons que l'on souhaite disposer de classes pour changer la couleur et/ou la taille du texte :

- couleur noir ou gris
- taille 75 %, 82 % ou 100 %

On est donc tenté d'introduire 6 classes combinaisons de ces 2 et 3 possibilités et nommées par exemple *noir_75p*, *gris_75p*, *noir_82p*, etc. Ceci est inutile car il est possible en HTML d'utiliser plusieurs classes dans un même élément en les séparant simplement d'un espace. Si on définit en CSS les classes suivantes :

```
.noir {color: black;}
.gris {color: gray;}
.t75 {font-size: 78%;}
.t82 {font-size: 82%;}
.t100 {font-size: 100%;}
```

On pourra les combiner à souhait par la suite de cette manière :

```
<p class="noir t82">...</p>
<p class="gris t100">...</p>
```

Bien sûr dans cet exemple le gain est faible (5 classes au lieu de 6) mais nous vous laissons multiplier les possibilités ou les propriétés à pouvoir modifier pour en apprécier le gain. La méthode exposée a le grand avantage de **modulariser les classes** pour plus de souplesse.

Cependant l'objection que l'on pourrait faire serait qu'il n'est plus possible de modifier légèrement le style d'une combinaison particulière de certaines propriétés. Elle n'est en fait pas recevable car il est aussi possible en CSS de spécifier des combinaisons de classes pour un même élément, il suffit de les accoler dans le sélecteur.

Par exemple, un texte petit de couleur grise est difficile à lire et il pourrait être judicieux de l'afficher automatiquement en caractères gras. La combinaison des classes *gris* et *t75* peut alors être modifiée en ajoutant cette règle dans les styles CSS :

```
.gris.t75 {font-weight: bold;}
```

On peut combiner de la sorte autant de classes qu'on le souhaite.

Remarque : l'ordre des classes dans le code HTML n'a pas d'importance. De ce fait il est aussi plus difficile de faire une erreur de syntaxe sur les classes *noir cadre fond-jaune* non ordonnées que sur la classe *noir_cadre_fond-jaune*.

Attention : comme indiqué sur la page Multiple classes (http://www.quirksmode.org/css/multipleclasses.html) (en anglais), Internet Explorer 6.0 de Microsoft n'interprète pas correctement les classes multiples.

Combiner classe et identifiant

Il est possible en HTML d'attribuer à la fois un identifiant et une ou plusieurs classes à un même élément de la page. Cette technique permet de réaliser une sorte d'héritage de styles. À nouveau on peut en tirer profit pour réduire la diversité des styles et les concevoir d'une manière plus générique et donc plus facilement ré-utilisable.

Prenons comme exemple la table des matières (TDM) regroupant les titres de la page web. L'adaptation classique en HTML est la suivante :

```
<!-- table des matières -->
<div id="tdm">
  <h1> Table des matières </h1>
  <ol>
    <li>
     <a href="#..."> Titre 1 </a>
     <ol>
      <li> <a href="#..."> Titre 1.1 </a> </li>
      ...
     </ol>
    </li>
    ...
  </ol>
</div>
```

Ici les liens pointant vers les titres des sections sont placés dans des listes de liens imbriquées. Maintenant nous souhaiterions que cette table des matières soit placée soit dans un cadre flottant à droite ou à gauche du texte, soit à gauche en occupant toute la largeur de la page (positionnement par défaut), selon l'humeur de l'administrateur (ou un paramètre utilisateur ?). Le plus simple est d'ajouter simplement des classes *float* et *left* ou *right* à l'élément conteneur dans les deux premiers cas, par exemple :

```
<div id="tdm" class="float left">
...
</div>
```

On peut ensuite ajouter les règles suivantes dans les styles CSS :

```
/* classes génériques */
div.float {
  margin: 0.5em;   /* garder de la place pour le texte autour et dedans */
  padding: 0.5em 0.25em;
  border: 1px solid #44a;   /* faire un encadré */
  background-color: #fafafa;
}
div.float.left {
  float: left;
  margin-left: 0;   /* se coller contre le bord gauche de la page */
}    Fixe
```

```
div.float.right {
  float: right;
  margin-right: 0;   /* idem bord droit */
}

/* stylisation de la TDM */
div#tdm h1 {...}
div#tdm ol {...}
div#tdm li {...}
```

Les classes génériques auront alors intérêt à être placées dans un fichier CSS séparé afin d'être facilement importées dans un autre projet web.

Imbrication

L'esperluette est l'opérateur d'imbrication :

```
.classe1 {
  &.classe2 { ... }
}
```

Autre exemple :

```
.classe1 {
  &--myDataAttribute { ... }
}
```

Fixe

Valeurs et unités

Distances et dimensions

Il existe plusieurs unités possibles pour spécifier une taille de texte, une taille de boîte ou encore une marge :

- en utilisant des dimensions absolues
  - en centimètres (cm)
  - en millimètres (mm)
  - en pouces (in)
  - les points (pt), 1 point vaut 1/72 de pouce
  - en picas (pc), 1 picas vaut 12 points
- en utilisant des dimensions relatives
  - à la police de caractères
    - la taille de la police (em)
    - la taille de la lettre x minuscule (ex)
  - à la taille de l'écran et la résolution employée
    - les pixels (px)
  - à la dimension d'un élément parent ou une autre dimension du même élément
    - les pourcents (%)

Le choix de l'unité dépendra du média auquel s'applique la feuille de style. Ainsi:

- les unités absolues sont destinées aux feuilles de styles d'impression
- le pixel, à l'inverse, est destiné aux feuilles de styles d'affichage

Pour les styles d'affichage, les valeurs les plus couramment employées sont les pixels (px), les tailles de police (em) et les pourcents (%).

Les couleurs

Les couleurs peuvent être spécifiées soit par mot-clefs soit par valeur RVB (composantes rouge, vert, bleu). Il y a plusieurs notations autorisées pour les composantes RVB :

- `#RVB` où chaque lettre R, V et B est un chiffre hexadécimal entre 0 et F
- `#RRVVBB` où chaque paire de lettres RR, VV et BB est un nombre hexadécimal entre 00 et FF
- `rgb(R, V, B)` où chaque lettre R, V et B est un nombre décimal entre 0 et 255
- `rgb(R%, V%, B%)` où chaque lettre R, V et B est un nombre décimal entre 0 et 100

On passe de la notation `#RVB` à la `#RRVVBB` en dédoublant chaque valeur. Bien évidemment, 100% équivaut aussi à 255 et à #FF. Par exemple les notations suivantes sont équivalentes :

```
p { color: #f00; }
p { color: #ff0000; }
p { color: rgb(255,0,0); }
p { color: rgb(100%, 0%, 0%); }
```

Remarque : les majuscules dans la notation hexadécimale ne sont pas du tout obligatoires.

Les mot-clefs officiellement reconnus sont au nombre de 17 : *aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, orange, purple, red, silver, teal, white,* et *yellow*. Les couleurs et les codes #RRVVBB correspondants sont présentés dans le tableau ci-dessous.

Les 17 couleurs web officielles

nom	valeur	rendu
maroon	#800000	
red	#ff0000	
orange	#ffA500	
yellow	#ffff00	
olive	#808000	
purple	#800080	
fuchsia	#ff00ff	
white	#ffffff	
lime	#00ff00	
green	#008000	
navy	#000080	
blue	#0000ff	
aqua	#00ffff	
teal	#008080	
black	#000000	
silver	#c0c0c0	
gray	#808080	

D'autres noms de couleurs élargissant cette palette de base ont été définis de manière propriétaire par des navigateurs. Mais en pratique, les concepteurs utilisent majoritairement la notation normalisée #RRVVBB.

Il existe également une palette standard de 216 couleurs de "sécurité" [1] (http://www.w3schools.com/Html/html_colors.asp), pour tout navigateur, pour les affichages en 256 couleurs. La représentation de ces couleurs en hexadécimal sur 3 chiffres (format #RVB) n'utilise que des chiffres multiples de 3 : 0 3 6 9 C et F. Ces 6 chiffres combinés donne les 216 (6^3) couleurs.

Références

- W3C : Color units (http://www.w3.org/TR/CSS21/syndata.html#color-units)
- Pour une liste assez conséquente : https://openclassrooms.com/courses/apprenez-a-creer-votre-site-web-avec-html5-et-css3/memento-des-proprietes-css
- L'ensemble des propriétés CSS (http://wiki.media-box.net/documentation/css)
- Référence CSS W3schools.com (http://www.w3schools.com/css/default.asp)

Fixe

Texte

Dans ce chapitre nous détaillons les différentes propriétés CSS relatives au traitement des caractères : police, taille, variante, espacement des lignes, etc. Le balayage des standards du W3C est accompagné de conseils pour bien exploiter le potentiel de traitement des caractères en CSS en évitant les écueils courants des navigateurs et des plates-formes. Alors que les Internautes sont nombreux à parcourir très rapidement les pages web, l'objectif est d'obtenir un texte lisible et, surtout, agréable à lire afin de retenir le lecteur plus que sur les autres sites...

Si vous êtes pressé consultez en priorité :

- font-size : taille des caractères
- color : couleur des caractères
- Gras et italique
- Alignement du texte
- line-height (espacement des lignes)

Les caractères

`font-size` : taille des caractères

La propriété `font-size` permet de modifier la taille des caractères. On peut utiliser une unité de distance ou un mot clef se référant à une taille prédéfinie dans le navigateur web ou modifiant une taille héritée d'un élément parent. Les unités utilisables sont répertoriées dans le chapitre Valeurs et unités, les mot-clefs sont listés plus bas. Par exemple ces styles :

```
p#t1 { font-size: 12px; }
p#t2 { font-size: 1.5em; }
p#t3 { font-size: 175%; }
```

avec le code HTML suivant :

```
<p id="t1">Paragraphe avec style #t1</p>
<p id="t2">Paragraphe avec style #t2</p>
<p id="t3">Paragraphe avec style #t3</p>
```

produisent ceci :

Paragraphe avec style #t1

Paragraphe avec style #t2

Paragraphe avec style #t3

On peut aussi utiliser des mot-clefs décrivant la taille de la police :

- de manière relative avec `larger` (plus grand) ou `smaller` (plus petit)
- de manière absolue avec `xx-small`, `x-small`, `small`, `medium`, `large`, `x-large`, `xx-large`, dans l'ordre du plus petit au plus grand

Dans le premier cas la taille est relative à celle spécifiée pour un parent et dans le second cas la taille correspond à une valeur prédéfinie dans le navigateur. Exemple ces styles :

```
div { font-size: medium; }
p { font-size: smaller; }
```

produisent ceci lorsqu'on imbrique le *p* dans le *div* :

```
<div>
   Texte hors du paragraphe
   <p>Texte dans le paragraphe<p>
</div>
```

Texte hors du paragraphe

Texte dans le paragraphe

Spécificités à connaître

1. Les Internet Explorer 5.0 à 7.0 de Microsoft ne permettent pas à l'utilisateur de modifier la taille de la police affichée lorsque l'unité utilisée est le pixel. De ce fait il est recommandé pour des raisons d'accessibilité de **ne jamais spécifier de tailles de police en pixels** pour permettre d'agrandir le texte.
2. La taille relative `small` correspond généralement à 16 pixels dans Internet Explorer et à 12 pixels dans les autres navigateurs. Il est donc peu recommandé à nouveau d'utiliser ce genre de spécification, à moins d'utiliser une feuille de style spéciale pour Internet Explorer.

`color` : couleur des caractères

La propriété `color` définit la couleur des caractères. Par défaut celle-ci est héritée du parent. Pour la changer, il faut utiliser une valeur de couleur selon les possibilités indiquées dans le chapitre Valeurs et unités.

