9

Introduction à CSS

La création de styles CSS (Cascading Style Sheets ou feuilles de style en cascade) est le complément indispensable du langage XHTML. Ce procédé correspond parfaitement à la séparation du contenu et de la présentation sur laquelle nous avons plusieurs fois insisté en décrivant les différents éléments XHTML. D'une part, cette séparation permet d'alléger les pages en centralisant les définitions des styles en un point unique, une seule définition pouvant s'appliquer à un grand nombre d'éléments. D'autre part, elle facilite également la maintenance et l'évolution des sites par voie de conséquence. Elle apporte aussi une plus grande rigueur dans la conception des pages et peut permettre un travail collaboratif entre plusieurs programmeurs travaillant en parallèle, d'où une réduction des délais de fabrication. À l'attention de ceux pour qui ces points peuvent paraîtrent marginaux, nous pouvons ajouter que les styles CSS apportent une bien plus grande richesse créative que ne le permettait le langage HTML utilisé sans CSS. Pour égaler les possibilités graphiques de l'association XHTML et CSS, il aurait fallu alourdir de quantité d'éléments spécifiques le langage XHTML, alors que la tendance actuelle est à l'opération inverse, pour rapprocher XHTML de XML. Dans ce chapitre d'introduction aux styles CSS, nous allons donc aborder les bases indispensables à la compréhension de leur mécanisme. Après les règles générales d'écriture d'un style qui s'avèrent simples, nous envisagerons toutes les nombreuses possibilités d'écriture des sélecteurs qui permettent d'appliquer le style voulu à l'élément voulu, quel que soit son contexte. Nous terminerons cette introduction par l'étude des différentes méthodes d'insertion des styles dans une page, puis par les règles de cascade (le « Cascading » de CSS) et d'héritage, dont la connaissance permet de gérer les situations complexes d'attribution des styles à un élément.

Créer des styles

Les règles générales

Avant d'aborder les différents méandres de la création de styles, il faut assimiler quelques règles de base et en particulier la syntaxe de la déclaration d'un style (nous parlerons souvent par la suite de « style » au lieu de « déclaration de style »).

Une déclaration de style comporte plusieurs parties, selon l'ordre suivant :

- Un sélecteur qui va déterminer à quel élément et éventuellement dans quelles conditions va s'appliquer le style. Autant que les propriétés, c'est la variété des sélecteurs qui fait la richesse de CSS.
- La déclaration des propriétés que l'on veut voir appliquées à l'élément sélectionné. Elle doit être incluse entre des accolades ouvrante ({) et fermante (}).
- Dans ces accolades doivent apparaître une ou plusieurs propriétés déterminées chacune par un mot-clé propre à CSS suivi du caractère deux-points (:), puis de la valeur attribuée à cette propriété. Si nous définissons plusieurs propriétés dans le même style, il faut séparer chaque déclaration de la précédente par le caractère point-virgule (;). Les propriétés sont en nombre limité et font l'objet d'une recommandation du W3C (voir l'annexe B). La version actuelle de CSS est la version 2.1 (au 13 juin 2005), dans laquelle ont été éliminées un certain nombre de propriétés qui n'étaient mises en application par aucun navigateur. C'est cette version que nous emploierons ici. À chaque propriété correspond un domaine de valeurs particulier constitué par exemple de mots-clés ou de nombres. Vous trouverez dans l'annexe B la liste des propriétés CSS 2.1 et l'éventail de leurs valeurs. Signalons enfin que l'utilisation d'une propriété ou d'une valeur erronée ne provoque pas d'erreur lors de l'exécution comme ce serait le cas dans un langage de programmation. Ces fausses définitions sont simplement ignorées.

La figure 9-1 résume la syntaxe d'écriture d'un style.

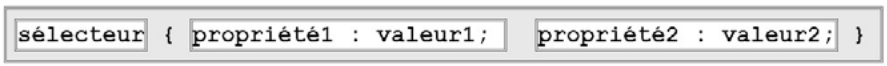


Figure 9-1
Syntaxe d'écriture d'un style

Sur ce modèle, nous pouvons par exemple écrire le style suivant :

```
div {color : red ; background-color :yellow ;}
```

dans lequel `div` est le sélecteur, `color` est la première propriété qui détermine la couleur du texte de l'élément, `red` la valeur attribuée à cette couleur, `background-color` qui désigne la couleur de fond est la seconde propriété et `yellow` sa valeur. Tous les éléments `<div>` de la page dans laquelle se trouve cette déclaration ont donc un contenu écrit en rouge sur fond jaune.

Validation du code CSS

Comme pour le code XHTML, il est possible de vérifier la validité des feuilles de style CSS en se connectant sur le site `http://jigsaw.w3.org/css-validator/`.

Le validateur signale les erreurs et permet d'apporter les corrections nécessaires. La figure 9-2 montre la page d'accueil du site dans lequel on peut valider le code CSS inclus dans une page XHTML (avec l'extension `.htm` ou `.html`) ou dans un fichier externe (avec l'extension `.css`) situé sur un serveur (repère ❶), ou encore inclus dans un fichier local à condition qu'il ne contienne que du code CSS (repère ❷)

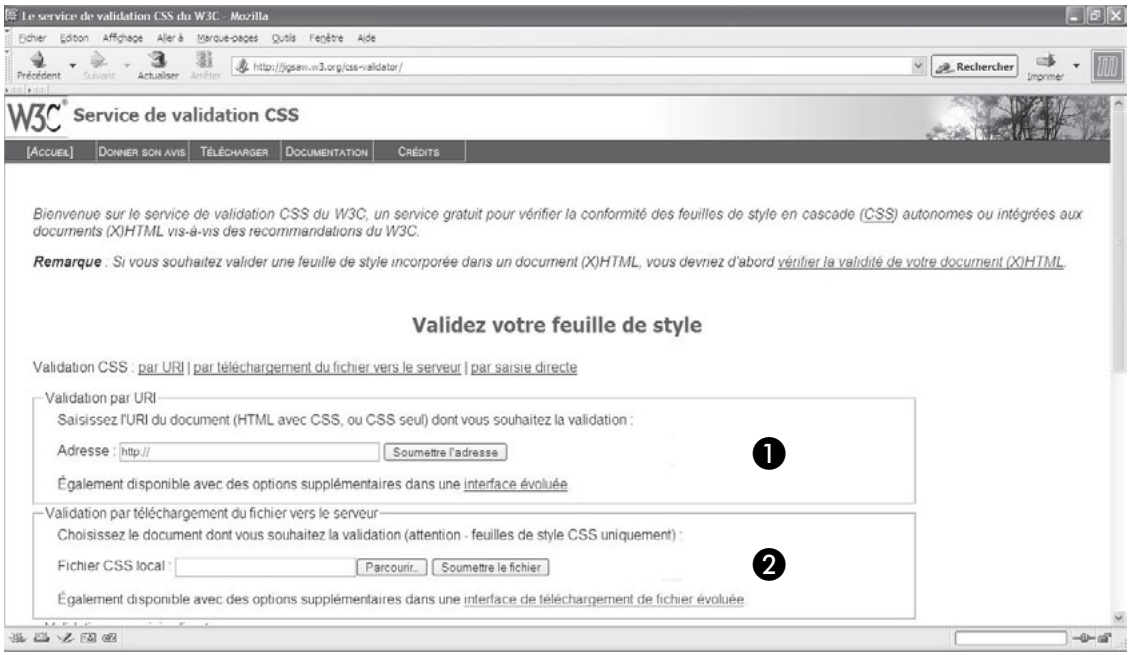


Figure 9-2
La page de validation des feuilles de style

Les sélecteurs

Une des grandes richesses de CSS est la multiplicité des possibilités de sélection des éléments auxquels on veut attribuer un style donné. Cette très grande diversité permet en effet d'appliquer un style aussi facilement à tous les éléments, en une seule ligne de code, qu'à un unique élément isolé dans la page web, sans avoir à écrire la définition localement. De plus, la combinaison de plusieurs sélecteurs dans la même déclaration ouvre la voie à une quasi-infinité de combinaisons, propres à répondre à tous les besoins, même les plus complexes.

