Université Sidi Mohamed Ben Abdallah de Fès
Faculté des Sciences Dhar El Mahraz Fès

Langage CSS

Support de cours Version 3

Licences d'Etudes Fondamentales SMI (Semestre 3)

2017 – 2018

Prof. El Habib NFAOUI (elhabib.nfaoui@usmba.ac.ma)

Plan

1. Principe de la séparation du fond et de la forme

- Le W3C définit le langage CSS comme le suivant:

"Le langage CSS (Cascading Style Sheets - feuilles de style en cascade-) est le langage qui permet aux concepteurs de pages web de modifier le rendu d'un **document structuré**, tels les documents HTML. En séparant le contenu de sa mise en forme, CSS simplifie la création et la maintenance des documents web. C'est le **principe de la séparation du fond et de la forme**. "

Pour mettre en place ce principe pour nos pages web, nous disposons de deux langages distincts : le HTML et le CSS. Le HTML permet de décrire les données, c'est le fond. Il n'a pas été créé pour définir le rendu visuel des documents web, mais pour définir la structure du document et organiser le contenu des informations qu'il contient. CSS modifie l'aspect visuel de ces données, c'est la forme. HTML et CSS sont donc indissociables (Figure 1 ci-dessous).

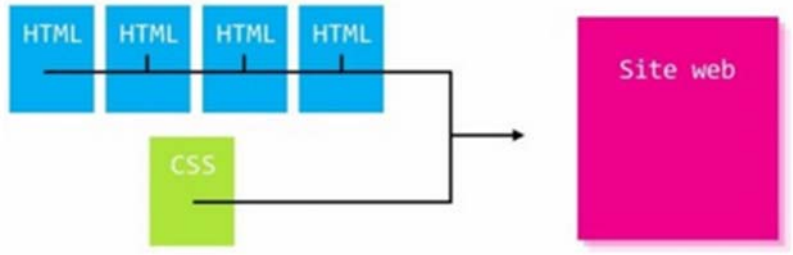

Figure 1. Principe de la séparation fond/forme

1. Principe de la séparation du fond et de la forme

- L'intérêt de la séparation entre la forme (CSS) et le contenu (HTML) est multiple, elle permet :
  - de pouvoir disposer d'une feuille de styles unique, applicable à des documents multiples ;
  - de pouvoir disposer de feuilles de styles alternatives, en fonction du choix des concepteurs ou des utilisateurs ou en fonction des médias ;
  - d'avoir une meilleure maintenabilité d'un site web (les mises à jour sont plus simples et plus rapides);
  - d'avoir une plus grande flexibilité (une seule modification du CSS modifie l'ensemble des pages d'un site web) ;
  - de concevoir le document en prêtant attention à la sémantique des éléments.
- CSS permet également l'accessibilité des données en supportant les feuilles de styles spécifiques par média. En effet, les médias alternatifs ne doivent pas être oubliés car la présentation d'une page HTML via CSS ne se limite pas à sa version écran classique. Plusieurs types de médias sont connectés au web et donc potentiellement clients de nos pages HTML. Cela comprend les terminaux mobiles bien sûr (téléphones et tablettes), mais également les téléviseurs, les médias embarqués dans l'automobile ou les lecteurs d'écrans qui permettent par exemple aux internautes malvoyants d'accéder à l'information. C'est la base de l'accessibilité du web. CSS peut nous aider à rendre notre contenu accessible pour tout et pour tous.
- CSS est un standard web en constante évolution. Il définit un ensemble de règles de mise en forme appliquées de façon combinée selon un degré de priorité (styles intrinsèques au navigateur, styles utilisateur, styles selon le média de consultation). Aujourd'hui, on en est à la **version 3** de ce langage (notée CSS3). Dans ce support de cours, j'ai donc préféré de présenter les notions de base et les progrès sans indiquer à chaque fois le numéro de version.