Exemple :

```
em {color: #800;}
span {color: #004580;}
```

Rend ces couleurs :

```
<em>Texte em</em> et <span>texte span</span>
```

Texte em et texte span

Gras, italique

`font-style`

La propriété `font-style` permet de passer en texte italique ou de revenir en texte droit, lorsqu'on est déjà dans un passage en italique. Les différentes valeurs possibles sont les suivantes :

- `normal`, texte normal
- `italic`, texte en italique
- `oblique`

La police utilisée doit avoir une variante correspondant à chaque valeur de cette propriété. Aussi, comme bien souvent la variante *oblique* n'existe pas, on se contentera des deux premières valeurs.

Par exemple ceci :

```
strong {font-style: italic;}
em {font-style: normal;}
```

donne :

Fixe

```
<strong>Texte strong</strong> et <em>texte em</em>
```

Texte strong et texte em

NB: le style standard de *strong* est caractères gras et celui de *em* est caractères en italique.

`font-weight`

La propriété `font-weight` définit la *graisse* de la police. Les valeurs possibles sont :

- `normal`, texte normal
- `bold`, texte gras
- `bolder`
- `lighter`
- une valeur entre 100 et 900 par pas de 100

À nouveau, seules les deux premières valeurs sont utiles car les navigateurs ne proposent aujourd'hui pas de graisse relative.

Par exemple ceci :

```
strong {font-weight: normal;}
em {font-weight: bold;}
```

donne :

Texte strong et ***texte em***

NB: le style standard de *strong* est caractères gras et celui de *em* est caractères en italique.

`text-decoration` : souligné, surligné, barré

La propriété `text-decoration` permet d'ajouter un trait au texte. Les valeurs possibles sont :

- `none`, texte normal
- `underline`, texte souligné
- `overline`, texte surligné
- `line-through`, texte barré

Cette propriété est très utile pour styliser les liens qui sont par défaut soulignés dans les navigateurs. Il est en effet assez courant de nos jours ne plus les souligner mais de seulement les colorer.

Exemple :

```
strong {text-decoration: underline;}
em {text-decoration: overline;}
span {text-decoration: line-through;}
```

Donne ceci :

Texte strong, *texte em* et ~~texte span~~

NB: le style standard de *strong* est caractères gras, celui de *em* est caractères en italique, enfin *span* est du texte normal.

Majuscules et minuscules

`font-variant`

La propriété `font-variant` permet de basculer les caractères en petites capitales d'imprimerie. Il faut pour cela lui donner la valeur `small-caps`, la valeur `normal` permettant de revenir à des lettres minuscules. Par exemple ce code CSS :

```
p {font-variant: small-caps;}
```

appliqué au texte « Texte du paragraphe. Nouvelle phrase. » donne ceci :

Texte du paragraphe. Nouvelle phrase.

Cet effet peut être intéressant dans les titres ou dans les menus. Par contre sur un texte complet, il est rare d'utiliser des petites majuscules.

`text-transform`

La propriété `text-transform` contrôle la casse du texte. Elle accepte les valeurs suivantes :

- *capitalize* met une majuscule à chaque début de mot
- *uppercase* passe tout en (grandes) majuscules
- *lowercase* passe tout en minuscules
- *none* annule un des effets précédents

Par exemple ce code CSS :

```
strong {text-transform: capitalize;}
em {text-transform: uppercase;}
tt {text-transform: lowercase;}
```

appliqué au texte « minuscules, MAJUSCULES dans ... » donne ceci :

Minuscules, MAJUSCULES Dans Strong

MINUSCULES, MAJUSCULES DANS EM

`minuscules, majuscules dans tt`

À nouveau ce genre d'effet ne trouve habituellement sa place que dans les titres ou les menus.

Le raccourci `font`

La propriété `font` est un raccourci pour spécifier plusieurs propriétés relatives à la police en une seule ligne dont certaines sont détaillées plus loin dans ce chapitre. Cette propriété, bien que pratique, est néanmoins peu tolérante quand à l'ordre des propriétés spécifiées, lequel est le suivant :

1. `font-style`
2. ~~Fixe~~ `font-variant`

3. `font-weight`
4. `font-size`
5. `line-height`
6. `font-family`

Dans cette liste, `font-size` et `font-family` **doivent obligatoirement être mentionnées**. Par ailleurs `font-size` et `line-height` doivent être accolés avec la barre oblique '/'. Il faut donc être rigoureux lorsqu'on utilise ce raccourci. Le W3C donne les exemples suivants :

```
p { font: 12px/14px sans-serif }
p { font: 80% sans-serif }
p { font: x-large/110% "New Century Schoolbook", serif }
p { font: bold italic large Palatino, serif }
p { font: normal small-caps 120%/120% fantasy }
```

Attention : toute propriété parmi les 6 listées plus haut qui n'est pas mentionnée lors de l'utilisation de `font` est **remise à sa valeur initiale**.

Police de caractères

Les grandes familles de polices

Les polices de caractères sont classées selon 5 catégories calquées sur l'imprimerie :

- à empattement (*serif*), les extrémités des caractères comportent de petits crochets décoratifs
- sans empattement (*sans-serif*), les extrémités des caractères sont droites
- à chasse fixe (*monospace*), la place occupée par un caractère est constante
- cursive (*cursive*), imitant plus ou moins une écriture manuscrite
- fantaisie (*fantasy*), il s'agit plutôt d'icônes que de caractères

Exemple (si le type de police n'existe pas dans votre navigateur, il sera remplacé par un autre type, classiquement un *sans-serif*) :

Texte avec une police à empattement

Texte avec une police sans empattement

Texte avec une police à chasse fixe

Texte avec une police cursive

Texte avec une police fantaisie

On considère habituellement qu'un long texte doit être rédigé en utilisant un des deux premiers types. Le débat entre avec ou sans empattement est un sujet à part entière, certains considérant les premières comme plus lisibles. En pratique on trouve très couramment des polices à empattement dans les livres, avec donc une connotation de tradition. Les polices sans empattement sont plus récentes et sont donc empreintes d'une idée de modernisme. Aussi utilise-t-on souvent ces polices sur le web. Enfin les polices à chasse fixe servent souvent à afficher du code source, elles ont ainsi une connotation ordinateur, imprimante, etc.

`font-family` : choix de la police

La propriété `font-family` définit la police de caractères. Elle accepte comme valeur un ou plusieurs nom(s) de polices ou de type de police (police *générique*). Dans le cas de plusieurs noms, il faut les séparer par des virgules et les éventuels espaces additionnels ne comptent pas. Le navigateur affichera alors le texte avec la première police trouvée dans l'ordre de la liste. Pour cette raison, il est de peu d'utilité de mentionner à la fin de la liste des polices ayant moins de chance d'être installées sur la plate-forme de l'utilisateur. Enfin il n'est pas obligatoire d'encadrer les noms de guillemets (") ou d'apostrophes (') et le nom des polices est insensible à la casse.

Comme nous le verrons plus loin, il est impératif de *toujours mentionner* en dernier le nom d'une police *générique* pour que le navigateur ait une solution de rechange s'il ne trouve pas les polices demandées. La liste possible correspond aux types de police précédemment évoqués :

- serif
- sans-serif
- monospace
- cursive
- fantasy

Par exemple ceci :

```
strong {font-family: Bodoni, Bitstream Vera Serif, Times New Roman, serif;}
em {font-family: courier, monospace;}
```

donne (le résultat dépend de la plate-forme utilisée) :

Texte strong (serif) et *texte em (monospace)*

Remarque : si par malheur le nom de la police à utiliser comportait un des mot-clef *inherit*, *initial*, *default* ou encore le nom d'un type de police (*serif*, *sans-serif*, etc.), il faudrait alors impérativement l'encadrer de guillemets (") ou d'apostrophes (') afin d'éviter toute confusion.

Police et taille de police

Même à taille égale, par exemple 16 pixels, deux polices peuvent ne pas sembler de même dimension. Ceci est lié à ce que les typographes appellent *la taille de l'x* (la lettre x). Pour faire simple, disons que ce n'est pas parce que les lettres les plus hautes occupent la même hauteur que les lettres les plus petites le font également.

Ainsi la police *verdana*, très utilisée sur le web, est une police qui paraît généralement plus grosse que les autres à taille égale. C'est très certainement la raison pour laquelle elle est si souvent choisie : à taille égale, elle paraît plus lisible puisque plus grosse. L'inconvénient d'utiliser cette police est que, si elle n'est pas disponible sur la plate-forme utilisée, le navigateur se rabattra sur une autre vraisemblablement moins grosse en apparence. L'utilisateur aura donc une impression différente de la page, probablement moins bonne que souhaitée.

Pour éviter ce type de désagrément, la recommandation que l'on peut faire est d'utiliser comme valeur de la propriété `font-family` une liste de polices dont la taille de l'x est similaire. Pour appréhender cette dimension, il suffit d'aligner verticalement un même mot écrit avec différentes polices. La difficulté, on le verra au paragraphe suivant, est de trouver une liste de polices qui sera probablement disponible sur les 3 plates-formes majeures : Linux, Macintosh et Windows.

Les polices et les plates-formes

Une des grosses difficultés du web est la diversité des plates-formes. Elle se manifeste bien sûr par la diversité des navigateurs web mais aussi par la diversité des polices de caractères installées. Aussi, si le nombre de polices disponibles sur un ordinateur est souvent très réduit et correspond aux polices par défaut du système et celles éventuellement installées par les applications comme les suites bureautiques, les polices installées d'une plate-forme à l'autre ne coïncident généralement pas non plus.

De ce fait, il n'est pas possible de se reposer sur la spécification d'une seule police et, comme il est difficile de trouver une police commune à toutes les plates-formes, il faut toujours accompagner la spécification d'une police d'un nom de police générique. Ceci ne vous empêche nullement de spécifier quelques polices un peu extravagantes en début de liste au cas où l'utilisateur les auraient, bien au contraire.

Si toutefois vous souhaitiez vraiment utiliser une police peu courante, une solution validée pour contourner le problème de l'installation de la police consiste à opérer le remplacement d'un texte par une image que vous aurez créée dans votre logiciel de dessin favori avec la police désirée. Bien sûr on ne peut pas imaginer faire ceci pour un long texte mais seulement pour des titres ou, à la limite, des menus.

Signalons pour être exhaustif qu'il y existe théoriquement des méthodes pour que le navigateur télécharge automatiquement une police manquante à partir d'un fichier de police. Toutefois ces méthodes sont très liées à la plate-forme utilisée et ne peuvent pas être exploitées avec une grande chance de succès. Le plus serein est donc d'accepter que les utilisateurs n'aient pas tous le même rendu des pages que vous créez, sinon utilisez des documents PDF !

Liste de polices probablement installées

À compléter...

Fixe

Ressources

Code Style font sampler (http://web.archive.org/20021029112039/www.codestyle.org/css/font-family/) en langue anglaise présente des sondages (*survey*) sur la disponibilité des polices sur les 3 plates-formes majeures : Linux, Macintosh et Windows.

Alignement et espacement

Alignement du texte

`text-align`

La propriété `text-align` définit l'alignement du contenu d'un élément, typiquement le texte dans un paragraphe. Les valeurs possibles sont :

- *left*, alignement à gauche
- *right*, alignement à droite
- *center*, alignement au centre
- *justify*, alignement à droite et à gauche

Une figure valant mieux que de longues explications, l'effet de chaque valeur est le suivant :

Texte aligné à gauche avec *left*	Texte aligné à droite avec *right*	Texte aligné au centre avec *center*	Texte justifié, aligné à droite et à gauche avec *justify*

On constate aisément dans cet exemple que le texte justifié peut ne pas toujours être une bonne solution lorsque la largeur du contenant n'est pas suffisante. Des espacements exagérés entre mots peuvent en effet apparaître, c'est un problème connu avec ce type d'alignement du texte. La solution serait d'introduire la césure des mots mais cela ne fait pas encore partie du standard CSS !