Sélectionner un seul élément

Il s'agit de la sélection la plus simple, puisque le sélecteur est constitué du nom de l'élément sans les caractères de début < et de fin de balise />. Nous écrivons par exemple :

```
p {color : yellow ; background-color :blue;}
```

pour que le texte de tous les paragraphes figure en jaune sur fond bleu.

Nous pouvons ainsi définir un style propre à chaque élément comme il en existe un par défaut dans les navigateurs.

Sélectionner plusieurs éléments

Nous pouvons très facilement appliquer le même style à plusieurs éléments différents en les énumérant et en les séparant par une virgule dans le sélecteur. Plutôt que de multiplier les définitions :

```
h1 {color : black ; background-color : red;}
div {color : black ; background-color : red;}
p {color : black ; background-color : red;}
```

nous pouvons écrire le style suivant :

```
h1,div,p {color : black ; background-color : red;}
```

Cette possibilité de regroupement peut être utile pour définir des styles communs à un ensemble d'éléments en écrivant ce type de sélecteur pour cet ensemble, puis en ajoutant d'autres propriétés spécifiques à un des éléments de la liste. On définit ainsi une sorte de tronc commun à un groupe, puis on affine chacun de ses composants.

Si nous écrivons par exemple le code suivant :

```
h1,div,p {color : black ; background-color : red;}
div {margin : 20px;}
```

l'élément `<div>` va avoir à la fois un texte noir, un fond rouge et une marge de 20 pixels, car la propriété `margin`, définie uniquement pour l'élément `<div>`, s'ajoute à celles déjà définies pour le sélecteur d'éléments `h1,div,p`.

Le sélecteur universel

Pour appliquer un style à tous les éléments, nous utiliserons le sélecteur universel * avant la définition d'une ou plusieurs propriétés. Par exemple, pour que la couleur du fond de tous les éléments soit le jaune, nous écrirons :

```
*{background-color : yellow;}
```

Cela n'empêche pas de modifier cette couleur de fond pour un élément particulier, en la redéfinissant uniquement pour celui-ci, par exemple :

```
*{background-color : yellow;}
p{background-color : gray;}
```

Dans ce cas, tous les éléments ont un fond jaune, sauf `<p>` qui a un fond gris redéfini spécialement.

Les classes

Nous avons vu que tous les éléments XHTML possèdent l'attribut `class`. Ce dernier permet d'appliquer un style défini dans une classe à un élément dont l'attribut `class` se voit attribuer le nom de cette classe. Pour créer une classe, le sélecteur est constitué du nom choisi pour la classe précédé d'un point (.). Le nom de la classe peut être un mot quelconque, en évitant quand même les noms des propriétés CSS et des éléments XHTML car cela occasionnerait des confusions. Nous pouvons par exemple définir la classe nommée `evidence` en écrivant le code :

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

À ce stade, la classe est abstraite et ne s'applique à aucun élément. Pour mettre en évidence un paragraphe précis de la page avec un texte rouge, nous devons alors écrire dans le code XHTML :

```
<p class="evidence">Texte contenu du paragraphe</p>
```

Le texte des autres paragraphes a toujours la couleur qui lui a été attribuée par ailleurs ou la couleur par défaut (noire).

Les classes présentent l'intérêt de pouvoir s'appliquer à n'importe quel élément, n'importe où dans le code de la page. Notre classe `evidence` peut donc s'appliquer à un titre `<h1>`, une division `<div>` ou un élément `<span>`, simplement en écrivant pour chacun d'entre eux l'attribut `class="evidence"`.

Nous pouvons également définir une classe en la déclarant applicable seulement à un élément en faisant précéder son nom de celui de l'élément. Nous pouvons écrire par exemple :

```
div.jaune {color : yellow;}
```

Dans ce cas, seules les divisions ayant un attribut `class` dont la valeur est `jaune` ont un texte jaune. Les autres éléments, et même s'ils ont le même attribut avec la même valeur, n'ont pas de style défini dans cette classe.

Le sélecteur universel `*` peut également être employé à la place du nom d'un élément dans la définition d'une classe. Le style s'applique alors à tous les éléments dont l'attribut `class` a pour valeur le nom de la classe. Nous écrivons alors par exemple :

```
div.jaune {color : yellow;}
```

Il est possible de définir d'abord une classe abstraite, puis de la particulariser en ajoutant une autre propriété pour un élément qui utilisera la même classe. Dans le code CSS ci-après :

```
.rouge {color : red;}

div.rouge {background-color : blue;}
```

suivi du code XHTML dans la page,

```
<div class="rouge">Texte contenu de la division </div>
```

le texte contenu dans l'élément `<div>` est affiché en rouge sur fond bleu.

Appliquer plusieurs classes au même élément

L'avantage de définir des classes abstraites est, nous l'avons vu, qu'elles peuvent s'appliquer à n'importe quel élément. Leur puissance peut être multipliée car nous pouvons appliquer plusieurs classes indépendantes à un même élément. Celui-ci a alors la combinaison des propriétés de chacune des classes. Pour utiliser plusieurs classes dans le même élément XHTML, il faut donner à son attribut `class` la liste des noms des classes en les séparant par un espace comme ceci :

```
<div class="classe1 classe2"> Ceci est un texte avec la classe 1 et 2 </div>
```

Les combinaisons d'emploi des classes sont alors multiples, chaque classe pouvant définir une caractéristique, et chaque élément pouvant en utiliser plusieurs au choix.

L'exemple 9-1 en donne une illustration. Nous y définissons cinq classes. La première, nommée `jaune`, (repère ❶) permet de vérifier le fonctionnement du sélecteur universel dans une classe. Elle s'applique au titre `<h1>` qui utilise la classe `jaune` et est affichée en vert. La deuxième, qui porte le même nom que la précédente, est affectée à l'élément `<div>` et crée un texte en jaune (repère ❷). Quand l'attribut `class` d'un élément `<div>` a pour valeur `jaune`, ce n'est pas la première classe qui est utilisée mais la seconde. La troisième crée un texte en rouge (repère ❸), la suivante un texte en italique (repère ❹) et la dernière crée un fond gris (repère ❺). Les différents éléments `<div>` de la page utilisent ces différentes classes. Le premier (repère ❼) utilise celle qui est nommée `classe1` et son texte est donc rouge. Le deuxième (repère ❽) utilise les deux classes nommées `classe1 et classe2`, et il combine à la fois un texte rouge et en italique. Le troisième (repère ❾) utilise les classes nommées `classe1 et classe3` et a donc un texte rouge sur fond gris. Enfin, la dernière (repère ❿) utilise les classes `jaune, classe2 et classe3`. Elle a donc les propriétés de ces trois classes, à savoir un texte jaune, en italique et un fond gris.

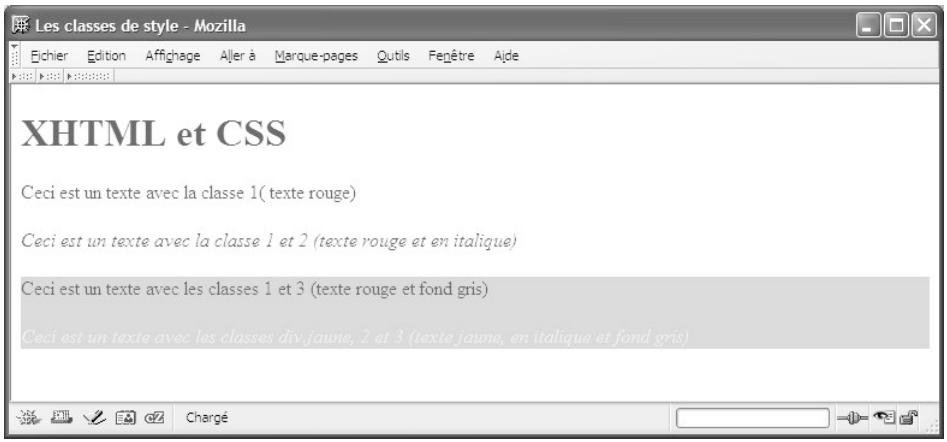
Exemple 9-1. Utilisation de plusieurs classes