2. Les éléments de base du langage CSS

2.1 Principe de CSS
2.2 Syntaxe de la déclaration d'un style CSS
2.3 Propriétés
2.4 Écrire des feuilles de style

2.1 Principe de CSS

- Les feuilles de styles CSS sont composées de blocs définissant un ensemble de déclarations à appliquer à une ou plusieurs balises HTML dans le but de définir la présentation de la page web. Ces balises sont désignées par le **sélecteur** CSS précédant le bloc.

- Au chargement de la page HTML, le navigateur construit **un modèle arborescent** correspondant à l'ensemble des balises de la page web appelé le DOM (Document Object Model). Il charge ensuite la (ou les) feuille(s) de styles puis applique les différentes règles aux objets du DOM au fur et à mesure de la lecture des styles.

2.2 Syntaxe de la déclaration d'un style CSS

- Une déclaration de style comporte plusieurs parties, selon l'ordre suivant:
  - Un sélecteur qui détermine à quel **élément** et éventuellement dans quelles conditions s'applique le style.
  - Les déclarations des propriétés que l'on veut voir appliquées à l'élément sélectionné. Elles doivent être incluses entre des accolades ouvrante ({) et fermante (}), déterminées chacune par un mot-clé propre à CSS suivi du caractère deux-points (:), puis de la valeur attribuée à cette propriété. Chaque déclaration doit être séparée de la précédente par le caractère point-virgule (;).

```
sélecteur {
        propriété1:valeur1;
        propriété2:Valeur2;
        ...
}
```

Autant que les propriétés, c'est la variété des sélecteurs qui fait la richesse de CSS.

Afin d'améliorer la lisibilité et la maintenance du document, il peut être utile, voire même souhaitable de commenter le code CSS à l'intérieur des balises d'insertion de définition de style. La commande **/* Commentaire */** permet l'insertion de commentaire au sein du code CSS.

Exemple

- Dans l'exemple ci-dessous, ***h1*** est le sélecteur, ***color*** est la première propriété qui détermine la couleur du texte de l'élément, **red** est la valeur attribuée à cette couleur, ***background-color*** est la seconde propriété qui représente la couleur de fond et **yellow** sa valeur. Tous les éléments ***<h1>*** de la page dans laquelle se trouve cette déclaration ont donc un contenu écrit en rouge sur fond jaune.

```
h1 {
  color : red ;
  background-color : yellow ;
}
```

2.3 Propriétés

- CSS définit plusieurs propriétés permettant de définir la présentation des pages Web. On trouve des propriétés pour la mise en forme du texte, l'arrière plan, des marges, des listes, le graphisme, le positionnement, les transitions, l'animation, etc. Le tableau 1 et le tableau 2 présentent respectivement quelques propriétés pour le texte et les fonds. Pour plus de détails, le lecteur pourra se référer à la documentation complète du CSS sur le site Web du W3C.

Tableau 1: Quelques propriétés pour le texte

Propriété	Rôle
color	Couleur du texte
font	(Déclaration groupée)
font-weight	Graisse
font-size	Taille de police
font-family	Famille de police
font-variant	Variante
font-stretch	Étirement du texte
text-decoration	Décoration de texte

2.3 Propriétés

Tableau 2: Quelques propriétés pour les fonds

Propriété	Rôle
background	(Déclaration groupée)
background-color	Couleur de fond
background-image	Image de fond
background-repeat	Répétition de l'image de fond
background-position	Position de l'image de fond
background-attachment	Attache de l'image de fond par rapport à la page
background-origin	Position du fond par rapport à la boîte de l'élément
background-size	Taille de l'image de fond par rapport aux dimensions de l'élément

2.4 Écrire des feuilles de style

- Dans cette section, nous allons étudier les différentes méthodes d'écriture des styles CSS et la façon dont on peut les lier à un document HTML. En fait, Les styles peuvent êtres écrits dans une **balise style en en-tête** de la page HTML, dans un **fichier séparé** qui sera lié à la page, ou bien dans **l'attribut style** d'une balise HTML.