Remarque : Les spécifications du W3C indiquent que la propriété `text-align` ne concerne que les contenus de type *inline* (en ligne), c'est-à-dire destinés à être affichés les uns à la file des autres comme du texte, des liens, etc.

`text-indent`

La propriété `text-indent` définit le retrait de la première ligne de texte d'un élément. La valeur spécifiée est une valeur de distance. Par exemple le code suivant :

```
p {text-indent: 2em;}
```

donne ceci :

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

Espacement des caractères, des mots et des lignes

`letter-spacing`

La propriété `letter-spacing` modifie l'espacement entre les lettres des mots. Cet espacement est nul par défaut, ce qui ne veut pas dire que les caractères se touchent, et peut prendre une valeur de distance quelconque. On l'utilisera notamment pour rendre plus lisibles des titres surchargés par la graisse ou les lettres capitales. Par exemple, le code CSS suivant :

```
strong {font-size: 150%}
span {font-size: 150%; font-weight: bold; letter-spacing: 2px;}
```

donne ceci :

Texte strong

Texte span

L'effet produit sur un gros texte est loin d'être négligeable comme on peut le voir, il est donc regrettable que cette propriété soit si peu utilisée sur le web.

Remarque : le style par défaut de *strong* est caractères gras alors que *span* est du texte normal.

`word-spacing`

La propriété `word-spacing` contrôle l'espacement des mots. Cet espacement est nul par défaut, ce qui à nouveau ne signifie pas que les mots se touchent, et peut prendre une valeur de distance quelconque. Comme l'espacement de mots standard des navigateurs est très bien, il n'y a que peu de raisons d'utiliser cette propriété, si ce n'est pour créer des effets de typographie. Par exemple, le code CSS suivant :

```
strong {word-spacing: 1em;}
span {word-spacing: 2em;}
```

donne ceci :

Texte strong

Texte span

À utiliser à bon escient...

`line-height`

La propriété `line-height` modifie l'espacement entre les lignes d'un texte. Cet espacement peut prendre une valeur de distance quelconque. L'espacement entre les lignes est un facteur important de lisibilité et d'agrément de lecture et, malheureusement, l'espacement par défaut des navigateurs est souvent trop étroit. L'œil reconnaît tout de suite un espacement qui convient et qui lui permet de suivre facilement la ligne en cours de lecture. Ainsi, bien que dans les exemples suivant il s'agisse de latin, il est clair qu'une valeur entre 1.5em et 2.0em rend le texte plus agréable à lire ! Signalons également qu'imposer une augmentation de l'interlignage peut parfois être le seul moyen d'éviter l'enchâssement des caractères typographiques dans le cas de titres particulièrement longs qui causent un retour à la ligne comme le montre le dernier exemple.

Espacement de 1.1em, de toute évidence pénible à lire :

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

Espacement de 1.6em, très convenable :

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

Espacement de 2.5em, exagéré :

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor

in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Bien évidement rien n'empêche d'utiliser des valeurs volontairement inhabituelles sur des textes courts afin de leur donner une importance. Évidemment il faut dans ce cas user de l'effet avec parcimonie sans quoi le lecteur risque de trouver les pages peu agréables.

Enfin voici ce qui se passe sur un long titre avec un espacement de ligne faible :

Titre avec un texte très long qui va sans doute déborder sur la ligne suivante et causer un affichage disgracieux sur la plupart des écrans

Colonnes

La propriété `column-count` permet de définir le nombre de colonnes d'un bloc de texte. Par exemple (CSS intégré dans du HTML) :

```
<ul style="column-count: 4">
  <li> Premier élément ;</li>
  <li> deuxième élément ;</li>
  <li> troisième élément ;</li>
…
</ul>
```

Les propriétés suivantes définissent la mise en forme des colonnes :

- `column-gap` : espacement entre les colonnes (distance) ;
- `column-rule` : paramètres du filet, par exemple `column-rule: 1px solid lightblue;` ; ces paramètres peuvent être indiqués séparément :
  - `column-rule-style` : type de filet pouvant prendre les valeurs `solid`, `dotted`,`dashed` ou `none`,
  - `column-rule-width` : épaisseur du filet,
  - `column-rule-color` : couleur ;
- `column-span` : si un élément s'étend sur plusieurs colonnes en largeur, indique le nombre de colonnes (ou bien `all`) ;
- `column-width` : largeur des colonnes (distance).

Fixe

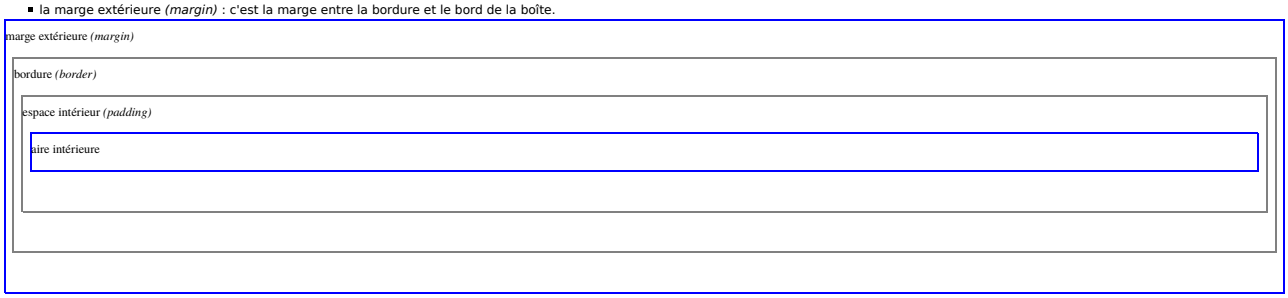
Fonds, bordures, marges et espacements

Surface

L'élément HTML — texte, image, tableau, … compris entre deux balises — est inclus dans une boîte dont on peut définir le fond et la bordure.

On distingue, de l'intérieur vers l'extérieur :

- l'aire intérieure : c'est la zone de contenu, pour laquelle on peut définir le fond *(background)* ;
- l'espace intérieur (*padding*, littéralement « rembourrage ») : c'est la marge entre l'aire intérieure et la bordure ;
- la bordure proprement dite *(border)* ;
- la marge extérieure *(margin)* : c'est la marge entre la bordure et le bord de la boîte.



Aire intérieure

Couleur du fond

La couleur de fond d'une boîte CSS est modifiée avec la propriété `background-color`. Les valeurs possibles sont bien sûr les valeurs de couleur. Par exemple ceci :

```
p { background-color: #f0f8fc; }
```

produit cela :

Paragraphe avec fond coloré

Par défaut, une boîte CSS n'a aucune couleur de fond: la valeur de la propriété `background-color` est en effet `transparent`.

Image du fond

La propriété `background-image` définit une image de remplissage du fond d'une boîte CSS. Elle accepte comme valeur l'URL de l'image et n'est pas du tout incompatible avec la propriété précédente `background-color`. En effet, la couleur de fond est peinte *sous* l'image. Ainsi, la couleur peut apparaître dans une image comportant des zones transparentes, et cette couleur remplira le reste de la surface de la boîte si l'image n'a pas des dimensions suffisantes.

On peut définir :

- la position de l'image, avec `background-position` ;
- la répétition de l'image, avec `background-repeat` ;
- le défilement de l'image, avec `background-attachment`.

Le raccourci `background`

La propriété `background` permet de définir une couleur ou une image de fond. Exemple : `<div style="background: lightblue;">Test de couleur</div>` donne :

Test de couleur

Opacité

La propriété `opacity` à 0 engendre une transparence totale, à 1 une opacité totale.

Exemple d'un test d'opacité / transparence à 50 % :

```
<div style="background-color: yellow; width: 100%; padding: 5px;">
  <div style="background-color: lightblue; opacity: 0.5; position:50px;">Bleu sur jaune.</div>
  <div style="background-color: red; opacity: 0.5; position:50px;">Rouge sur jaune.</div>
</div>
```

donne :

Bleu sur jaune.
Rouge sur jaune.

Bordures et marges

La bordure est séparée du contenu par un espace intérieur *(padding)*, et du bord de la boîte par une marge *(margin)*.

Espace intérieur

On peut distinguer les quatre espaces intérieurs :

- `padding-top` : espace intérieur haut ;
- `padding-bottom` : espace intérieur bas ;
- `padding-left` : espace intérieur gauche ;
- `padding-right` : espace intérieur droit.

La syntaxe est du type *`propriété: valeur`*, ou *valeur* est exprimée comme d'habitude en centimètres, millimètres, pouces, pixels ou cadratins (voir le chapitre *Valeurs et unités > Distances et dimensions*).

Exemple

```
.foo {
  padding-top: 0.5em;
  padding-bottom: 0.5em;
  padding-left: 1em;
  padding-right: 1em
}
```

Fixe

Si les quatre valeurs sont égales, on peut utiliser la propriété `padding`

Exemple

```
.foo {padding: 0.5em}
```

En fait, la propriété `padding` permet de définir les quatre espaces intérieurs en une seule fois :

```
.foo {padding: top right bottom left}
```

L'ordre est celui du sens horaire : haut (*top* à 12h), droite (*right* à 3h), bas (*bottom* à 6h), gauche (*left* à 9h).

En reprenant l'exemple avec quatre propriétés, l'équivalent est :

```
.foo {padding: 0.5em 1em 0.5em 1em}
```

Lorsqu'une valeur est absente, la propriété vaut la même valeur que celle située deux positions avant :

- Si *left* est absent, sa valeur est la même que *right*,
- Si *bottom* est absent, sa valeur est la même que *top*.

ou une position avant (si une seule valeur) :

- Si *right* est absent (donc une seule valeur spécifiée), sa valeur est la même que *top*, idem pour *left* et *bottom* également absents.

Bordure

La bordure est définie par

- sa couleur : `border-color`, dont la valeur est une distance (voir *Valeurs et unités > Les couleurs*) ;
- son style : `border-style`, qui peut prendre les valeurs
  - `solid` : ligne continue ;
  - `dotted` : pointillé ;
  - `dashed` : ligne discontinue (tirets) ;
  - `none` : Aucune bordure ;
- sa largeur : `border-width`, dont la valeur est une distance ou un mot-clef de type `medium`.

Exemple

```
<div style="border-color: gray;
  border-style: dotted;
  border-width: medium">
    Bla bla bla
</div>
```

donne

Bla bla bla

On peut synthétiser tout cela avec `border`, l'exemple ci-dessus devient alors

```
<div style="border: gray dotted medium">
  Bla bla bla
</div>
```

On peut définir une bordure différente

- en haut, avec `border-top` ;
- en bas, avec `border-bottom` ;
- à gauche, avec `border-left` ;
- et à droite, avec `border-right`.

Exemple

```
.foo {
  border-top:solid 1px black;
  border-bottom:solid 1px black;
  border-left:none;
  border-right:none
}
```

et on peut définir indépendemmant chaque caractéristique de chaque trait : `border-top-width`, `border-top-style`, …

Marge extérieure

La marge extérieure est définie par la propriété `margin`, selon le même modèle que l'espace intérieur. On peut définir les quatre marges indépendamment (`margin-top`, `margin-bottom`, `margin-left`, `margin-right`) ou bien les définir avec une seule propriété (`margin`).

Exemples d'applications

Double soulignement

On peut combiner la propriété `text-decoration` avec `border-bottom` pour produire un double soulignement

```
.double-soulignement {
  text-decoration: underline;
  padding-bottom: 0.90em;
  border-bottom: solid 1px
}
```

Soulignement pointillé

La propriété `border-bottom` permet de produire un « faux » soulignement, mais avec plus de possibilités de mise en forme. Par exemple, pour une mise en forme du type de celle par défaut avec la balise HTML `<abbr>…</abbr>`, on peut utiliser `border-bottom: dotted 1px`.