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr">
 <head>
  <title> Les classes de style </title>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
  <style type="text/css" title="classes">
   *.jaune{color: green;}❶
   div.jaune {color: yellow;}❷
   .classe1 {color: red;}❸
   .classe2 {font-style: italic}❹
   .classe3 {background-color: #CCC;}❺
  </style>
 </head>
 <body>
  <h1 class="jaune">XHTML et CSS</h1>❻
  <div class="classe1">❼
```

```
    Ceci est un texte avec la classe 1(texte rouge)<br /><br />
  </div>
  <div class="classe1 classe2">❽
    Ceci est un texte avec la classe 1 et 2 (texte rouge et en italique)<br /><br />
  </div>
  <div class="classe1 classe3">❾
    Ceci est un texte avec les classes 1 et 3 (texte rouge et fond gris)<br /><br />
  </div>
  <div class="jaune classe2 classe3">❿
    Ceci est un texte avec les classes div.jaune, 2 et 3 (texte jaune, en italique
    ➥ et fond gris)
  </div>
 </body>
</html>
```

La figure 9-3 donne un aperçu du résultat obtenu.

Figure 9-3
L'utilisation des classes multiples



Sélecteur d'identifiant id

Pratiquement, chaque élément peut avoir un attribut `id` qui doit être unique dans une page donnée. Nous pouvons écrire un style qui ne sera applicable qu'à l'élément dont l'`id` a une valeur précise en donnant cette valeur au sélecteur (comme pour une classe) et en le faisant précéder du caractère dièse (#).

En écrivant le style suivant :

```
div {color: black;}
#bleu {color: white; background-color: blue;}
```

puis le code XHTML :

```
<div id="bleu">Texte en blanc sur bleu</div>
<div>Texte en noir </div>
```

seul l'élément contenant l'attribut id="bleu" bénéficie du texte en blanc sur fond bleu, et aucun autre, car l'identifiant doit être unique dans une page, et les autres éléments ont un texte noir. Ce type de définition est très ciblé car le style ne peut s'appliquer qu'à un seul élément du fait de l'unicité de l'identifiant id.

La casse des identifiants

Les valeurs de l'attribut id sont sensibles à la casse en XHTML ; bleu et Bleu sont donc deux identifiants différents (sauf dans Internet Explorer pour qui la casse importe peu).

Les sélecteurs d'attributs

Il est également possible d'appliquer un style à un élément déterminé dès qu'il possède un attribut donné, quelle que soit la valeur de cet attribut. Pour appliquer ce sélecteur, le nom de l'élément doit être suivi du nom de l'attribut placé entre crochets ([) et (]). En définissant le style suivant :

```
acronym[title] {color: red; background-color: gray;}
```

tous les éléments <acronym> qui possèdent un attribut title, quelle que soit sa valeur, ont un contenu affiché en rouge sur fond gris, ce qui permet d'attirer l'attention du visiteur afin qu'il laisse le curseur sur le contenu pour voir apparaître la bulle d'aide donnant la signification de l'acronyme (voir l'exemple 9-2 repères ❶, ⓫ et ⓬).

De même, en définissant le style pour l'élément <img />

```
img[longdesc] {border-color: red; border-weight: 2px;}
```

toutes les images ayant un attribut longdesc ont une bordure rouge de deux pixels de large (voir l'exemple 9-2 repères ❷, ⓭ et ⓮). Nous pouvons également créer un style applicable à tous les éléments qui possèdent un attribut donné en utilisant le sélecteur universel * placé devant les crochets qui contiennent le nom de l'attribut choisi. Si nous définissons le style suivant :

```
*[title] {background-color: yellow;}
```

ce sont tous les éléments ayant l'attribut title qui sont affichés avec un fond jaune.

Comme pour les classes, il est possible de sélectionner plusieurs attributs pour un élément en faisant suivre son nom de plusieurs attributs entre crochets. Si nous écrivons le style suivant :

```
h2[title][id]{background-color: yellow;}
```

seuls les titres <h2> ayant à la fois les attributs title et id ont une couleur de fond jaune (voir l'exemple 9-2 repères ❹ et ❾).

Les sélecteurs de valeur d'attribut

Le sélecteur précédent applique un style à un élément par la seule présence d'un attribut précis. Pour affiner ce système, nous pouvons également appliquer un style à un élément à condition que tel attribut ait une valeur précise en utilisant la syntaxe suivante :

```
element [attribut="valeur"] {Définition du style;}
```

En écrivant le style suivant :

```
code[title="code JavaScript"] {color: blue;}
```

tout le contenu des éléments `<code>` ayant l'attribut `title` dont la valeur est la chaîne `code JavaScript` sera affiché avec une police de taille 12 pixels (voir l'exemple 9-2 repères ❺ et ⓯). La comparaison des valeurs est effectuée au caractère près, par conséquent un élément `<code>` dont l'attribut `title` aurait la valeur `code JavaScript`, avec simplement un espace supplémentaire, ne répondrait pas à la condition posée.

Il est ici possible de particulariser davantage l'application du style en sélectionnant plusieurs attributs et leurs valeurs en utilisant la syntaxe :

```
element[attribut1="valeur1"][attribut2="valeur2"] {Définition du style;}
```

Par exemple, avec la définition suivante, appliquée à une cellule de tableau :

```
td {font-size: 12px;}

td [title="nom"] [align="center"] {font-size: 14px; color: red;}
```

seules les cellules ayant un attribut `title` avec la valeur `nom` et un attribut `align` ayant la valeur `center` sont affichés avec un texte rouge et avec une taille de fonte de 14 pixels, alors que le texte des autres cellules a une taille de 12 pixels.

Les exemples précédents impliquent que l'attribut ait exactement la valeur fixée. Il est possible d'étendre encore ce sélecteur en attribuant un style à tous les éléments, dont un attribut donné à une valeur qui ne correspond que partiellement à une chaîne donnée. Pour se voir attribuer le style, les éléments pourront contenir autre chose en plus de la valeur fixée. Pour obtenir cette sélection, il faut utiliser la syntaxe suivante, dans laquelle le signe = est remplacé par ~= :

```
element [attribut ~="valeur"] {Définition des styles;}
```

En écrivant par exemple le style suivant :

```
td[id ~="nom"]{background-color: #222;color: white;}
```

toutes les cellules du tableau dont l'attribut `id` contient la valeur `nom` seront affichées avec un fond gris foncé et en caractères blancs (voir l'exemple 9-2 repères ❻, ❼, ⓰ et ⓱). Cette possibilité pourrait par exemple être exploitée quand un tableau est construit dynamiquement par un script, l'attribut `id` étant créé comme la concaténation de la chaîne `nom` d'un espace et d'un nombre entier (les chaînes `nom1` ou `prenom` ne conviennent pas).

Si nous appliquons ce style au code XHTML suivant :

```
<tr>
 <td id = "nom 1"> Engels</td>
 <td id = "prenom"> Jean</td>
```

```
</tr>
<tr>
 <td id = "nom 2"> Geelsen</td>
 <td id = "prenom"> Jan</td>
</tr>
<tr> <td id="editeur"> Eyrolles </td> <td>Paris</td></tr>
```

seules les premières cellules des deux premières lignes du tableau ont un contenu affiché en blanc sur fond gris.

Une dernière possibilité consiste à attribuer un style à un élément dont la valeur d'un attribut donné commence par une chaîne fixée. Pour cela, le signe égal (=) doit être précédé du signe | selon la syntaxe suivante :

```
élément [attribut |="valeur"] {Définition des styles;}
```

Dans l'exemple suivant :