a. Insérer des styles dans l'en-tête du document

- On peut d'abord placer des styles CSS à l'intérieur de la balise ***<style>*** dans l'en-tête HTML (contenu de l'élément <head>). La balise *<style>* a pour vocation de renfermer les définitions des styles CSS utilisables **dans la page qui le contient (on l'appelle aussi style de document)**. Il doit toujours être inclus dans l'élément *<head>* et il ne peut contenir que des définitions de styles CSS et des commentaires HTML délimités par <!-- et --> ou des commentaires CSS délimités par /* et */ .

```
<head>
    <style type="text/css">
            body{
                color: navy;
            }
    </style>
</head>
```

Cette technique sépare correctement le contenu et la mise en forme. Cependant la portée du style défini se limite au document HTML du fichier. Dans l'idéal, le design général du site s'appliquera automatiquement, sans devoir être explicité dans chaque document HTML. Pour aboutir à cet effet, nous placerons les règles CSS dans un fichier distinct (style externe).

b. Lier les styles à partir d'une feuille séparée (style externe)

- Afin de séparer le contenu et la présentation des pages web, l'écriture des styles dans les fichiers externes est fortement conseillée (recommandée). Le fichier ne doit contenir que des sélecteurs et les définitions des styles ainsi que des commentaires CSS (délimités par les caractères /* et */) mais aucune balise d'élément HTML. Le fichier CSS doit toujours être enregistré sous l'extension **.css** et être présent sur le serveur, tout comme les fichiers HTML qui l'utilisent. Les fichiers CSS font partie des ressources du site web ou bien l'application web tout comme les images, les vidéos, les chaînes de caractère, etc.

  Exemple:
  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;
}
```


b. Lier les styles à partir d'une feuille séparée (style externe)

- Pour lier le **fichier externe de styles à une page HTML on peut utiliser l'élément HTML <link>** ou la **règle CSS @import**.

  - **Liaison par la balise <link>**

  Pour lier cette feuille de styles à toutes les pages HTML du site, il est d'usage de placer une balise <link> dans l'en-tête (<head>) de ces dernières.

  Exemple : Appel de la feuille de style commun.css située dans le dossier « chemin _ressource »

```
<head>
        <link rel="stylesheet" type="text/css" href="chemin_ressource/commun.css" />
</head>
```

b. Lier les styles à partir d'une feuille séparée (style externe)

- **Utiliser la règle css @import**

  Cette règle permet elle aussi de lier une feuille de styles externe à son document HTML. Ce n'est pas une balise HTML, mais une règle CSS. Il est donc nécessaire de la déclarer dans l'élément <style> de l'en-tête du document :

Exemple : Appel de la feuille de style commun.css avec la règle @import depuis la page HTML

```
<head>
    <style type="text/css">
        @import url(chemin_ressource/commun.css);
    </style>
</head>
```


b. Lier les styles à partir d'une feuille séparée (style externe)

Méthode link ou @import?

Les méthodes **link** et **@import** permettent de gérer une feuille de styles externe qui est liée à plusieurs pages HTML, ce qui permet de limiter le téléchargement des styles par chacune des pages du site. Le résultat obtenu par les deux méthodes est en pratique identique, cependant on note quelques différences:

- <link> est une balise HTML qui n'est pas uniquement dévolue aux feuilles de styles. Quand elle désigne une feuille de styles CSS, elle s'accompagne des attributs et valeurs suivantes : rel="stylesheet", type="text/css" et media=[type de média souhaité], voire title dans le cas de feuilles de styles persistantes et alternatives. Par exemple :

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

- La règle @import, propriété CSS, sera suivie de l'URL d'un fichier contenant les styles à appliquer en plus de la feuille de styles en cours. Par exemple :

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

  Cette propriété permet en outre **d'inclure des feuilles de styles dans d'autres**, ce qui permet de créer des feuilles de styles dynamiques sans devoir recopier plusieurs fois le même code.

c. Intégrer les styles dans l'attribut style d'une balise HTML

- Cette méthode d'inclusion des styles permet de modifier un seul élément de la page. C'est une méthode non optimisée et non préconisée.