Contour

Le contour s'apparente à la bordure dont il reprend en grande partie les propriétés de couleur, de style et de largeur. Il en diffère cependant sur plusieurs points :

- un contour ne peut pas avoir le style `hidden`
- il admet une nouvelle valeur de couleur, `invert`, qui le dessine en inversant les valeurs de couleurs d'arrière-plan et de premier plan du contexte
- il n'est pas comptabilisé dans les dimensions de l'objet et ne modifie pas le placement des éléments lorsqu'on modifie sa largeur
- il suit les limites de l'élément et peut donc ne pas être rectangulaire lorsque l'élément s'étale sur plusieurs lignes par exemple

- il ne peut pas être ouvert, et se referme toujours autour de l'élément

Les syntaxes des propriétés de contours sont similaires à celles des bordures:

- `outline-color` prend les mêmes valeurs que `border-color`, plus `invert`
- `outline-style` prend les mêmes valeurs que `border-style`, sauf `hidden`
- `outline-width` prend les mêmes valeurs que `border-width`
- La propriété raccourcie `outline` s'utilise comme la propriété `border`

L'utilisation principale des contours est de signaler visuellement quel élément a reçu le focus à la suite d'une action de l'utilisateur: entrée dans un champ de formulaire, sélection d'un bouton ou d'un lien. Chaque navigateur graphique restitue alors un contour selon ses propres styles par défaut. Ces contours jouent donc un rôle clé pour les utilisateurs qui interagissent avec la page Web via le clavier ou un dispositif similaire, et non avec la souris. Les auteurs de pages Web ne doivent en aucun cas tenter de forcer leur disparition, et ne doivent agir qu'avec prudence s'ils souhaitent en modifier l'apparence.

Enfin, les propriétés de contours sont inégalement supportées selon les navigateurs graphiques, et leurs valeurs par défaut diffèrent (par exemple, une bordure grise pointillée dans Internet Explorer, une inversion des couleurs de l'élément dans Opera)...

Arrondis

On définit l'arrondit d'un cadre avec le rayon de ses quatre coins :

```
border-radius: 10px;
```

Profondeur

La 3D peut-être mesurée par trois axes : x (longueur), y (hauteur), z (profondeur). Pour définir dans quel ordre les objets doivent se superposer, on utilise la propriété `z-index`, qui représente leurs positions sur l'axe z.

Exemple :

```
<div style="background-color: lightblue; position: 50px; z-index: 9; margin-bottom:-10px;">Bleu sous rouge</div>
<div style="background-color: red; position: 50px; z-index: 11;">Rouge sur bleu.</div>
```

donne :

Bleu sous rouge
Rouge sur bleu.

Vue de Wikilivres vu en 3D, avec Firefox Tilt.

Fixe

Listes

Les listes en HTML

Les listes en HTML sont de différents types :

- Liste non-ordonnée (***U**nordered **L**ist*) : La balise `<ul>` est utilisée et contient une série d'item encapsulés dans des balises `<li>`. Chaque item est précédé d'une puce.
- Liste ordonnée (***O**rdered **L**ist*) : La balise `<ol>` est utilisée et contient une série d'item encapsulés dans des balises `<li>`. Chaque item est numéroté.
- Liste de définitions (***D**efinition **L**ist*) : La balise `<dl>` est utilisée et contient une série d'item sous la forme de paires termes-définition encapsulées respectivement dans des balises `<dt>` (Terme) et `<dd>` (Description).

Numérotation et puces des items de la liste

La propriété `list-style-type` s'applique aux balises `<li>` et permet de modifier le style de numérotation dans une liste ordonnée[1].

Listes ordonnées

Par défaut la numérotation HTML s'incrémente automatiquement en chiffres arabes :

```
<ol>
<li>Premier</li>
<li>Deuxième</li>
...
<li value=99>Énième</li>
</ol>
```

donne :

1. Premier
2. Deuxième

   ...

99. Énième

De plus, elle peut avoir l'une des valeurs suivantes (liste non exhaustive) pour les listes ordonnées :

upper-roman
: Numéro en chiffres romains, en utilisant des majuscules,

lower-roman
: Numéro en chiffres romains, en utilisant des minuscules,

upper-latin
: Lettre de l'alphabet latin, en utilisant des majuscules,

lower-latin
: Lettre de l'alphabet latin, en utilisant des minuscules.

Exemple, en définissant un style :

```
.list-upper-roman li {  list-style-type: upper-roman;  }
```

```
En utilisant le style défini dans la feuille de style :
<div class="list-upper-roman">
<ul>
<li>Premier</li>
<li>Deuxième</li>
<li>Troisième</li>
<li>Quatrième</li>
<li>Cinquième</li>
</ul>
</div>
ou en appliquant directement un style à chaque item :
<ol>
<li style="list-style-type: katakana-iroha">Premier
<li style="list-style-type: katakana-iroha">Deuxième
<li style="list-style-type: katakana-iroha">Troisième
<li style="list-style-type: katakana-iroha">Quatrième
<li style="list-style-type: katakana-iroha">Cinquième
</ol>
```

Cela donne :

En utilisant le style défini dans la feuille de style :

I. Premier
II. Deuxième
III. Troisième
IV. Quatrième
V. Cinquième

ou en appliquant directement un style à chaque item :

イ、Premier
ロ、Deuxième
ハ、Troisième
ニ、Quatrième
ホ、Cinquième

Listes non ordonnées

Elle peut avoir l'une des valeurs suivantes (liste non exhaustive) pour les listes non ordonnées :

disc
: Disque plein (par défaut),

circle
: Cercle (vide),

square
: Carré,

none
: Rien.

Exemple :

```
<ul>
<li style="list-style-type: disc;">disc
<li style="list-style-type: circle;">circle
<li style="list-style-type: square;">square
```

```
<li style="list-style-type: none;">none
<li style="list-style-type: inherit;">inherit
</ul>
```

Cela donne :

- disc
- circle
- square

none

- inherit

Si l'on veut utiliser un autre caractère, typiquement un tiret cadratin ou demi-cadratin, il faut utiliser la propriété `:before`, par exemple

```
ul {
  list-style-type: none;
}

ul li:before {
  content: "\2014";
  position: absolute;
  margin-left: -1.5em;
}
```

Utiliser une image comme puce

La propriété `list-style-image` peut être utilisée pour spécifier l'URL d'une image à utiliser comme puce pour les items de la liste.

```
<li style="list-style-image: url('https://upload.wikimedia.org/wikipedia/commons/thumb/6/63/Wikipedia-logo.png/35px-Wikipedia-logo.png');">image
```

Références

1. http://www.w3schools.com/cssref/pr_list-style-type.asp

Fixe

Tableaux

Afficher des tables

Les valeurs de la propriété `display` ont été étendues depuis CSS2.1 à[1] :

- `table`
- `inline-table`
- `table-row-group`
- `table-header-group`
- `table-footer-group`
- `table-row`
- `table-column-group`
- `table-column`
- `table-cell`
- `table-caption`

Les bordures adjacentes des cellules peuvent être fusionnées en une seule bordure fine avec la propriété `border-collapse`. Exemple : `"border-collapse:collapse;"`.

L'espacement entre les bordures des cellules peut être défini avec la propriété `border-spacing` depuis CSS2. Mais la configuration de cette propriété n'a aucun effet si les bordures ont été faites pour ressembler à une bordure fine avec `"border-collapse:collapse;"`.

Références

1. http://www.w3.org/TR/CSS21/visuren.html#propdef-display

Fixe

Le positionnement des éléments

Initiation au positionnement en CSS

Pour éviter l'usage inconsidéré des tableaux de mise en page, l'utilisation correcte de chaque élément (balises `div`, `p`, `h1`, `ul`, `li` etc.) ainsi que leur positionnement en CSS est dorénavant indispensable.

Les éditeurs HTML visuels n'utilisent qu'une infime partie des possibilités des balises (faussement nommées « calques » en général), ce qui les rend souvent peu compatibles et peu pratiques.

Cet article en deux parties explique comment positionner les éléments en CSS de façon optimale.

Voici les principaux points à retenir de cet article :

- Tous les éléments (balises) HTML peuvent être positionnés, décorés, dimensionnés... grâce aux styles CSS. Cela signifie que les CSS ne s'appliquent pas qu'aux éléments contrairement à certaines idées reçues : une image (<img>), une liste (<ul>), un paragraphe (<p>), etc. peuvent être stylés en CSS sans avoir besoin d'être contenus dans un élément . Évitez d'ailleurs de surcharger vos documents d'éléments inutiles.
- Employez toujours les éléments selon leur fonction et non selon leur aspect (puisque cet aspect pourra facilement être modifié via CSS) : un paragraphe sera défini par la balise <p>, un titre par la balise <h1>, <h2>... Notez que l'élément est neutre est sert de « bouche-trou » pour englober d'autres éléments ou servir lorsqu'aucun autre élément n'est approprié. N'utilisez pas pour englober des éléments seuls !
- Il existe deux types généraux d'éléments HTML : les balises de type « en-ligne » et les balises de type « bloc ». Cette différence est fondamentale et a de nombreuses implications sur les styles qui peuvent être appliqués.
- L'imbrication des éléments les uns dans les autres, et les notions de parenté (parent, ancêtre, frère...) permettent de styler facilement les éléments en utilisant cette hiérarchie dans les sélecteurs CSS. Cela évite d'employer des noms de classes ou d'id multiples et inutiles, qui complexifient la lecture et la compréhension des styles.
- Il existe plusieurs schémas de positionnement en CSS : un schéma de positionnement dans le Flux (positionnement par défaut), et trois schémas de positionnement qui sortent l'élément du Flux (positionnement absolu, positionnement fixé et positionnement flottant). Un dernier « positionnement » (relatif) est en fait une variante du positionnement courant qui provoque un décalage. La notion de Flux est fondamentale.
- Chaque schéma de positionnement a ses règles et spécificités. Mieux vaut bien comprendre ces règles pour éviter des interactions ou des comportements non désirés... et déclarer que les CSS ne fonctionnent pas ! :)

Les balises de bloc et les balises en-ligne

Ce chapitre sur la structure des balises est primordial, vous devez auparavant avoir consulté le didacticiel sur les bloc/en-ligne.

Voici un résumé pour les lecteurs pressés

- les éléments de rendu « bloc » se placent toujours l'un en dessous de l'autre par défaut (comme un retour chariot). Par exemple : une suite de paragraphes (balise <p>) ou les éléments d'une liste (balise <li>). Par ailleurs, un élément « bloc » occupe automatiquement, par défaut, toute la largeur disponible dans son conteneur.
- les éléments de rendu « en-ligne » se placent toujours l'un à côté de l'autre afin de rester dans le texte. Par exemple : le renforcement d'une partie de texte à l'aide de la balise.
- les autres éléments de type en-ligne (« non-remplacés ») n'ont pas de dimension à proprement parler par défaut (il n'occupent que la place minimum nécessaire à leur contenu). C'est le cas des éléments , , etc.

Ancêtre, Parents, Enfants, Frères

Comprendre l'imbrication des éléments (boîtes) les uns dans les autres est essentiel.

Chaque document HTML est toujours composé de conteneurs. Ceux-ci peuvent être ancêtres, parents, enfants ou frères. Ces différents éléments composent une hiérarchie d'imbrications.

- Un élément « ancêtre » est un élément qui contient un élément ou une hiérarchie d'éléments
- Un bloc « parent » est un élément contenant directement un autre bloc. Par exemple, un `DIV` contenant un paragraphe P. Attention : si ce paragraphe contient lui-même des éléments (ex : `strong`), `DIV` ne sera pas parent de l'élément `strong` mais uniquement son ancêtre. Le parent est donc l'ancêtre immédiat.
- Un bloc contenu directement dans un autre bloc est dit « enfant » de cet élément. Par exemple, ici les éléments `LI` sont enfants de leur conteneur `UL`.
- Les éléments ayant le même élément parent sont appelés « frères ».