```
    td[id |="nom"] {font-style: italic;}
```

toutes les cellules de tableau créées par les éléments `<td>` dont l'attribut `id` commence par la chaîne `nom` devraient avoir un contenu affiché en italique. Je précise qu'à ce jour aucun navigateur ne gère ce sélecteur.

L'exemple 9-2 résume toutes les possibilités d'utilisation des sélecteurs de valeur d'attribut que nous venons d'aborder.

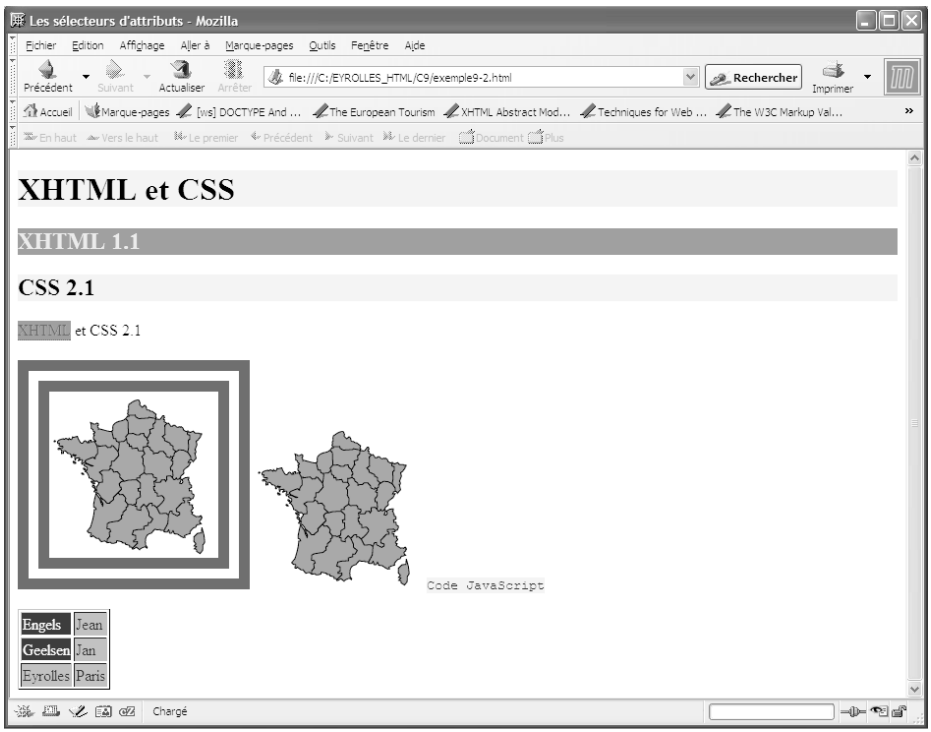
Exemple 9-2. Les sélecteurs d'attributs

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xml:lang="fr" xmlns="http://www.w3.org/1999/xhtml">
 <head>
  <title>Les sélecteurs d'attributs</title>
  <meta http-equiv="Content-type" content="text/html; charset=iso-8859-1" />
  <link rel="shortcut icon" type="images/x-icon" href="../images/favicon.ico" />
  <style type="text/css">
   acronym[title] {color: red; background-color: gray;}❶
   img[longdesc]{border: 2em red double}❷
   *[title]{background-color: #EEE;}❸
   h2[title][id]{background-color: gray; color: yellow;}❹
   code[title="code JavaScript"]{color: blue;}❺
   td{background-color: #AAA; color: blue;}❻
   td[id ~="nom"]{background-color: #222; color: white;}❼
  </style>
 </head>
 <body>
  ❽ <h1 title="Les outils du Web" >XHTML et CSS</h1>
  ❾ <h2 title="Les outils du Web" id="xhtml">XHTML 1.1</h2>
  ❿ <h2 title="Les outils du Web" >CSS 2.1</h2>
  <p>⓫ <acronym title="eXtensible HyperText Markup Language">XHTML</acronym> et
  ➥ ⓬ <acronym>CSS 2.1</acronym><br /><br />
```

```
    ⓭<img src="../images/france.gif" alt="France"
    ➥ longdesc="funhtml.com/france.html" />
    ⓮<img src="../images/france.gif" alt="France" />
    ⓯<code title="code JavaScript">Code JavaScript</code>
   </p>
   <table border="1">
    <tr>
     ⓰<td id = " nom 1 "> Engels </td>
     <td id = " prénom "> Jean </td>
    </tr>
    <tr>
     ⓱<td id = " nom 2 "> Geelsen </td>
     <td id = " prénom "> Jan </td>
    </tr>
    <tr>
     ⓲<td id="éditeur"> Eyrolles</td>
     <td> Paris </td>
    </tr>
   </table>
  </body>
</html>
```

La figure 9-4 montre les résultats obtenus pour ces définitions de styles.

Figure 9-4
Les sélecteurs d'attributs



Note
Internet Explorer ignore ces sélecteurs.

Les sélecteurs contextuels parent-descendant

Plutôt que de définir un style pour toutes les occurrences d'un élément, nous pouvons souhaiter ne l'appliquer qu'en fonction de sa position relative par rapport à un autre dans la hiérarchie des éléments de la page. Ce type de sélecteur est dit contextuel. Nous pouvons par exemple définir un style général pour l'élément `<p>` et vouloir lui en appliquer un autre quand il se trouve inclus dans un élément `<div>`. Pour cela, il faut utiliser la syntaxe suivante :

```
element_parent element_enfant {Définition des styles;}
```

En écrivant par exemple le style suivant :

```
p {color: blue;}
div p{color: red;}
```

et en l'appliquant au code XHTML ci-après :

```
<div>Texte de la division
<p>Texte du paragraphe inclus dans div</p></div>
<p>Texte d'un paragraphe non inclus dans div</p>
```

seuls les contenus des éléments `<p>` inclus dans `<div>` sont de couleur rouge, tous les autres étant bleus, et le texte inclus directement dans `<div>` a la couleur par défaut qui est le noir.

Il est aussi possible de préciser la hiérarchie en appliquant plus de deux éléments en les séparant tous par un espace.

Pour appliquer un style différent aux éléments de liste `<li>` inclus directement dans une liste non ordonnée `<ul>` ou dans une liste ordonnée `<ol>` elle-même incluse dans une liste non ordonnée `<ul>`, nous pouvons définir les sélecteurs suivants :

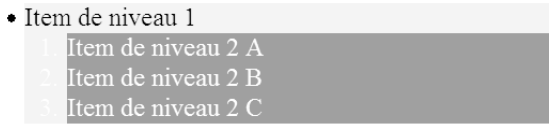
```
ul li {background-color: #EEE; color: black;}
ul li ol li {background-color: gray; color: white;}
```

Une fois qu'ils sont appliqués au code des listes ci-après :

```
<ul>
 <li> Item de niveau 1
  <ol>
   <li> Item de niveau 2 A </li>
   <li> Item de niveau 2 B</li>
   <li> Item de niveau 2 C</li>
  </ol>
 </li>
</ul>
```

nous obtenons pour les items de niveau 1 un texte noir sur fond gris, et pour les trois items de niveau 2, un texte blanc sur fond gris comme le montre la figure 9-5.

Figure 9-5
Les sélecteurs contextuels parent-descendant



Les sélecteurs parent-enfant directs

Nous pourrons préciser le sélecteur précédent en n'appliquant qu'un style à un élément à condition qu'il soit un enfant direct d'un autre élément et non plus un descendant indirect (comme dans la relation petit-enfant/grand-parent).

Pour opérer ce type de sélection, il faut utiliser la syntaxe :

```
element_parent > element_enfant {Définitions des styles;}
```

dans laquelle la présence d'un espace entre chaque élément et le symbole > est autorisée mais non signifiante.

En écrivant par exemple le style suivant :

```
span {color: blue;}
div > span{color: red; background-color: #EEE;}
```

et en l'appliquant au code XHTML suivant :

```
<div>Les standards <span>XHTML 1.1</span>❶ et <span>CSS 2.1</span>❷
 ➥ s'imposent aujourd'hui
<p>Texte d'un <span>paragraphe</span>❸ inclus dans div</p>
</div>
<p>Texte d'un <span>paragraphe</span>❹ non inclus dans div</p>
```

seul le contenu des paragraphes `<span>` qui sont des éléments enfants directement inclus dans une division `<div>` (repères ❶ et ❷) est affiché en rouge sur fond gris, les autres éléments `<span>` (repères ❸ et ❹) n'étant pas des enfants directs de `<div>` ont un texte bleu.