```
<p style="text-decoration: underline;"> CSS est un langage de style…</p>
```

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

d. Ordre de priorité des styles

- Si une règle de style vient contredire une règle précédente, c'est en **général** le dernier style défini qui s'applique.
- 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;}
```

d. Ordre de priorité des styles

Styles liés et style 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>

e. 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 effectuée dans div est spécifique.

3. Sélecteurs

3.1 Introduction

- Les sélecteurs permettent de sélectionner les éléments HTML auxquels on veut attribuer un style donné. Ils sont la base de la recherche d'éléments HTML dans le DOM.

Rappelons la syntaxe d'un sélecteur CSS nommé sélecteurCSS permettant de définir la valeur valeurCSS pour la propriété des éléments HTML ciblés par ce sélecteur :

```
sélecteurCSS {
        propriétéCSS: valeurCSS;
}
```

3.2 Sélectionner un seul élément (Sélecteur d'élément)

- Le sélecteur d'élément permet de définir un style propre à un élément. Le sélecteur est constitué du nom de l'élément.

Exemple:

Le texte de tous les paragraphes (élément <p>) figure en jaune sur fond bleu.

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


3.3 Sélectionner plusieurs éléments (Sélecteur multiple)

- 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.

Exemple:

```
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.

Exemple:

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

3.4 Le sélecteur universel

- Pour appliquer un style à tous les éléments, nous utiliserons le sélecteur universel * avant la définition d'une ou de plusieurs propriétés.

Exemple:

Par exemple, pour que la couleur de fond de tous les éléments soit le jaune, nous écrirons :

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


3.5 Les classes de style

- Nous avons vu que tous les éléments HTML 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 (.).

Exemple:

Nous pouvons par exemple définir la classe nommée **menu_haut** en écrivant le code :

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

Pour appliquer cette classe par exemple à l'élément <h2>, nous écrivons:

```
<h2 class="menu_haut">Texte contenu du paragraphe</h2>
```

- 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;
}
```

3.5 Les classes de style

- **Appliquer plusieurs classes au même élément**

L'avantage de définir des classes de style est 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 HTML, 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.

3.6 Le sélecteur d'identifiant (id)

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

Exemple:

```
/* Code CSS */
#id_div1{
        color: white;
        background-color: blue;
}
div {
        color: black;
}

<!-- Code HTML 5 -->
        <div id="id_div1">Texte en blanc sur bleu</div>
        <div>Texte en noir </div>
```

3.7 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 ( ]).

  Exemple:

```
/* Style CSS */
abbr[title] {
        color: red;
        background-color: gray;
}
```

  Tous les éléments ***<abbr>*** qui possèdent un attribut ***title*** , quelle que soit sa valeur, ont un contenu affiché en rouge sur fond gris.

3.7 Les sélecteurs d'attributs

- 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.

  Exemple:

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

  Ce style sera appliqué sur tous les éléments ayant l'attribut ***title***.

- Il est possible de sélectionner plusieurs attributs pour un élément en faisant suivre son nom de plusieurs attributs entre crochets.

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

  Le style sera appliqué sur les titres ***<h2>*** ayant à la fois les attributs ***title*** et ***id***.

3.8 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 :

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

Exemple:

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

Le style est appliqué sur les éléments ***<code>*** ayant l'attribut ***title*** dont la valeur est la chaîne "code JavaScript".

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

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


3.8 Les sélecteurs de valeur d'attribut

Autres possibilités:

- élément [attribut **~=**"valeur"] { Définition des styles; } : Le style est attribué à l'élément dont l'attribut « attribut » a une valeur contenant exactement la chaîne « valeur » au sein de valeurs **séparées par des espaces**.

  Exemple 1:

  [attr~="kiwi"] : si l'attribut contient exactement « kiwi » au sein de valeurs séparées par des espaces.