Le flux

La mise en place des différents éléments de la page se fait par défaut selon le flux courant.

Les éléments (balises) sont placés les uns après les autres dans le code HTML de la page.

L'ordre dans lequel apparaissent les balises dans le code HTML sera l'ordre dans lequel ils seront affichés et apparaîtront dans le document, c'est le flux. Cela signifie que, par défaut, chaque élément est dépendant des éléments frères qui l'entourent.

Par défaut, les balises « bloc » et les balises « en-ligne » ont un comportement différent dans le flux normal (voir tutoriel sur la structure des balises).

NOTE : les blocs positionnés en « absolu » ou « fixé » sortent du flux naturel et échappent quelque peu à cette règle. Ils ne sont alors plus dépendant des éléments frères. Pour se placer, un tel bloc se réfère non pas à son Parent direct, mais au premier Ancêtre positionné qu'il rencontre.

Sachez utiliser chaque balise à bon escient !

- Il est souvent inutile de faire des imbrications multiples de , ceux-ci ne servant qu'à délimiter des zones de page
- Évitez d'utiliser les balises <table> pour la mise en page de votre site : ils ne sont pas fait pour ça mais pour des données tabulaires et statistiques
- Utilisez les balises <h1>...<h6> pour vos titres et non des balises <p> améliorées
- Utilisez les listes <ul> pour vos menus, car leur utilisation est tout à fait appropriée pour ça (voir les articles de la catégorie « menus en CSS »)

Alignement de blocs tels que les images

Centrer un bloc requiert une technique bien différente de celle pour centrer un texte :

```
    width: 50%;
    display: block;
    margin: auto;
```

Position relative et absolue

On distingue quatre sortes de positionnement, correspondant aux valeurs de la propriété `position` :

1. `static` : statique, les éléments se placent dans l'ordre où ils apparaissent dans le code source. C'est le positionnement par défaut quand il n'est pas précisé.
2. `relative` : relative, les éléments sont décalés par rapport au positionnement statique, puis du nombre de pixels définis avec `left`, `right`, `top`, `bottom`. Attention : si `left` est défini, `right` est ignoré, et si `top` est défini, `bottom` est ignoré.
3. `absolute` : absolue, les éléments sont décalés par rapport au bloc qui les contient.
4. `fixed` : fixe, les éléments sont décalés par rapport à la fenêtre du navigateur. Ils ne bougent pas dans la fenêtre quand on fait défiler son contenu.

Exemple :

```
<div style="border-style: solid;">
    <p style="position: static; bottom: 5px; left: 50px;">Statique</p><br/>
    <p style="position: relative; bottom: 5px; left: 50px;">Relative</p><br/>
    <p style="position: absolute; bottom: 5px; left: 50px;">Absolue</p><br/>
    <p style="position: fixed; bottom: 5px; left: 50px;">Fixe</p><br/>
</div>
```

donne :

Statique

Relative
Fixe

(on voit la position statique non affectée par le décalage, puis la relative décalée en-dessous, le fixe est en bas à gauche de la fenêtre et l'absolue en bas du contenu de celle-ci).

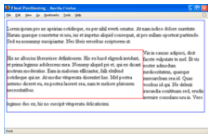
Éléments flottant

La propriété "float" permet aux éléments de flotter dans le flux normal. Les boites bougent alors à gauche ou à droite jusqu'à la limite de leur conteneur. Les éléments situés après un flottant sont alors remontés pour combler le trou laissé autour de ce dernier.

La propriété "float" ne peut donc prendre que les valeurs "left" ou 'right". Pour que cela fonctionne, il faut aussi définir la propriété `width` du flottant, sinon il occupe toute la largeur de son conteneur.

Exemple :

```
    border: 1px solid #F00;
    float: left;
    width: 425px;
```



Exemple d'élément flottant (en rouge). La présente image est également un élément flottant à droite du wikilivre (le texte qui la suit est remonté à sa gauche).

Fixe

Le modèle de boîte

Ombrage des boites

La propriété `box-shadow` existe depuis CSS3. Elle crée un effet d'ombrage qui suit le contour de la boite, avec quatre arguments possibles : abscisse, ordonnée, diffusion, couleur.

CSS3	WebKit	Gecko
box-shadow	-webkit-box-shadow	-moz-box-shadow

Exemples

Cette boite affiche une ombre rectangulaire en bas à droite.

```
div {
    border: 1px solid;
    box-shadow: 10px 10px 0px black;
    -webkit-box-shadow: 10px 10px 0px black;
    -moz-box-shadow: 10px 10px 0px black;
    padding: 0.5em;
}
```

Cette boite affiche une ombre arrondie en bas à droite.

```
div {
    border: 1px solid;
    box-shadow: 10px 10px 0px black;
    -webkit-box-shadow: 10px 10px 0px black;
    -moz-box-shadow: 10px 10px 0px black;
    padding: 0.5em;
    border-radius: 10px;
    -webkit-border-radius: 10px;
    -moz-border-radius: 10px;
}
```

Cette boite affiche une ombre diffuse en rouge en haut à gauche.

```
div {
    border: 1px solid;
    box-shadow: -10px -10px 5px red;
    -webkit-box-shadow: -10px -10px 5px red;
    -moz-box-shadow: -10px -10px 5px red;
    padding: 0.5em;
    border-radius: 10px;
    -webkit-border-radius: 10px;
    -moz-border-radius: 10px;
}
```

Cette boite affiche une ombre diffuse en rouge, vert et bleu, en bas à droite.

```
div {
    border: 1px solid;
    box-shadow: 10px 10px 5px red, 15px 15px 5px green, 20px 20px 5px blue;
    -webkit-box-shadow: 10px 10px 5px red, 15px 15px 5px green, 20px 20px 5px blue;
    -moz-box-shadow: 10px 10px 5px red, 15px 15px 5px green, 20px 20px 5px blue;
    padding: 0.5em;
    border-radius: 10px;
    -webkit-border-radius: 10px;
    -moz-border-radius: 10px;
}
```

Taille des boites

La propriété `box-sizing` est apparue en CSS3[1] pour simplifier les couches.

W3C	WebKit	Gecko
box-sizing	-webkit-box-sizing	-moz-box-sizing

Elle prend un seul argument, parmi les valeurs suivantes :

- `content-box`
  boites intérieures normales.
- `border-box`
  boites intérieures autour de la bordure.
- `inherit`
  comportement hérité du parent.

Exemple

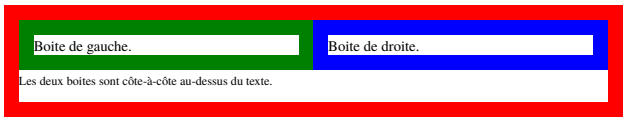
```
div.bigbox {
    width: 40em;
    border: 1em solid red;
    min-height: 5em;
}

div.littlebox {
    width: 50%;
    border: 1em solid;
    box-sizing: border-box; /* cela définit les boites contre la bordure du conteneur */
    -webkit-box-sizing: border-box;
    -moz-box-sizing: border-box;
```

```
    float: left;
}
```

```
<div class="bigbox">
    <div class="littlebox" style="border-color: green;">Boite de gauche.</div>
    <div class="littlebox" style="border-color: blue;">Boite de droite.</div>
</div>
```

Rendu :



Références

1. http://www.w3.org/TR/css3-ui/#box-sizing

Fixe

Techniques avancées

Remplacement d'un texte par une image

Le texte doit être balisé et identifié de manière unique, par exemple par

```
<h1 id="premier_titre">Titre</h1>
```

dans le fichier XHTML. On masque alors le texte par le paramètre

```
display:none
```

dans le CSS et on affiche l'image, (avec tous les attributs) en fond.

Les personnes non voyantes auront un CSS spécial, affichant le titre, sans l'image.

Variations autour des listes

Liste « plate »

Cette section est vide, pas assez détaillée ou incomplète.

Listes en onglets

Menu déroulant

Modification du texte de l'élément

`content`

`quotes`

Validation

Afin de vérifier qu'il n'y a pas d'erreur de syntaxe dans le code CSS, il existe des outils comme https://jigsaw.w3.org/css-validator/.

Minification

Réduire le temps de chargement en supprimant tous les caractères non indispensables à l'exécution du code, comme les espaces.

Des outils comme https://github.com/fmarcia/UglifyCSS le font automatiquement.

Concaténation

Moins un site Internet possède de fichiers différents, et plus il sera rapide à charger car le nombre de connexions HTTP sera moindre. Certains outils de minification procèdent dont à la concaténation des CSS.

CSSOO

Cette section est vide, pas assez détaillée ou incomplète.

Préprocesseurs

SASS

Cette section est vide, pas assez détaillée ou incomplète.

LESS

http://lesscss.org/

Fixe

Navigateurs et débogage

Débogage

Un mauvais CSS peut perturber l'affichage des pages liées.

Les principales erreurs sont :

- Accolades fermantes manquantes.
- Point virgule manquant.
- Identifications ou nommage des balises mal effectué.

Un bon moyen de vérifier si c'est le code HTML ou CSS qui est fautif, est de désactiver le CSS. Par exemple dans Firefox, menu "Affichage", choisir "style de page" à "aucun style". Le CSS ne sera alors plus pris en compte.

De plus, tous les navigateurs n'interprètent pas les classes de la même façon selon le code qui les appelle. En effet, le rendu CSS peut être très différent d'un navigateur à l'autre, c'est pourquoi après chaque modification, il faut tester au moins les quatre principaux engins (Gecko, Webkit, Trident, et Presto). Cela évitera un effort important de recherche de bug par la suite. Pour ce faire il existe des outils comme http://browsershots.org.

De plus, on peut utiliser JavaScript pour charger un code CSS plutôt qu'un autre selon le navigateur qu'il a détecté :

```
<script type="text/javascript">
    alert(navigator.appCodeName);
</script>
```

Checklist

1. Concernant le choix de passer par une classe ou un id, veuillez n'utiliser les classes que pour une collection d'éléments apparaissant dans plusieurs sections d'une même page. Sinon la collection peut être désignée par l'id de son conteneur.
2. En cas d'espace entre des éléments qui n'en ont pas, assurez-vous que cela ne proviennent pas des marges ou des paddings.
3. En cas de mise en page mystérieusement cassée, assurez-vous qu'il n'y a pas de retour à la ligne entre une balise `anchor` et une `image` ou span incrustée.
4. Vérifier que les quatre coins de chaque élément sont visibles, par exemple avec un outil de modèle boite basé sur le navigateur (ex : *Web Developer Toolbar* pour Chrome, *Firebug* pour Firefox, *Opera Dragonfly*, ou *IE Developer Toolbar*) pour identifier les chevauchements. Si le coin droit est masqué à cause d'une trop grande largeur fixe de l'élément, réduire celle-ci. Si c'est la hauteur qui cache le coin du bas, on peut réduire la hauteur ou bien désactiver le chevauchement, ou encore repositionner les voisins qui avaient été placés en absolu, en relatif, ou avec des marges négatives.
5. Utiliser les menus contextuels des styles appliqués dans *IE Developer Toolbar* pour voir quelles règles en cascade écrasent les voulues.
6. S'assurer que l'élément parent utilisé pour augmenter la spécificité est disponible dans tous les templates.
7. Dans le cas où l'URL ne passerait pas dans le validateur HTML[1], copier-coller le code HTML.
8. Vérifier que le padding de chaque enfant ne force pas conteneur à s'étendre hors de ses dimensions, et réduire les dimensions en fonction, s'il es impossible d'éviter de définir à la fois le padding et la hauteur ou largeur dans le même élément.
9. Vérifier lors du test de mise en page avec les barres d'outils du navigateur, qu'aucun bug *min-height* n'est déclenché en ajoutant ou retirant des menus du navigateur.
10. Vérifier que les derniers styles définis ne sont pas écrasés par les existants avec plus de spécificité, ou par ceux définis ensuite en augmentant le poids du sélecteur en utilisant les références aux ID et classes parents.
11. Utiliser l'inspecteur (CTRL + Shift + C) pour vérifier les écrasements des règles CSS localisées (les lignes rayées), pour voir s'il convient d'augmenter le poids d'un des sélecteurs.
12. S'assurer d'augmenter le poids des sélecteurs spécifiques à IE lorsqu'on le fait pour les styles universels.
13. Vérifier la largeur et la hauteur des dimensions du conteneur autorisent suffisamment d'espace pour fournir le positionnement vertical et horizontal déclaré en CSS.
14. Vérifier que la largeur et la hauteur de tous les éléments flottant tient dans le conteneur.