> **Note**
> Internet Explorer ignore ces sélecteurs.

Les sélecteurs d'éléments adjacents

Les sélecteurs précédents font appel à la notion de descendance parent enfant. Si on considère dans le code de la page deux éléments consécutifs enfants du même parent, comme deux éléments de bloc `<div>` inclus dans `<body>` par exemple, il n'existe pas de relation de descendance entre eux mais une relation de fratrie. Pour sélectionner ce type

de relation entre éléments, nous disposons du sélecteur + qu'il faut utiliser en adoptant la syntaxe suivante :

```
element1+element2 {Définitions des styles;}
```

Dans ce cas, le style s'appliquera à l'élément de type 2 uniquement s'il suit immédiatement un élément de type 1 dans le code XHTML sans y être inclus. Si nous définissons les styles suivants :

```
p {background-color: yellow; color: blue;}
div+p {background-color: blue; color: yellow;}
```

puis le code XHTML ci-après :

```
<div>Les standards XHTML 1.1 et CSS 2.1 s'imposent aujourd'hui
 <p>Texte d'un paragraphe enfant de div</p>
</div>
<p>Texte d'un paragraphe frère de div</p>
<p>Texte d'un paragraphe frère de div</p>
```

seul le deuxième élément `<p>` a le style : « texte jaune sur fond bleu », le premier n'étant pas frère mais enfant de l'élément `<div>`. Le troisième paragraphe aura de plus le même style que le premier car il ne suit pas immédiatement l'élément `<div>`.

On note que si les deux éléments sont reliés par le sélecteur +, seul le second présente un style défini. En effet, si nous définissons le code suivant au lieu des styles précédents :

```
p {background-color: yellow; color: blue;}
p+p {background-color: blue; color: yellow;}
```

et si nous l'appliquons au même code XHTML que précédemment, seul le troisième paragraphe a le style « texte jaune sur fond bleu ». Les premier et deuxième paragraphes bien qu'étant consécutifs n'ont pas le même parent direct. Notons encore qu'avec ce dernier style, si un élément `<div>` contenait plus de deux éléments `<p>`, le style « jaune sur bleu » créé avec le sélecteur `p+p` s'appliquerait à tous les paragraphes enfants de `<div>` sauf le premier. De même, s'il existait deux paragraphes consécutifs en dehors de l'élément `<div>`, le second bénéficierait également du même style.

Note

Internet Explorer ignore tout de ces sélecteurs.

Pseudo-classes et pseudo-éléments

Les sélecteurs précédents permettent d'attribuer un style à un ou plusieurs éléments bien définis dans la hiérarchie d'un document XHTML. Les pseudo-classes et les pseudo-éléments permettent d'attribuer un style à une partie abstraite d'un document non identifiable dans cette hiérarchie, par exemple le premier caractère ou la première ligne d'un paragraphe. D'autres pseudo-classes permettent d'attribuer un style à un document en fonction

des actions prévisibles mais non déterminées de l'utilisateur final, par exemple le fait de placer son curseur sur un lien ou un composant de formulaire.

Les pseudo-classes applicables aux liens

Deux pseudo-classes spécifiques aux éléments possèdent un attribut `href` faisant référence à un document externe (lien vers une autre page) ou interne (ancre vers une partie du même document). Il s'agit des pseudo-classes suivantes :

- `:link`, qui permet d'attribuer un style à un lien qui pointe vers un document non encore vu. C'est l'état normal de tous les liens à l'ouverture de la page.
- `:visited`, pour attribuer un style à un lien qui pointe vers un document déjà vu, après un retour sur la page d'origine.

Avec ces pseudo-classes, on pourra par exemple attribuer une valeur ou une taille de police spécifique au texte des liens visités ou pas. Nous les étudierons en détail au chapitre 12. Pour les employer, il faut faire précéder le nom de la pseudo-classe de celui de l'élément selon le modèle suivant concernant l'élément `<a>` :

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

Les pseudo-classes dynamiques

Elles permettent d'attribuer un style à un élément en fonction des actions effectuées par le visiteur. Ces pseudo-classes sont dynamiques car le style attribué disparaît avec le motif de leur création. Elles sont au nombre de trois :

- `:focus`, pour attribuer un style à l'élément qui a le focus, soit qu'il lui ait été donné par le code XHTML à l'aide des attributs `tabindex` ou `accesskey`, soit qu'il l'ait obtenu par un déplacement du pointeur provoqué par l'internaute. Le style disparaît quand l'élément perd le focus. Cette pseudo-classe est mal prise en compte par les navigateurs actuels. Nous pouvons définir par exemple les styles suivants pour affecter les éléments `<a>` et `<input />` quand ils reçoivent le focus :

```
a:focus{color: red;}
input:focus{background-color: blue;}
```

- `:hover`, pour attribuer un style à un élément visible dont la zone est survolée par le pointeur de la souris. Quand le pointeur quitte cette zone, le style est annulé, ce qui peut produire des effets visuels intéressants. Dans le style suivant, les divisions qui sont survolées par le curseur ont un fond rouge et un texte blanc le temps du survol.

```
div:hover{backround-color: red; color: white;}
```

- `:active`, pour attribuer un style à un élément dit actif, c'est-à-dire quand l'utilisateur clique sur son contenu. Là aussi, l'effet est transitoire et ne dure que le temps de l'activation de l'élément.

```
a:active{background-red; color: yellow;}
```

Nous utiliserons ces pseudo-classes de manière plus approfondie au chapitre 12. Les pseudo-classes dynamiques sont mal prises en compte par les navigateurs actuels.

Pseudo-classes diverses

On peut aussi mentionner l'existence de deux autres pseudo-classes d'usage assez général.

- `:first-child`, qui permet d'appliquer un style à un élément donné uniquement s'il est le premier enfant d'un autre élément. L'élément parent n'est pas nommé et peut être quelconque (à condition de respecter les règles d'inclusion parent-enfant de la DTD). En écrivant par exemple le style :

```
span:firsthild {background-color: red; color: white;}
```

  puis le code XHTML ci-après :

```
<div>Les standards tels que <span>XHTML 1.1</span> et <span>CSS 2.1</span>
➥ s'imposent aujourd'hui</div>
```

  seul le texte du premier élément `<span>` contenu dans `<div>` est de couleur blanche avec un fond rouge. De même, en écrivant le style suivant :

```
a:first-child {backgroor: red; color: white;}
```

  et en l'appliquant dans l'exemple de code XHTML :

```
<p> Liens utiles :
<a href= "w3.org" alt="w3c"> W3C </a>  
<a href= "php.net" alt= "PHP"> PHP</a>
</p>
```

  seul le texte « W3C » sera en rouge puisque le lien est le premier enfant du paragraphe `<p>`.

- `:lang(code)`, qui ressemble au sélecteur basé sur la valeur d'un attribut car il permet d'attribuer un style à un élément si son attribut `lang` a la valeur précisée par son code de langue. En écrivant par exemple :

```
p:lang (fr) {color: black;}
p:lang (en-us) {color: gray;}
```

  les paragraphes ayant l'attribut `xml:lang="fr"` sont affichés en caractères noirs et ceux ayant l'attribut `xml:lang="en-us"` seront affichés en gris. Cette pseudo-classe est mal prise en compte par les navigateurs actuels.