  Exemple 2:

  ```
  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. 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 d'un espace et d'un nombre entier.

- élément [attribut ***=** "val"] { Définition des styles; } : si l'attribut contient la **sous-chaîne** « val » au sein de la chaîne contenant la valeur.

  Exemple:

  [attr***=**"kiwi"] : si l'attribut contient la sous-chaîne « kiwi » au sein de la chaîne contenant la valeur.

3.8 Les sélecteurs de valeur d'attribut

- élément[attribut **^=** "valeur"]{ définition du style } : Le style est attribué à l'élément dont la valeur de l'attribut commence par la chaîne « valeur ».

  Exemple:
  [attr^="kiwi"] : sélection si la valeur de l'attribut attr débute par la chaîne « kiwi ».

- élément [attribut **|=**"valeur"] { Définition des styles; } : Le style est attribué à l'élément dont la valeur de l'attribut débute par la chaîne « valeur » au sein de valeurs séparées par des traits-d'union. Autrement dit, il cible l'élément qui a: **attribut = valeur-** .

  Exemple 1:
  [attr|="kiwi"] : si la valeur de l'attribut attr débute par « kiwi » au sein de valeurs **séparées par des traits d'union**.

  Exemple 2:
  *[lang |= en] : cible pour l'attribut « **lang** » les valeurs « **en-US** », « **en-GB** », etc.

3.8 Les sélecteurs de valeur d'attribut

- element[attribut **$=**"valeur"]{définition du style} : si l'attribut finit par la chaîne valeur.

  Exemple
  [attr$="kiwi"] : si l'attribut « attr » finit par la chaîne « kiwi ».

3.9 Les sélecteurs contextuels parent-enfant

- 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 .

Syntaxe:

élément_parent élément_enfant {
définition des styles;
}

Exemple:

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

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

3.10 Les sélecteurs parent-enfant directs

- Le sélecteur parent-enfant direct permet d'appliquer un style à un élément à condition qu'il soit un **enfant direct** d'un autre élément et non plus un descendant indirect.

Syntaxe :

```
élément_parent > élément_enfant {
        Définitions des styles;
}
```

Exemple:

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

3.11 Les sélecteurs parent-enfant

- CSS offre d'autres possibilités de sélection d'un élément enfant en fonction de ses caractéristiques. Voici deux sélecteurs parmi plusieurs autres:

Syntaxe	Définition
élément:nth-child(n) { style }	le énième enfant de son parent (élément) en commençant par le début.
élément:nth-last-child(n) { style }	le énième enfant de son parent en commençant par la fin.

3.12 Les sélecteurs d'éléments adjacents

Syntaxe :

élément1 + élément2 {
    Définitions des styles;
}

Ce sélecteur permet de créer un style qui s'appliquera à l'élément de type 2 uniquement s'il suit immédiatement (**successeur direct**) un élément de type 1 dans le code HTML **sans y être inclus**.

4. Pseudo-classes et pseudo-éléments

4.1 Les pseudo-classes applicables aux liens
4.2 Les pseudo-classes dynamiques
4.3 Pseudo-classes utilisables essentiellement pour des éléments de formulaire
4.4 Les pseudo-éléments

4.1 Les pseudo-classes applicables aux liens

- Il y'a deux pseudo-classes spécifiques aux éléments qui 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

  Cette pseudo-classe 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

  Cette pseudo-classe permet d'attribuer un style à un lien qui pointe vers un document déjà visité, après un retour sur la page d'origine.

  Exemple:

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

4.2 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.
  - La pseudo-classe **: hover {...}** agît lorsque le curseur de la souris pointe l'élément.
  - La pseudo-classe **: active {...}** agît lors de l'activation de l'élément par un clic de souris notamment.
  - La pseudo-classe **:focus {...}** agît lors de la réception du focus par le clavier ou par la souris.

- Les pseudo-classes peuvent également être combinées afin que l'élément réagisse à une combinaison de types d'événement.