> **Demande de traduction**
>
> *Cette page nécessite une traduction.*
> *Si vous connaissez cette langue et le sujet de la page, vous pouvez poursuivre ou corriger la traduction.*

Make sure that the width of right aligned content such as label copy is only as large as needed to avoid using left margins or padding for alignment since it may not be consistent across browsers

Make sure that floated elements with wrapping content have a defined width and proper text alignment

Make sure that any containers that expand on :hover have the a default width and height which is greater than or equal to the size of any nested elements

Make sure that fixed-width containers use overflow:hidden so shrinkwrapped nested blocks don't cause margin collapsing in older versions of Firefox

Make sure that all inline elements are absolutely positioned or contained in block level elements

Change the position attributes of the container to make sure it allows absolute or relative placement of nested elements

Create a copy of the template and remove all divs except the misaligned one and its adjacent divs to debug positioning issues

Put a container div around block level elements that don't respond to display:inline or display:inline-block declarations

Remove margins from inner divs when debugging ignored positioning values in absolutely positioned divs

Remove all inline text and start with a non-breaking space as the placeholder content to debug width and height issues

Set margin, padding, font-size, and border size to zero for debugging unseen inheritance issues

Use display:block with a fixed height if margins are inconsistent for inline text in situations such as creating a header using markup and one-pixel high element with margin offsets to create rounded corners

Use a unique fixed width for each row instead of margin offsets to create rounded corners inside of inline-block containers for IE6 and IE7

Use a unique background-position for each row to create rounded corners with a background image covering half of the container but not the other

Use IE specific styles when absolutely positioned elements or margins are inconsistent with other browsers

Use a fixed width container when absolutely positioned elements do not appear within the correct relatively positioned parent in IE

Use position:relative on a container with overflow:scroll or auto if content with position:absolute or relative behaves like position:fixed upon scrolling in IE

Use obvious background colors to highlight misplaced or misaligned nested elements when debugging inner divs

Use inline styles and remove ID and class references to display elements that are invisible due to an unknown CSS error

Use word-wrap:break-word(CSS3 Property) or wbr(and its alternatives) to break lines within long words to prevent overflow of an unbreakable string from causing horizontal scrollbars or inadvertent wrapping of blocks to a new line

Use table specific font rules in IE5 set since those applied to html and body tags do not cascade to table elements

Use divs instead of paragraphs when margins don't take effect

Use divs instead of paragraphs to avoid the inherit vertical margins between paragraphs

Use display:inline on an absolutely positioned elements to avoid hasLayout issues

Use margins to move absolutely positioned elements instead of directional shifts

Use border-collapse: collapse; and border-spacing:0 to remove gaps between table columns

Use border styling on the td element instead of the tr element after doing the above to create grid lines on a table

Use empty-cells: hide to remove borders from empty table cells

Use position:relative on inline text and position:absolute on block-level containers and media

Fixe

Use inline-block to give hasLayout to a relatively positioned container

Make sure class selectors aren't inadvertently overridden by default values of ID selectors, especially background-position changes for sprites, by dividing the properties of ID selectors using longhand notation

Use vertical-align middle to line up inline-block elements in IE; use vertical-align:top for form elements

Use overflow:visible on parent elements if negative margins don't reveal overflow content

Create more classes with specific names instead of creating more complex parent=>child references

Make sure there's not a more specific CSS rule or one that comes later in the style sheet which is overriding the desired effect

Make sure the there's not a number at the beginning of the ignored class or ID name

IE will only change the state of a hovered child element when something changes on the anchor tag, so use a redundant effect like a:hover{ visibility:visible; } if something like a:hover span { display:block; } doesn't work

IE will only change the state of a hovered child element that is referenced using a parent class or id if a property is set on the anchor tag that hasn't been declared in any other pseudo classes for that anchor, such as text indent:0, float:none, direction:inherit, visibility:inherit, white-space:normal, etc ex. .class a:hover span{} needs .class a:hover{visibility:inherit;}

IE will increase the width of a container with an italicized font style so use overflow:hidden to avoid inconsistent wrapping if possible

If links on a page are clickable in IE only, look for an absolutely positioned element that overlaps the links and raise the z-index value of the link container or reposition the layout so the overlap does not happen

If the property value for a group of elements doesn't work, redefine that property value for one of those elements to see if a comma is missing or an unsupported selector nullified the entire rule

```
#business_photo, .sentence, .instruction, .list &amp;gt; li { padding-bottom: 24px; }

/* Redefine the same padding value because the descendant selector nullifies the above line in IE6 */
#business_photo { padding-bottom: 24px; }
```

If neighboring elements are mysteriously repositioned when an event or hover triggers the display or a submenu or other hidden content, make sure that content is not relatively positioned in external CSS, inline CSS, or generated JavaScript attributes.

If margins between a series of lists with headers are inconsistent due to special cases such as scrollbars for overflow in long lists or different margins for the first or last list, remove margins from the lists themselves and put them on the headers and use padding at the top and bottom of the container to handle spacing between the first or last list and the borders

If a border does not show up on around an element, make the color black or white to see if the color is too light or dark to distinguish from the background

If a div is being pushed beneath its specified position to a new line, use the mouse to select the content within the div to see if the dimensions of a block element within it are causing overflow outside the container. Make sure to set the width of any forms within the container in addition to validating the HTML to make sure no block tags are missing or inline tags have block tags nested within them.

If a hover or visited state of a hyperlink is not working correctly, do the following checks: -Make sure the pseudo classes are defined in the correct order -Make sure the pseudo classes don't share the same definition as a less-specific element, like a:hover{color: #ccc;} and .mylink:hover{color: #ccc;} since the browser will group them together and parse the less-specific rules in the cascade before the more specific rule, such as .mysite{color: #eee;}. If this is the case, add a reference to the parent container to the pseudo class rules, for example .mymenu .mylink:hover{color: #ccc;} to increase the selector weight. -Make sure the pseduo classes are defined or the element selector instead of the class selector, for example .nav a:hover vs nav a.submenu:hover. This may not work in IE6. -Make sure the rules are defined in the proper order, which is the following: 1. :link, :visited 2. :hover, :focus 3. :active

Since a link can simultaneously be in the :visited, :hover, and :active states, and they have the same specificity, the last one listed wins.

Increase or decrease the text inside a container to make sure it wraps text and resizes properly

Increase or decrease the font size of the browser a few times to see how it affects your backgrounds and margins, especially for text used in headings and navigation

Apply classes to different elements to make sure each class is not dependent upon a specific tag

Include several block elements such as a few paragraphs and a large list inside a div to see if it breaks the flow of the container

Set position:relative on floated elements if they are not visible in IE6 or IE7

Set a fixed width or height for a data table cell if word wrapping causes any nested containers with relative dimensions to expand. If there is a div container with a percentage width inside any table cell, content that forces word wrapping in another cell causes the relative width of the div will grow in IE6 and IE7 to match the overflow width of the table cell if an explicit width or height is not set on the cell to contain the content

Reset the line-height of an element if it stretches larger than its defined height in IE

Use clear to break an element away from a floated sibling and wrap it to a new line

Separate HTML structure and CSS design by developing the naming conventions and organization of the container styles without thinking about the content

Create a reusable library of headers, footers, submenus, navigation bars, lists, forms, and grid layouts

Use consistent semantic style names for classes and IDs

Design modules to be transparent on the inside

Extend objects by applying multiple classes to an element

Use CSS reset styles, web-safe fonts, and grids on all projects

Avoid location dependent styles that break outside of the cascade

Avoid using a class name that describes the tag or property it is mapped to, such as .hidediv or .floatleft

Avoid using IDs to style content within the main layout containers

Avoid using images to replace text

Avoid drop shadows and rounded corners over irregular backgrounds

Avoid nesting divs inside of spans or having spans adjacent to divs to prevent triggering side-effects caused by the behavior of anonymous block boxes

Using 1px dotted borders in IE6 is buggy and will display a dashed border. Use an IE6 specific style of 2px border-width and a lighter shade of the desired color to offset the larger pixel size.

Any vertical-align value besides top or bottom will be rendered inconsistently across browsers for form elements

Only use the !important declaration to create user style sheets or to override an inline style if the HTML markup is not editable. The !important declaration must follow the value of the property that it intends to give weight to. Since inline styles can be added to any HTML tag, using the style attribute selector is the only way to override those styles if a class or ID attribute is not defined for the element, as in the following example:

```
div[style]{ background-color: #f00 !important; }
```

When using abs pos elements, make them the last child element of the rel pos container in the HTML source order for IE6

Use an abs pos span nested inside an href to create dropdowns that show up in different places

Use the nested span trick to highlight the parent container of a nav list by setting it's background color to the desired nav background and making it visible on hover. Make the ul the rel pos parent and give it overflow hidden, and make the empty span bottom:0; z-index:-1; height: 99em; and width equal to the nav

```
<ul id="nav"><li><a href="#"><span></span>Test 1</a></li></ul>
```

Use the nested span trick to create polygonal shaped links by offsetting their background positioning and placing them over the desired content

Use a:focus{width:0,height:0,outline:0} to remove dotted borders

Use abs/rel positioning to emulate outlines in IE6/7

Use resets to debug margin/padding issues

Fixe

Use margins is remove padding and top/bottom positioning for abs positioned elements if spacing is off

Use :default to select all radio button sets

Use :active as :focus in IE6

Make sure parent div used for specificity is available in all templates

Use relative/absolute div with a background color to replace text with ellipsis using the follwing steps:

Create a content div with the original content and a height equal to the font-size * the number of rows you want visible

When a JavaScript event is triggered, reduce the height to the new number of rows you want visible by adding a class with that setting

Make sure class selectors aren't inadvertently overridden by default values of ID selectors, especially background-position changes for sprites, by dividing the properties of ID selectors into pieces

Use vertical-align middle to line up inline-block elements in IE; use vertical-align:top for form elements

Use change events to delegate input and select elements

Use active link states and relative positioning to respond to link clicks

Use the following alternatives for buffering offscreen content: -Negative margins instead of display:none -Relative positioning offsets instead of visibility:hidden for text -background-position instead of visibility:hidden for images -display:inline block instead of display:table -clip:rect instead of max-width/height

Absolutely positioned elements moved outside of their relatively positioned parents in Firefox 2 cause the containers to stretch

Use margins for the outermost elements

Use padding for elements that have a background color or image

-moz-only-whitespace matches an element that has no child nodes at all or empty text nodes or text nodes that have only white-space in them. Only when there are element nodes or text nodes with one or more characters inside the element, the element doesn't match this pseudo-class anymore.

-moz-selection (::selection) pseudo-element applies to the portion of a document that has been highlighted (e.g. selected with the mouse) by the user.

The :-moz-first-node pseudo-class matches an element that is the first child node of some other element. It differs from :first-child because it does not match a first child element with (non-whitespace) text before it.

The :-moz-last-node pseudo-class matches an element that is the last child node of some other element. It differs from :last-child because it does not match a last child element with (non-whitespace) text after it.

The ::-moz-list-bullet pseudo-class is used to edit the bullet of a list element.