Les pseudo-éléments

Leur nom vient de ce qu'ils permettent d'agir sur une partie du contenu d'un élément comme s'il était contenu dans un nouvel élément fictif. On dénombre les quatre pseudo-éléments suivants :

- `:first-letter`, qui permet d'affecter un style à la première lettre du contenu d'un élément indiqué avant ce sélecteur. On l'utilise classiquement pour créer des effets de lettrines en définissant pour ce sélecteur une taille de caractères très supérieure à la taille de l'élément. En écrivant par exemple le style suivant :

```
p:first-letter {fonize: 300%; color: blue;}
```

la première lettre de chaque paragraphe sera trois fois plus grande que les autres et de couleur bleue. Le pseudo-élément :first-letter n'admet que les propriétés suivantes :

```
font, font-size, font-family, font-style, font-weight, color, background, margin,
padding, border, text-decoration, vertical-align, text-transform, line-height,
float, letter-spacing, word-spacing, clear.
```

- :first-line, qui permet d'affecter un style à la première ligne du contenu de l'élément indiqué. Cet affichage permet d'attirer l'attention sur un texte. En écrivant le style suivant :

```
div:first{font-size: 150%; font-weight: bold;}
```

la première ligne de chaque division sera affichée en gras et dans une taille 1,5 fois plus grande que la police en cours. Le pseudo-élément :first-line n'admet que les propriétés suivantes :

```
font, font-size, font-family, font-style, font-weight, color, background,
word-spacing, letter-spacing, text-decoration, vertical-align, text-transform,
line-height.
```

- :before, qui permet d'insérer un contenu doté d'un style particulier avant le contenu réel de l'élément précisé, en l'associant avec la propriété content. En écrivant le style suivant :

```
cite:beforeontent:"<<"; font-weight: bold;}
```

chaque contenu d'une citation <cite> sera précédé des caractères << en gras.

- :after, qui joue un rôle similaire au précédent mais définit un contenu doté d'un style à la fin du contenu de l'élément utilisé. En écrivant :

```
cite:after {cnt: ">>";font-weight: bold;}
```

chaque citation contenu dans l'élément <cite> sera suivie des caractères >> en gras.

La déclaration !important

Chaque déclaration de style peut revêtir un caractère de plus grande importance par rapport à une autre déclaration concernant le même élément et la même propriété qui comporte une valeur différente. Ces deux déclarations peuvent entrer en conflit au moment de la création de la présentation par le navigateur. Pour donner cette importance à un style, il faut insérer la déclaration d'importance à l'aide du mot-clé !important en le plaçant entre la valeur attribuée à la propriété et le point-virgule qui termine la déclaration. Dans l'exemple suivant :

```
*{color: black !important; background-color: yellow;}
div{color: blue; background-color: white;}
```

les couleurs de texte et de fond des sélecteurs * et div sont en conflit, mais comme la propriété color définie dans le sélecteur universel * est marquée !important, le texte de la division figure en noir. En revanche, le fond de la division est de couleur blanche car la valeur yellow n'est pas marquée !important et que la déclaration faite dans div est spécifique.

Nous reviendrons en détail sur les règles de priorités dans la section consacrée à la définition des effets de cascade en fin de chapitre.

Il est évidemment possible d'utiliser la déclaration `!important` pour plusieurs propriétés dans la même déclaration, par exemple :

```
*{color: black !important; background-color: yellow !important;}
div{color: blue; background-color: white;}
```

Dans ce cas, tous les éléments `<div>` ont un contenu affiché en noir sur fond jaune, et tous les styles définis en propre pour cet élément sont ignorés.

Écrire des feuilles de style

Nous allons envisager maintenant les différentes méthodes d'écriture des style CSS et la façon dont on peut les lier à un document XHTML.

Dans l'élément <style>

Défini dans la première partie de ce livre, l'élément `<style>` a pour vocation de renfermer les définitions des styles CSS utilisables dans la page qui le contient. Rappelons qu'il doit toujours être inclus dans l'élément `<head>` et qu'il ne peut contenir que des définitions de styles CSS et des commentaires XHTML délimités par `<!--` et `-->` ou des commentaires CSS délimités par /* et */.

Dans l'éventualité où toutes les pages d'un site ont en commun un certain nombre de styles et que chaque page possède quelques styles propres, les styles communs peuvent être écrits dans un fichier externe (voir la section suivante) et inclus dans l'élément `<style>` au moyen de la directive `@import` selon la syntaxe suivante :

```
@import url(fichier.css);
```

L'URL du fichier peut être relative ou absolue et elle peut être suivie de la désignation du média auquel les styles importés doivent s'appliquer spécifiquement. La page web a alors le comportement correspondant au cas où tous les styles contenus dans le fichier `fichier.css` sont écrits explicitement dans l'élément `<style>`. Cette directive doit figurer avant les autres définitions de style. Un élément `<style>` peut donc avoir la structure suivante, et comporter plusieurs directives `@import` :

```
<style type= "text/css">
 @import url(commun.css)all;
 @import url(ecran.css)screen;
 @import url(imprimante.css)print;
 div,p {font-style: italic;}
 h1,h2 {color: red;}
</style>
```

Si les styles définis après la directive `@import` sont en contradiction avec ceux qui sont contenus dans le fichier importé, les conflits éventuels sont tranchés selon les règles définies dans la section concernant les effets de cascade dans ce chapitre.

Dans un fichier externe

La tendance actuelle étant à la recommandation de la séparation du contenu et de la présentation des pages web, l'écriture des styles dans les fichiers externes est fortement conseillée, même si dans nos exemples nous ne l'utiliserons que très peu par commodité de présentation. Il s'agit de fichiers écrits en texte brut réalisables avec un éditeur simple comme le Bloc-notes de Windows mais aussi avec EditPlus ou même des éditeurs spécialisés qui fournissent une aide à la saisie. Le fichier ne devra contenir que des sélecteurs et les définitions des styles ainsi que des commentaires CSS (délimités par les caractères `/*` et `*/`) mais aucune balise d'élément XHTML. Le fichier CSS doit toujours être enregistré sous l'extension `.css` et être présent sur le serveur, tout comme les fichiers XHTML qui l'utilisent.

L'exemple de code suivant montre un fichier CSS nommé `commun.css`.

```
/* Styles communs à toutes les pages */
/* fichier:« commun.css » */
body {background-color: white; color: marine;}
h1 {color: black; font-size: 20px;}
div,p {font-size: 12px;}
a:link {color: blue;}
a:hover {color: red;}
```

Un fichier externe peut inclure les styles d'un autre fichier externe en faisant appel à la directive `@import` de la même façon que nous avons définie plus haut pour les éléments `<style>`.

Pour affecter un fichier de styles à une page de code XHTML, il faut utiliser l'élément `<link />` dans l'en-tête du document avec par exemple le code suivant :

```
<link rel="stylesheet" type="text/css" href="commun.css" media="screen"
➥ title="Styles de base" />
```

Cet élément a été vu en détail au chapitre 2 mais il n'est pas inutile de rappeler que l'attribut `href` contient l'adresse relative ou absolue du fichier CSS, ce qui n'interdit pas de lier un document XHTML à un fichier CSS présent sur un autre serveur. On peut utiliser autant d'éléments `<link />` que souhaité, chacun étant par exemple adapté à un type de terminal particulier précisé par l'attribut `media`.

Dans l'attribut style

Nous signalons cette possibilité pour mémoire car la DTD XHTML 1.1 autorise encore la présence de cet attribut. Cependant, il n'est pas conseillé de l'utiliser.

Il servait jusqu'à présent à créer des styles pour tout élément, en particulier pour créer des styles très ponctuels pour l'élément `<span>`.

Nous pouvons écrire par exemple :

```
<p> Le langage <span style="color: red "> XHTML </span> représente la dernière
➥ évolution du <span style="color: gray"> HTML </span> </p>
```

Dans ce cas, les mots « XHTML » et « HTML » ont respectivement les couleurs rouge et grise.

Il va de soi que ce type de code ne correspond en rien à la philosophie de l'association XHTML et CSS, qui commande une séparation du contenu et de la mise en forme. De plus, toute modification de ces styles demande une exploration de tout le code XHTML afin de repérer tous les attributs `style`, ce qui rend la maintenance plus longue à réaliser.

Nous abandonnerons cette possibilité encore offerte, mais appelée à disparaître. De nombreux moyens existent pour s'en passer. En effet, la création des classes de styles suivantes et leur utilisation via l'attribut `class` permettent d'obtenir le même effet que le code précédent.

```
/*Dans l'élément style*/
.red {color: red;}
.gris {color: gray;}
< !—Dans le corps du document XHTML-->
<p> Le langage <span class="red"> XHTML </span> représente la dernière évolution
➥ du <span class="gris"> HTML </span> </p>
```

La définition des styles s'y trouve centralisée dans l'élément `<style>` et ce n'est que leur utilisation qui est incluse dans le code XHTML. Il est donc plus aisé de procéder à des modifications selon les besoins.