```
a:focus:hover {font-size: 12pt; color: red;}
```

Dans cet exemple, cette commande correspond à un lien qui reçoit le focus et qui est pointé par le curseur d'une souris.

4.3 Pseudo-classes utilisables essentiellement pour des éléments de formulaire

- La pseudo-classe **:enabled** affecte un style aux éléments <input> dont l'attribut *enabled* vaut la valeur enabled (valeur par défaut), c'est-à-dire qui sont accessibles à l'utilisateur (cases qui peuvent être cochées par exemple).

- La pseudo-classe **:disabled** affecte un style aux éléments <input> dont l'attribut disabled vaut la valeur disabled, donc qui ne sont pas accessibles à l'utilisateur.

- La pseudo-classe **:checked** affecte un style aux éléments <input> et particulièrement à ceux du type bouton *radio* ou case à cocher quand ils sont dans l'état coché. Notez que les styles appliqués sont dynamiques donc réversibles et disparaissent quand on décoche une case par exemple.

4.4 Les pseudo-éléments

- Ils permettent d'agir sur une partie du contenu d'un élément. Ils s'écrivent à l'aide d'un double deux-points (::) depuis CSS 3.

  - **::first-letter** 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.

Exemple:

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

Dans cet exemple, la première lettre de chaque paragraphe sera trois fois plus grande que les autres et de couleur bleue.

  - **::first-line** 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 :

Exemple:

```
div:first-line {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.

4.4 Les pseudo-éléments

  - **::before** 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.

Exemple:

```
cite:before {content:"<<"; font-weight: bold;}
```

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

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

Exemple:

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

Dans cet exemple, chaque citation contenue dans l'élément <cite> sera suivie des caractères >> en gras.

5. Héritage des propriétés de style

- Certaines des propriétés définies dans les feuilles de style sont héritées, c'est-à-dire qu'elles sont transmises aux éléments imbriqués; d'autres ne peuvent pas l'être.

  Si une propriété héritée est appliquée à un élément, elle s'appliquera en même temps à tous les éléments que celui-ci contient (éléments enfants).

  Exemple:

  Nous pouvons par exemple définir les styles suivants:

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

  Si dans le code HTML5 de la page figurent les éléments suivants:

  ```
  <div>Texte<p>Premier paragraphe <span class="">HTML 5</span> et les styles
  <strong>CSS</strong>sont indispensables </p> à tous !</div>
  ```

  Dans cet exemple, l'élément <p> est enfant de et les éléments et sont eux-mêmes enfants de <p>. Par héritage, ceux-ci ont un contenu qui possède les styles définis pour leur parent direct ou indirect . 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 .

5. Héritage des propriétés de style

- L'héritage concerne un grand nombre de propriétés CSS, 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. 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. Cela crée un héritage pour des propriétés qui normalement ne sont pas héritées.

6. Unités de tailles et couleurs

6.1 Unités de tailles

6.2 Codage des couleurs

6.1 Les unités de taille

- Toutes les propriétés CSS peuvent prendre une valeur dans un domaine particulier propre à chacune d'elle. Il est important de définir les codes et unités à adopter pour les valeurs qui leur seront attribuées.

- **Unités de taille**

Les unités de taille sont souvent utilisées pour les propriétés de bordures, dimensions, marges extérieures et intérieures, etc. Elles peuvent être fixes, définies par une longueur, ou relatives à l'affichage utilisé. Elles s'expriment par un nombre entier ou décimal selon les cas, suivi d'une unité. On dénombre les unités suivantes:

- **Unités de taille relatives :**

Les unités de taille relatives sont fonction de la taille du texte environnant ou du nombre de points sur l'écran.

– **em :** largeur d'une majuscule comme M.

– **ex** : correspond à la taille de la lettre « x » minuscule dans la police utilisée.

– **px** : 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 pourcentages**

Les pourcentages ne sont pas des unités mais une convention d'écriture. Le symbole **%** représente 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.