The :default pseudo-class is used to identify a user interface element that is the default among a group of similar elements

Use -ms-text-align-last for orphan lines at the end of paragraphs

Use xsl:number to mimic generated content for counters in IE

Use outlines to style buttons with white accents

Use q { quotes: "" "" } to define quote characters

Use the following to trigger quote character display:

```
q:before { content: open-quote }
q:after  { content: close-quote }
```

Use fixed positioning with @media print to show header and footer containers on every page

Use absolute:positioning instead of fixed positioning to avoid showing header and footer containers on every page

Préfixes automatiques

Certains outils permettent d'automatiser l'ajout des préfixes dépendant des navigateurs[2], comme gulp.js[3].

Voir aussi

- https://github.com/yangshun/front-end-interview-handbook

Références

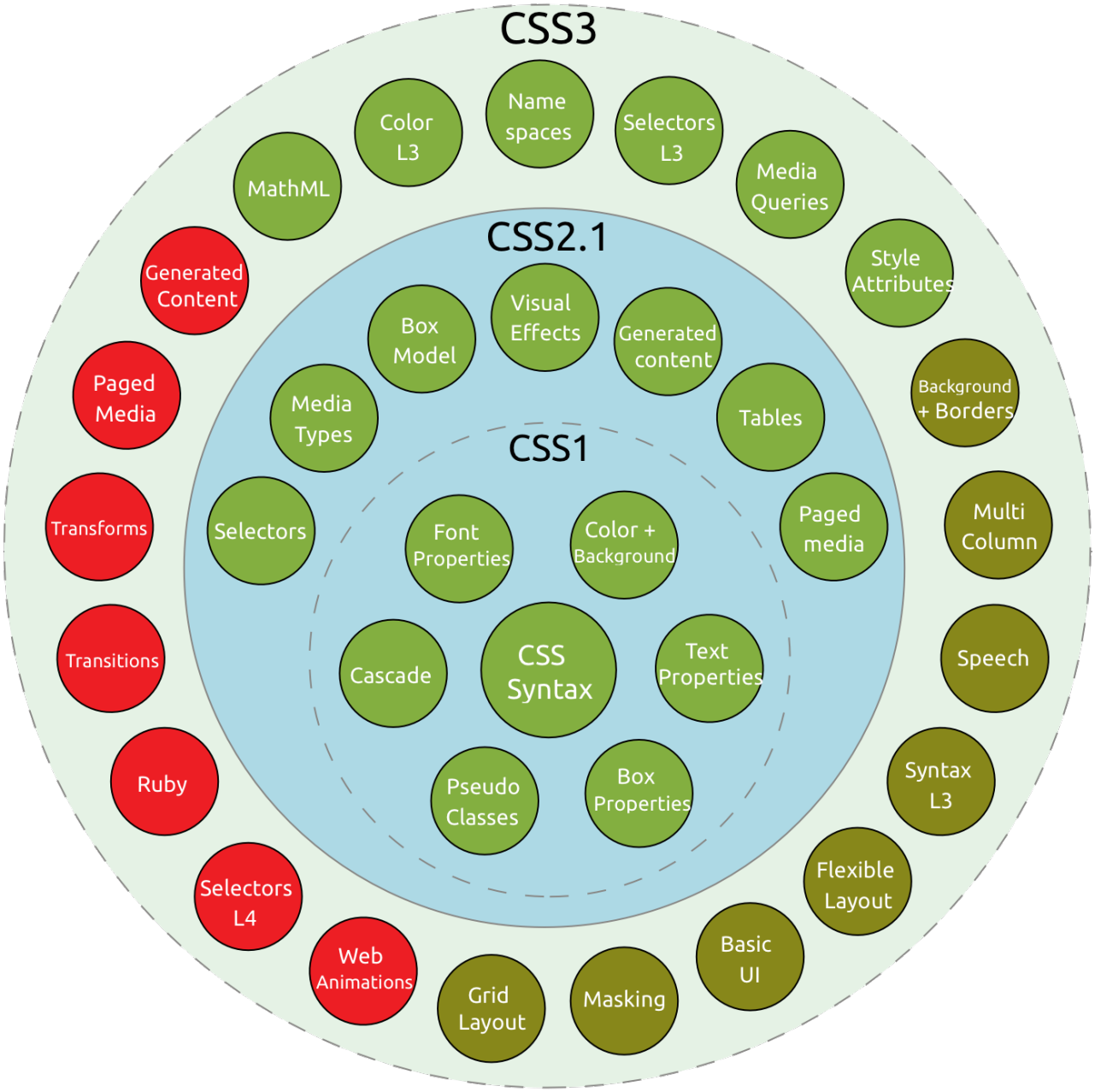
1. https://validator.w3.org/
2. http://www.alsacreations.com/article/lire/1306-prefix-free-prefixes-CSS3.html
3. http://www.alsacreations.com/tuto/lire/1686-introduction-a-gulp.html

Fixe

CSS 3

Description

Le CSS3 a été lancé en 1999 pour ajouter une couche de nouveaux mots clés.



Sélecteurs

Pour sélectionner des éléments[1] :

- + : adjacence directe
- ~ : adjacence indirecte
- ^= : de début d'attribut
- $= : de fin d'attribut
- *= : de contenu d'attribut
- |= : de début d'attribut suivi par un tiret (-)[2].

Pseudo-classes

1. :active
2. :any
3. :any-link
4. :checked
5. :default
6. :dir
7. :disabled
8. :empty
9. :enabled
10. :first
11. :first-child

12. :first-of-type
13. :fullscreen
14. :focus
15. :hover
16. :indeterminate
17. :in-range
18. :invalid
19. :lang
20. :last-child
21. :last-of-type
22. :left
23. :link
24. :not
25. :nth-child
26. :nth-last-child
27. :nth-last-of-type
28. :nth-of-type
29. :only-child
30. :only-of-type
31. :optional
32. :out-of-range
33. :read-only
34. :read-write
35. :required
36. :right
37. :root
38. :scope
39. :target
40. :valid
41. :visited

Pseudo-éléments

1. ::after
2. ::backdrop
3. ::before
4. ::first-letter
5. ::first-line
6. ::selection

Fonctions

- `calc()` réalise des calculs comprenant les quatre opérations fondamentales : addition, soustraction, multiplication et division. Exemple : `width: calc(100px - 2em); padding: 1em;` permet à l'élément de ne pas déborder à cause de son padding.
- `var()` fait l'appel à une variable déja attribuée. Exemple : `--maCouleur: #08F; color: var(--maCouleur);`


Cette section est vide, pas assez détaillée ou incomplète.

flexbox


Cette section est vide, pas assez détaillée ou incomplète.

Références

1. https://www.w3schools.com/cssref/css_selectors.asp
2. https://developer.mozilla.org/fr/docs/Web/CSS/S%C3%A9lecteurs_d_attribut

Fixe

Bootstrap

Présentation

Bootstrap est un framework CSS et JavaScript développé par Twitter depuis 2011. Une présentation rapide et exhaustive de ses composants est disponible sur https://hackerthemes.com/bootstrap-cheatsheet/.

Gestion des tailles d'écran

Afin de construire des sites adaptatifs (*responsive*), il est déconseillé de définir les mêmes tailles d'objet (en pixels) sur tous les écrans.

Bootstrap solutionne cela en permettant de gérer une grille où les sélecteurs peuvent varier selon les tailles d'écrans, par media queries.

Exemple

Le code suivant définit du CSS différent pour chacune des quatre tailles d'écran par leur bornes. Voici un exemple avec quatre tailles[1] :

- xs (extra small) : < 768 px par défaut.
- sm (small) : entre 768 inclus et 992 px exclus.
- md (medium) : entre 992 inclus et 1200 px exclus.
- lg (large) : ≥ 1200 px.

```
@media (max-width: @screen-xs-max) { ... }
@media (min-width: @screen-sm-min) and (max-width: @screen-sm-max) { ... }
@media (min-width: @screen-md-min) and (max-width: @screen-md-max) { ... }
@media (min-width: @screen-lg-min) { ... }
```

Références

1. https://getbootstrap.com/docs/3.3/css/

Fixe

Glossaire

A

- attribut

B

C

Canevas

Représente l'espace fermé dans lequel la structure de formatage est traitée. Pour l'écran d'ordinateur, il s'agira de la zone de visualisation du navigateur ; pour une page papier, il s'agira de l'espace imprimable de la page ; etc.

Recommandation CSS2 - Le canevas (http://www.yoyodesign.org/doc/w3c/css2/intro.html#q4) (FR)

D

E

F

G

H

I

J

K

L

M

N

O

P

Plate-forme

Désigne généralement le type de système d'exploitation d'un ordinateur. On parlera de la plate-forme Linux, la plate-forme Macintosh, la plate-forme Windows, etc.

- propriété

Q

R

S

- sélecteur

Structure de formatage

Tout document HTML ou XML donne lieu à la construction d'un *arbre du document* reflétant l'organisation de ses contenus et de leur structure. A partir de l'arbre du document, le moteur de rendu CSS d'un navigateur produit une structure dite « de formatage » qui est utilisée pour appliquer les règles de style aux éléments. La structure de formatage est déduite de l'arbre du document, mais peut en différer lorsque des contenus sont générés ou supprimés via CSS.

T

U

V

W

X

Y

Z

Voir aussi

- Catégorie:Lexique en français de la programmation sur le Wiktionnaire

Fixe

Mots réservés

Propriétés

```
align-content
align-items
align-self
animation
animation-delay
animation-direction
animation-duration
animation-fill-mode
animation-iteration-count
animation-name
animation-play-state
animation-timing-function
backface-visibility
background
background-attachment
background-blend-mode
background-clip
background-color
background-image
background-origin
background-position
background-repeat
background-size
border
border-bottom
border-bottom-color
border-bottom-left-radius
border-bottom-right-radius
border-bottom-style
border-bottom-width
border-collapse
border-color
border-image
border-image-outset
border-image-repeat
border-image-slice
border-image-source
border-image-width
border-left
border-left-color
border-left-style
border-left-width
border-radius
border-right
border-right-color
border-right-style
border-right-width
border-spacing
border-style
border-top
border-top-color
border-top-left-radius
border-top-right-radius
border-top-style
border-top-width
border-width
bottom
box-decoration-break
box-shadow
box-sizing
break-after
break-before
break-inside
caption-side
clear
clip
color
column-count
column-fill
column-gap
column-rule
column-rule-color
column-rule-style
column-rule-width
column-span
column-width
columns
content
counter-increment
counter-reset
cursor
direction
display
empty-cells
filter
flex
flex-basis
flex-direction
flex-flow
flex-grow
flex-shrink
flex-wrap
float
font
font-family
font-feature-settings
font-kerning
font-language-override
font-size
font-size-adjust
font-stretch
font-style
font-synthesis
font-variant
font-variant-alternates
font-variant-caps
font-variant-east-asian
font-variant-ligatures
font-variant-numeric
font-variant-position
font-weight
hanging-punctuation
height
hyphens
image-orientation
image-rendering
image-resolution
ime-mode
justify-content
```

Fixe

```
left
letter-spacing
line-break
line-height
list-style
list-style-image
list-style-position
list-style-type
margin
margin-bottom
margin-left
margin-right
margin-top
mark
mark-after
mark-before
marks
marquee-direction
marquee-play-count
marquee-speed
marquee-style
mask
mask-type
max-height
max-width
min-height
min-width
nav-down
nav-index
nav-left
nav-right
nav-up
object-fit
object-position
opacity
order
orphans
outline
outline-color
outline-offset
outline-style
outline-width
overflow
overflow-wrap
overflow-x
overflow-y
padding
padding-bottom
padding-left
padding-right
padding-top
page-break-after
page-break-before
page-break-inside
perspective
perspective-origin
phonemes
position
quotes
resize
rest
rest-after
rest-before
right
tab-size
table-layout
text-align
text-align-last
text-combine-upright
text-combine-upright
text-decoration
text-decoration-color
text-decoration-line
text-decoration-style
text-indent
text-justify
text-orientation
text-overflow
text-shadow
text-transform
text-underline-position
top
transform
transform-origin
transform-style
transition
transition-
transition-delay
transition-duration
transition-timing-function
unicode-bidi
vertical-align
visibility
voice-balance
voice-duration
voice-pitch
voice-pitch-range
voice-rate
voice-stress
voice-volume
white-space
widows
width
word-break
word-spacing
word-wrap
writing-mode
z-index
```

CSS3

column-gap, flex-wrap, skewY(10)...