Cascade et héritage

Savoir définir des styles est une chose, être sûr du résultat final en est une autre. Même si l'on a défini des styles dans l'élément `<style>` ou dans un fichier externe, il faut encore tenir compte, en dehors des problèmes d'interprétation propres aux différents navigateurs, du fait que l'on n'est pas le seul à avoir créé des styles pour des sélecteurs donnés. Les styles CSS ont en effet des origines diverses. Ils peuvent provenir du concepteur (c'est-à-dire de soi-même ou « l'auteur » selon le vocable du W3C). Il peut s'agir aussi de l'utilisateur final qui a la possibilité de définir sa propre feuille de style dans son navigateur. Même si les utilisateurs se prêtent rarement à cet exercice, cela est possible, en particulier pour ceux qui ont des problèmes de vue et qui vont définir par exemple un affichage en très gros caractères, ou ceux qui ont des problèmes de vision de couleurs et qui vont donc définir des choix leur permettant d'obtenir un contraste fond/texte lisible. La dernière origine des styles est le navigateur lui-même qui possède sa propre feuille de style par défaut, dont un modèle est édicté par le W3C, et dont vous trouverez la copie dans l'annexe B.

C'est la partie logicielle CSS du navigateur qui se charge de résoudre les conflits pouvant exister entre les différentes déclarations de styles attachées à un élément. Les différentes

opérations permettant d'attribuer finalement un style propre à un élément constituent ce que l'on nomme les règles de cascade de CSS (d'où le mot « Cascading » de CSS). Pour déterminer les priorités, les navigateurs font d'abord l'inventaire de toutes les déclarations qui s'appliquent à un élément donné, puis procèdent à la sélection en fonction de différents critères que nous allons envisager maintenant.

Sélection selon le média

Les styles pour lesquels le média spécifié ne correspond pas au moyen de visualisation sont éliminés. Le média peut avoir été indiqué dans les éléments `<link>` ou `<style>`, dans les deux cas au moyen de l'attribut `media` (qui peut prendre les valeurs `screen`, `print`, `projection`, `aural`, `braille`, `handheld`, `tty`, `tv` et `all`, dont nous avons déjà donné la définition), ou encore à la suite de la directive `@import` pour ceux qui sont importés directement dans l'élément `<style>`.

Sélection selon le créateur du style

Pour un média donné, des règles de priorité sont définies en fonction de l'origine des styles et du fait qu'ils soient marqués avec la déclaration `!important` ou pas. Les styles marqués `!important` l'emportent toujours sur ceux qui ne le sont pas. En cas d'égalité, les priorités de CSS 2.1 sont les suivantes, de la plus importante à la moins importante :

- Les styles de l'utilisateur employant la déclaration `!important`, ce qui lui permet d'avoir le dernier mot. Cela semble normal, mais ce n'était pas le cas avec CSS 1.
- Les styles du concepteur (l'auteur) marqué avec la déclaration `!important`.
- Les styles du concepteur non déclarés `!important`.
- Les styles du visiteur non déclarés `!important`.
- Les styles par défaut du navigateur passe en dernier et ne peuvent donc s'imposer à personne.

Les exemples qui suivent vont illustrer ces règles.

En écrivant dans le même fichier ou dans le même élément `<style>`, ou encore dans des fichiers différents, les styles suivants, nous obtenons des résultats divers selon les cas.

```
h1{color: blue !important;}
h1{color: red;}
```

L'élément `<h1>` a un contenu affiché en bleu car le style marqué par la déclaration `!important` l'emporte sur tous ceux n'ayant pas cette déclaration.

```
/*style de l'utilisateur */
h1{color: blue;}
/*style de l'auteur */
```

```
h1{color: red;}
/*style du navigateur */
h1{color: black;}
```

L'élément `<h1>` a un contenu affiché en rouge car les styles du concepteur, non déclarés `!important`, l'emportent sur ceux de l'utilisateur, également non déclarés `!important`, et toujours sur ceux du navigateur.

En revanche, avec les déclarations de styles suivantes :

```
/*style de l'utilisateur
h1{color:blue !important;}
/* style du concepteur
h1{color:red !important;}
```

c'est le style de l'utilisateur qui l'emporte quand il entre en conflit avec un style de l'auteur également déclaré `!important`, et l'élément `<h1>` a donc un contenu bleu.

Sélection par spécificité

Pour les styles non encore départagés par les conditions précédentes, on définit une spécificité pour chacun d'eux, à l'aide d'un nombre N de quatre chiffres, sous la forme `abcd`, et chaque chiffre est calculé de la manière suivante :

- `a=1` si le style est défini localement dans un attribut `style` et `a=0` sinon. Le nombre de la spécificité vaut alors 1000 et les autres nombres ne sont pas calculés, le style l'emportant sur tous les autres (sauf un style utilisateur marqué `!important`). Quand `a=0`, les chiffres suivants `b`, `c`, `d` sont évalués.
- Le chiffre `b` représente le nombre de sélecteur d'attributs `id` présents dans l'ensemble du sélecteur. Pour le sélecteur :

  ```
  div#gfonce {Définition des styles}
  ```

  nous avons `b=2`.

  Le chiffre `c` représente le nombre de classes, de pseudo-classes et de sélecteurs d'attribut présents dans le sélecteur. Pour le sélecteur :

  ```
  div.for a:hover{}
  ```

  le chiffre `c` vaut 2 (une classe `.force` et une pseudo-classe `:hover`).
- Le chiffre `d` représente le nombre d'éléments XHTML utilisés dans le sélecteur. Pour le sélecteur :

  ```
  div p a :be{}
  ```

  le chiffre `d` vaut 3 (trois éléments : `div`, `p` et `a`).

Les exemples de calcul des spécificités suivantes permettront de se familiariser avec ces règles, un peu complexes à assimiler d'emblée.

```
<h1 style="color: red;"> Texte </h1>
```

N=1000, ce style ne peut être dominé que par un style utilisateur déclaré !important (y compris s'il est lui-même déclaré de cette façon).

```
div.h1#force{décl}
```

N=0102 car le sélecteur contient un identifiant (#force) et deux éléments XHTML (div et h1).

```
h1.gras a.rouge {}
```

N=0022 car le sélecteur contient deux classes (.gras et .rouge) et deux éléments XHTML (h1 et a).

```
div#menu {background-color: #FC9;}
```

N=0101 car le sélecteur contient un sélecteur d'identifiant (#menu) et un élément (div).

Sélection selon l'ordre d'apparition

Dans le cas où, malgré tous les critères précédents, deux styles sont encore en conflit, l'ordre d'apparition les départage, le dernier apparu dans le code étant celui qui l'emporte. Ajoutons que les styles importés au moyen de l'élément <link /> sont réputés apparaître avant ceux qui sont écrits dans l'élément <style>. De plus, si plusieurs fichiers de styles sont importés dans le même en-tête <head>, les styles du dernier importé l'emportent en cas de conflit. Par exemple, les cas suivants peuvent se produire :

- Cas 1 :

  ```
  h1, h2 {color: yellow;}
  h1 {color: navy;}
  ```

  Dans ce cas, le titre <h1> a un texte de couleur navy car ce style apparaît en dernier et écrase le précédent.

- Cas 2 :
  Style écrit dans un fichier externe :

  ```
  /*Style écrit dans le fichier externe « monstyle.css »
  h1 {color: navy;}
  ```

  Styles liés et styles internes :

  ```
  Dans l'en-tête <head>
  <link rel="stylesheet" type="text/css" href="monstyle.css" />
  <style type="text/css">
  h1 {color: red;}
  </style>
  ```

  ou encore :

  ```
  <style type="text/css">
  @import url(monstyle.css)
  h1 {color: red;}
  </style>
  ```

Le titre `<h1>` a un texte de couleur `red` car les styles liés apparaissent avant ceux qui sont écrits dans l'élément `<style>`, même si l'élément `<link />` est écrit après l'élément `<style>` dans l'en-tête.