6.1 Les unités de taille

- **Les unités absolues**

Elles sont recommandées quand les caractéristiques physiques (mesurables) du média sont connues.

- **in** : soit un pouce anglais (un inch), 1 inch ou 1 pouce = 2,54 cm.
- **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 unités les plus pertinentes pour un affichage à l'écran restent **px**, **%**, et **em**.

6.2 Codage des couleurs

- Pour créer une couleur sur un écran, le mode colorimétrique utilisé est le RGB (Red - Green - Blue) ou RVB en français (Rouge - Vert - Bleu). Il y'a un octet de la mémoire pour le rouge, un octet pour le vert et un pour le bleu. Cela signifie que la quantité de rouge peut aller de 0, c'est-à-dire pas de rouge, à 255 (ou FF en hexadécimal), pour le maximum de rouge. Au total, cela donne plus de 16 millions de possibilités.

- Le code RGB (Red - Green - Blue) consiste à fournir l'intensité de chacune de ces trois couleurs dans l'ordre, de trois façons possibles :
  - soit en hexadécimal, chaque composante étant exprimée sur deux chiffres, compris entre **00** et **ff**;

  Exemple :

  #0000ff = bleu (deux chiffres hexadécimaux pour chaque couleur RGB)

  Notation hexadécimale raccourcie

  Il est possible d'utiliser une notation hexadécimale raccourcie, dans laquelle chaque chiffre hexadécimal doit être doublé pour obtenir le code réel de la couleur. Par exemple, #00f est équivalent à #0000ff et représente le bleu.

6.2 Codage des couleurs

- soit en décimal, l'intensité de chaque couleur étant codée à l'aide de trois chiffres, compris entre **000** et **255**, avec la fonction **rgb(xx,xx,xx);**

  Exemple:
  rgb(255,0,0) =rouge

- soit encore en pourcentage, puisque dans l'expression **rgb(xx,xx,xx)**, le code xx de chaque couleur peut être aussi un pourcentage compris entre **0%** et **100%**.

  Exemple:
  rgb(0,100%,0) =vert

Remarque:
Il existe d'autres options et codes pour gérer les couleurs: rgba(), hsl(), hsla().

- Noms de couleurs

À certaines couleurs « standard » ont été attribués des mots réservés : *blue*, *white*, *red*...La liste de ces mots-clés est donnée sur le tableau 3 ci-dessous.

Tableau 3: Les 16 couleurs de base du HTML

Nom en français	Nom HTML	Code hexadécimal	Code décimal
Blanc	white	#ffffff	rgb(255,255,255)
Bleu	blue	#0000ff	rgb(000,000,255)
Bleu foncé	navy	#000080	rgb(000,000,128)
Bleu-vert	teal	#008080	rgb(000,128,128)
Cyan	aqua	#00ffff	rgb(000,255,255)
Gris clair	silver	#c0c0c0	rgb(192,192,192)
Gris foncé	gray	#808080	rgb(128,128,128)
Jaune	yellow	#ffff00	rgb(255,255,000)
Marron	maroon	#800000	rgb(128,000,000)
Noir	black	#000000	rgb(000,000,000)
Rose	fuchsia	#ff00ff	rgb(255,000,255)
Rouge	red	#ff0000	rgb(255,000,000)
Vert	green	#008000	rgb(000,128,000)
Vert brillant	lime	#00ff00	rgb(000,255,000)
Vert olive	olive	#808000	rgb(128,128,000)
Violet	purple	#800080	rgb(128,000,128)

7. Requêtes de média (Media queries)

7.1 Introduction

- La conception multi-écrans est devenue une réalité pour les **concepteurs Web d'aujourd'hui**. L'accès au Web, autrefois réservé aux seuls ordinateurs de bureau, s'étend aujourd'hui aux mobiles, aux consoles de jeu portables, aux télévisions, etc. **La consultation multi-écrans nécessite une conception multi-écrans**.