Cette section est vide, pas assez détaillée ou incomplète.

Références

- http://cssreference.io/

Fixe

Notions de design

En CSS, la description d'une couleur peut se faire soit en la nommant parmi un échantillon de couleurs indexées, soit en modulant ses composantes rouge, verte et bleue, soit en utilisant des couleurs système.

Lorsque l'on utilise un nom parmi les mots-clés suivants : aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white et yellow. Par exemple :

```
background-color: white; /* met le fond en blanc */
```

Décrire une couleur selon le système rvb peut se faire de plusieurs manières :

- soit en hexadecimal, sous la forme « #RRVVBB » où RR est la quantité de rouge dans la couleur, VV la quantité de vert et BB, chaque quantité variant de 00 (minimum) à FF (maximum), par exemple :

```
background-color: #c8d7e3; /* met le fond dans une couleur déclarée en hexadecimal */
```

- soit sous la forme hexadécimale raccourcie : « #RVB », où chaque composante est sur un chiffre (le chiffre est automatiquement doublé pour obtenir un nombre sur 2 chiffres), par exemple :

```
background-color: #c0f; /* équivaut à #cc00ff */
/* forme pratique pour utiliser l'une des 216 couleurs standards web */
```

- soit en indiquant les valeurs de rouge, vert et bleu en nombre décimal (entre 0 et 255) ou en pourcentage (entre 0% et 100%), par exemple :

```
background-color: rgb(75,56,124);
background-color: rgb(67%,34%,78%);
```

*Pour plus de détails voir : **w:Rouge vert bleu**.*

- soit en décrivant une couleur par le système TSL, comme suit: en degré, la teinte (entre 0 et 360), en pourcentage, la saturation et la lumiére (entre 0% et 100%), par exemple :

```
background-color: hsl(0,100%,50%); /* c'est le rouge */
background-color: hsla(0,100%,50%, 0.5); /* le rouge une autrefois mais 50% (0.5) transparent */
```

Enfin, les couleurs système sont liées aux préférences de l'utilisateur, telles qu'elles sont exprimées par la configuration de son interface client. Voir la documentation officielle (http://www.yoyodesign.org/doc/w3c/css2/ui.html#system-colors) pour davantage de détails.

Le css ne permet pas de décrire une couleur par le système CMJN, qui fait référence dans le domaine de l'imprimerie.

Voir aussi

- Rédaction technique/De l'usage des couleurs dans un document

Fixe

Interface HTML

Comme vous l'avez appris. Le CSS et le HTML sont complémentaires. Le CSS contient toutes les informations concernants la mise en forme et le document HTML, la donnée. Nous allons voir ici comment faire pour indiquer, dans un document HTML, quel(s) code(s) CSS s'applique(nt). Pour cela, il faudra distinguer les codes s'appliquant :

- sur plusieurs pages
- sur une seule page
- à un seul élément

Intégrer une feuille de style externe

C'est la façon de faire la plus courante : dans l'élément `<head>` on ajoute un élément `<link>` :

```
...
<head>
  ...
  <link rel="stylesheet" type="text/css" href="fichier.css" media="screen" />
</head>
...
```

le fichier CSS est un simple fichier texte qui contient le code CSS qui s'appliquera à toute la page. L'utilisation de cette balise permet très facilement d'utiliser une feuille de style pour plusieurs documents HTML.

Intégrer du style directement dans la page

Dans le cas où on voudra que le code CSS ne s'applique qu'à une page : on pourra l'inclure directement dans le code HTML.

```
<html>
  <head>
    <style type="text/css">
      /* Ici le code CSS */
    </style>
  </head>
  ...
</html>
```

Intégrer du style directement dans un élément HTML

Lorqu'un style s'applique à un seul élément : plutôt que de lui créer une classe, on peut inclure le code dans l'élément lui-même grâce à l'attribut `style`.

```
<elt style="prop1: val1; prop2: val2; prop3: val3">
```

par exemple : un paragraphe en rouge.

```
<p style="color: red">
  le corps du paragraphe
</p>
```

Définir une feuille de style selon le média

CSS permet de différencier la présentation d'un même document (X)HTML en fonction du média visé, par exemple pour l'adapter :

- à l'écran d'ordinateur de bureau
- à la projection en grand écran
- au rendu sur l'écran réduit d'un mobile ou PDA
- à l'impression papier
- au rendu sonore via une synthèse vocale

La plupart des propriétés CSS peuvent s'appliquer à tous les médias. Cependant, il existe également des propriétés spécifiques à un ou à certains médias. Par exemple :

- `display` permet d'inclure ou d'exclure du rendu un contenu (X)HTML quel que soit le media (y compris donc une lecture par une synthèse vocale).
- à l'inverse, `voice-family` permet d'indiquer les familles de voix utilisées par une synthèse vocale lisant le contenu du document, tandis que `font-family` indique les familles de polices de caractères utilisées pour son affichage à l'écran ou son impression papier.

Chaque feuille de style devrait donc se limiter aux seules propriétés et valeurs adaptées à un ou plusieurs médias, et être accompagnée de la mention du ou des types de média visés. Ceci permettra notamment d'éviter le téléchargement de styles inexploitables ou produisant un résultat inapproprié selon le navigateur et le média concerné.

Le ou les types de médias visés peuvent être indiqués :

- soit lors de la liaison HTML, grâce à l'attribut `media` des éléments `link` et `style` :

```
<link rel="stylesheet" href="styles.css" media="screen" />
```

```
<style type="text/css" media="screen">
  @import url(styles.css);
</style>
```

- soit dans la feuille de style elle-même en précisant le type de média visé par une règle `@import`, ou en regroupant plusieurs styles dans une règle `@media` :

```
@import url(styles.css) screen;

@media screen {
   body {...}
   p {...}
}
```

Lorsqu'une feuille de style s'applique à une sélection de plusieurs médias, ceux-ci sont alors séparés par une virgule :

```
<link rel="stylesheet" href="styles.css" media="screen, projection, handheld" />
```

En l'absence de mention du média, une feuille de style s'applique par défaut à tous les médias. La mention du type de média `all` a le même effet :

```
<link rel="stylesheet" href="global.css" />
<link rel="stylesheet" href="global.css" media="all" />
```

Les principales valeurs de type de média sont, pour celles qui sont actuellement supportées par les navigateurs :

- `all` : quel que soit le média de sortie (reconnu par tous les navigateurs Web traditionnels)
- `screen` : écrans d'ordinateur de bureau (reconnu par tous les navigateurs Web traditionnels)
- `print` : pour l'impression papier (reconnu par tous les navigateurs Web traditionnels)
- `handheld` : écrans de très petite taille (mobiles, PDA. Cependant, de nombreux navigateurs pour mobiles reconnaissent également le type de média `screen`)

- `projection` : quand le document est projeté (Opera en mode plein écran)

Deux autres types de médias sont plus rarement reconnus par les logiciels de rendu, et sont encore peu exploités actuellement :

- `tv` : quand le document est affiché sur un appareil de type télévision, caractérisé par une résolution moyenne
- `speech` : pour le rendu via une synthèse vocale (actuellement disponible dans Opera. Ce type de média n'est en revanche pas pris en compte par les logiciels de lecture d'écran actuels utilisés en particulier par les personnes handicapées visuelles)

Certains type de média restent théoriques, fautes de support par les agents utilisateurs :

- `tty` : si le document est consulté par une application en mode texte (utile pour consulter les documentations lors de l'administration d'un serveur)
- `braille` : lorsque le document sera rendu sur une tablette braille
- `embossed` : pour l'impression braille.

Enfin, l'ancien type de media `aural`, destiné aux synthèses vocales, a été déprécié et remplacé depuis CSS2.1 par `speech`.

Les feuilles de styles alternatives

Les feuilles de styles alternatives permettent de proposer un choix de plusieurs rendus différents pour un même document (X)HTML. Ce choix s'apparente à celui d'un thème graphique.

Un lien vers une feuille de style alternative se définit comme le lien vers la feuille de style par défaut. Cependant :

- l'attribut `rel` change et devient `alternative stylesheet`.
- L'attribut `title` permet de définir et de différencier chaque présentation alternative. En présence de styles alternatifs, une feuille de stylé dénuée d'attribut `title` devient *permanente*, et s'applique quelque-soit le style alternatif choisi.

```
<link rel="alternative stylesheet" href="une_feuille_alternative.css" title="titre" media="screen" />
```

Un exemple avec trois feuilles alternatives : une simple, par défaut. Une autre, à dominantes noire. Une dernière, à dominantes bleues. Notez également la première feuille de style, permanente, qui s'appliquera quel que soit le style alternatif choisi :

```
<link rel="stylesheet" href="common.css" />
<link rel="stylesheet" href="defaut.css" title="Simple, par défaut" media="screen" />
<link rel="alternative stylesheet" href="noir.css" title="Thème à dominantes noire" media="screen" />
<link rel="alternative stylesheet" href="ciel.css" title="Thème bleu" media="screen" />
```

Cependant, les styles alternatifs ne sont que très partiellement exploités par les navigateurs graphiques actuels.

Un exemple complet

Cet exemple présente une page d'un site de cours. Cette page est la seule du site qui regroupe les définitions. Certaines sont importantes : elles seront mises en valeurs. Une définition est particulière...

Cet exemple a été conçu pour exploiter toutes les notions abordées sur cette page.

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <title>Une page</title>
    <!-- une feuille de style correspondant à la charte graphique du site
    Elle est incluse sur toutes les pages pour rendre le site homogène -->
    <link rel="stylesheet" href="site.css" title="Simple, par défaut" media="screen" />

    <!-- une feuille de style pour l'impression. elle rend
    le texte noir sur blanc et éliminant les couleurs de fond -->
    <link rel="stylesheet" href="impression.css" media="print" />

    <!-- une feuille de style alternative où le texte est blanc sur noir -->
    <link rel="alternative stylesheet" href="negatif.css" title="Noir sur blanc" media="screen" />

    <style type="text/css">
      /* Cette page est la seule de site à inclure
      des définitions de la classe importante */
      dt.importante
        {
        color: red;
        text-decoration: underline;
        }
    </style>
  </head>
  <body>
    <h1>Glossaire</h1>
    <p>Cette page est une liste de définitions, dont certaines sont importantes :</p>
    <dl>
      <dt>un terme</dt>
        <dd>une définition</dd>
      <dt>un terme</dt>
        <dd>une définition</dd>
      <dt class="important">un terme important</dt>
        <dd>une définition</dd>
      <dt>un terme</dt>
        <dd>une définition</dd>
      <dt>un terme</dt>
        <dd>une définition</dd>
      <dt class="important">un terme important</dt>
        <dd>une définition</dd>
      <dt>un terme</dt>
        <dd>une définition</dd>
      <dt>un terme</dt>
        <dd>une définition</dd>
      <dt>un terme</dt>
        <dd>une définition</dd>

      <!-- ici, une définition particulière. c'est la seule donc il
      n'est pas nécessaire de créer une classe supplémentaire -->
      <dt style="color: blue">un terme particulier</dt>
        <dd>une définition</dd>

      <dt>un terme</dt>
        <dd>une définition</dd>
    </dl>
  </body>
</html>
```

Voir aussi

Le chapitre *Zones de mise en forme* du wikilivre *Programmation HTML*.

Fixe

Fixe