L'héritage

L'héritage est le fait qu'un élément enfant possède les mêmes styles que l'élément qui le contient (son parent dans la hiérarchie des éléments d'une page). Nous pouvons par exemple définir les styles suivants :

```
div{color: white; background-color: blue;}
```

Si dans le code XHTML de la page figurent les éléments suivants :

```
<div>Texte<p>Premier paragraphe <span class="">XHTML</span> et les styles
➥ <strong>CSS 2.1 </strong>sont indispensables </p> à tous !</div>
```

L'élément `<p>` est enfant de `<div>` et les éléments `<span>` et `<strong>` sont eux-mêmes enfants de `<p>`. Par héritage et bien sûr faute d'avoir définis des styles propres pour les éléments `<p>`, `<span>` et `<strong>`, ceux-ci ont un contenu qui possède les styles définis pour leur parent direct ou indirect `<div>`. Ils sont donc tous en blanc sur fond bleu. Si nous créons un style différent pour l'élément `<p>`, ses éléments enfants héritent alors de ces styles et non plus de ceux de l'élément `<div>`.

De même, si nous définissons un style pour les éléments de listes ordonnées ou non `<ol>` et `<ul>`, tous les items de la liste, quel que soit leur niveau d'imbrication, ont les mêmes caractéristiques par défaut sans qu'il faille créer un style propre pour eux.

L'héritage concerne un grand nombre de propriétés CSS que nous allons aborder dans les chapitres suivants, mais toutes les propriétés ne sont pas systématiquement héritées, par exemple les marges, les bordures, ou les dimensions et la position pour des raisons évidentes de mise en page. Depuis CSS 2, la quasi-totalité des propriétés peut prendre la valeur `inherit` permettant de définir explicitement l'héritage de la valeur que possède la même propriété dans l'élément parent. Toute modification opérée pour l'élément parent est donc répercutée à ses enfants. Comme trop de précision ne nuit pas, et qu'il est préférable de ne pas laisser l'initiative d'interprétation aux navigateurs, nous conseillons en cas de doute de définir clairement la propriété voulue pour un élément plutôt que de compter sur la réalisation ou non de l'héritage.

Les unités

Toutes les propriétés CSS peuvent prendre une valeur dans un domaine particulier propre à chacune d'elle. En dehors des nombreux mots-clés existants, nous allons faire ici l'inventaire des différents types de valeurs parmi les plus générales que l'on retrouve pour un grand nombre de propriétés.

Les unités de longueur

Elles s'appliquent aussi bien à la taille d'une police qu'à la largeur d'une bordure ou la hauteur d'un élément.

Elles s'expriment par un nombre entier ou décimal selon les cas, suivi d'une unité. On dénombre les unités suivantes :

- Les unités relatives
  - `em` : qui se réfère à la taille de la police utilisée ou à la valeur de la propriété `font-size` (voir le chapitre 12)
  - `ex` : qui correspond à la taille de la lettre « x » minuscule dans la police utilisée.
  - `px` : qui correspond à la taille de 1 pixel. Contrairement à une idée répandue, la taille de 1 pixel n'est pas une taille absolue car elle dépend du média de visualisation et de la distance entre l'œil et le média.
- Les unités absolues : elles sont recommandées quand les caractéristiques physiques (mesurables) du média sont connues.
  - `in` : soit un pouce anglais (un inch), donc 25,4 mm ;
  - `cm` : le centimètre ;
  - `mm` : le millimètre ;
  - `pt` : le point qui représente conventionnellement 1/72 de pouce ;
  - `pc` : le pica qui représente 12 points, soit 1/6 de pouce.
- Les pourcentages qui, comme chacun le sait, ne sont pas des unités mais une convention décriture, le symbole % représentant la fraction 1/100. Leur utilisation fait toujours référence à une autre dimension, celle de l'élément parent le plus souvent, ce qui permet de calculer la dimension voulue.

Les couleurs

Une valeur de couleur s'exprime en mettant en œuvre l'une des trois manières suivantes :

- Un mot-clé parmi une liste limitative donnée à l'annexe C. Tous les mots-clés sont en anglais, par exemple `black`, `yellow` qui correspondent à des couleurs connues ; d'autres sont plus fantaisistes comme `whitesmoke`.
- Un code hexadécimal de couleur basé sur les composantes RGB d'une couleur dans le système additif. Chaque composante prend une valeur qui va de 0 à FF, et l'ensemble doit être précédé du caractère dièse (#), par exemple `#F4C5A8`. Il est possible de ne préciser que trois nombres hexadécimaux de 0 à F, par exemple `#FC5`, les navigateurs convertissant ces valeurs par réplications (la couleur notée `#FC5` est interprétée comme `#FFCC55`).

- À l'aide de la fonction `rgb()` qui admet trois paramètres représentant la valeur des composantes RGB d'une couleur selon la syntaxe `rgb(Red, Green, Blue)`, chaque composante est exprimée par un nombre entier variant de 0 à 255 ou par un pourcentage de 0 à 100 %. On peut définir par exemple des couleurs de la façon suivante :

```
rgb(45, 78, 2ou encore rgb(25%, 85%, 12%).
```

Exercices

Exercice 1 : Écrivez la syntaxe générale de la déclaration d'un style.

Exercice 2 : Écrivez le sélecteur et le style donnant une couleur rouge et un fond noir aux éléments `<h1>` et `<h3>`.

Exercice 3 : Écrivez le sélecteur afin que tous les éléments de la page soient écrits en vert.

Exercice 4 : Écrivez une classe qui définit un fond jaune et appliquez-la aux éléments `<h1>` et `<p>`.

Exercice 5 : Écrivez une classe spécifique à un élément `<code>` afin que son texte soit bleu.

Exercice 6 : Écrivez les classes correspondant aux styles « fondgris », « textevert » et « textejaune ». Appliquez la première et la troisième à un paragraphe, puis la première et la deuxième à une division `<div>`.

Exercice 7 : Écrivez le sélecteur afin que l'élément dont l'attribut `id` vaut « menu » ait un fond rouge.

Exercice 8 : Écrivez le sélecteur afin que tous les éléments ayant un attribut id aient un texte noir sur fond jaune.

Exercice 9 : Écrivez le sélecteur afin que les éléments `<h1>` ayant un attribut `title` aient un texte bleu, les autres ayant un texte noir.

Exercice 10 : Écrivez les sélecteurs afin que les paragraphes inclus dans `<body>` aient un texte gris et que ceux inclus dans `<div>` aient un texte marron.

Exercice 11 : Écrivez le sélecteur afin que seuls les éléments enfants de <p> aient un texte bleu, tous les autres ayant un texte noir.

Exercice 12 : Pour une liste imbriquée sur deux niveaux, écrivez le sélecteur pour que les éléments `<li>` de premier niveau inclus dans `<ol>` soient rouges, et que ceux de second niveau soient en vert.

Exercice 13 : Écrivez le sélecteur pour appliquer un style différent à un élément `<li>` selon qu'il est inclus dans `<ol>` ou `<ul>`.

Exercice 14 : Écrivez le sélecteur pour que le survol d'un élément `<h1>` provoque le changement de couleur du texte en rouge.

Exercice 15 : Écrivez le sélecteur afin que seule la première ligne d'un paragraphe soit en rouge, le reste s'affichant en gris.

Exercice 16 : Où peut-on écrire des styles CSS ?

Exercice 17 : Écrivez des styles dans un fichier externe et les incorporer dans l'élément `<style>`.

Exercice 18 : Comment un utilisateur peut-il s'assurer que ses styles personnels sont bien appliqués ?

Exercice 19 : En reprenant les exercices précédents, définissez les couleurs en utilisant la fonction `rgb()`.

Exercice 20 : Testez les différents styles des exercices précédents dans le validateur du W3C, en les écrivant dans l'élément `<style>` ou dans un fichier externe.