- Chaque support de consultation possède des caractéristiques physiques, comme sa largeur et sa hauteur, des caractéristiques techniques comme sa résolution ou son format d'écran, qui lui sont propres.

- La spécification CSS **Media Queries*** (Les **requêtes de média**) définit les techniques pour l'application de feuilles de styles en fonction des périphériques de consultation utilisés pour du HTML. Les Media Queries (ou requêtes selon les médias) permettent de cibler des médias précis et de créer des styles spécifiques adaptés à chaque catégorie.

- L'ensemble des pratiques destinées à rendre un site Web adapté dynamiquement à chaque écran se désigne sous le vocable « **design adaptatif** » (responsive design en anglais). L'idée est simple : Prenons par exemple le cas des appareils mobiles. Plutôt que de développer une version mobile dédiée, avec de nouveaux squelettes HTML qui utilisent des feuilles de styles et des médias alternatifs, nous allons adapter (ajuster) notre design actuel avec du CSS, pour que celui-ci s'adapte à différentes plateformes. Ces bonnes pratiques permettent d'exploiter encore plus les avantages de la séparation du contenu et de la présentation : l'intérêt est de pouvoir satisfaire des contraintes de dimensions, de résolutions et d'autres critères variés pour améliorer l'apparence graphique et la lisibilité (voire l'utilisabilité) d'un site web.

*. http://www.w3.org/TR/css3-mediaqueries/

7.2 Syntaxe

- La philosophie des Media Queries (ou requêtes de media) en CSS est d'offrir un panel de critères plus vaste et plus précis, à l'aide de propriétés et de valeurs numériques, ainsi que de combinaisons multiples de ces mêmes critères. Le but est de cibler plus finement les périphériques de destination en fonction de leurs capacités intrinsèques.

- Les Media Queries sont des expressions booléennes, donc qui prennent une valeur true ou false et qui sont associées à des ensembles de styles spécifiques. Si la condition est réalisée, donc si l'expression a la valeur true, les styles qui suivent sont appliqués.

- Une requête de média se compose :
  - D'un **type** de média ciblé;
  - Des opérateurs logiques comme **and**, ou (symbolisé par une **virgule**), **only** ou **not**;
  - Des **conditions** écrites entre parenthèses et décrivant les caractéristiques physiques et techniques du support de visualisation ciblé.

- Pour préciser ces différents paramètres, trois méthodes sont possibles :
  - La requête est écrite dans la balise <link> grâce à l'attribut *media*.
  - La règle CSS @media.
  - La règle CSS @import.

7.2 Syntaxe

- **L'attribut *media* de la balise *<link>***

Avec l'élément <link/>, la condition est écrite dans l'attribut ***media*** sous la forme :

```
<link href="monstyle.css" media="type_de_média and ( condition )"
                          type="text/css" rel="stylesheet" />
```

L'utilisation de la balise <link> semble la plus naturelle pour contenir les requêtes de média. Avec cette méthode d'importation, le fichier CSS n'est chargé que si le type de média est celui utilisé par l'internaute, et si les conditions correspondent aux caractéristiques du support de visualisation. L'avantage de cette méthode est la répartition des règles CSS dans des fichiers uniques ciblant un média unique.

7.2 Syntaxe

- **La règle CSS *@media***

L'instruction CSS *@media* est caractérisée par le fait que les requêtes de média sont **intégrées parmi les autres règles CSS**. Cette méthode peut être recommandée lorsque les règles spécifiques à un média sont peu nombreuses, mais elle devient vite illisible lors de son utilisation avec un site Web d'importance.

```
@media type_de_média and (condition) {
   /* règles CSS */
}
```


7.2 Syntaxe

- **La règle CSS @import**

```
@import url("commun.css") type_de_média and (condition);
```

La méthode d'importation utilisant l'instruction @import nécessite au minimum deux fichiers CSS. Le fichier CSS appelé par la balise <link> va jouer le rôle d'aiguilleur, en répartissant l'importation des fichiers CSS selon le type de média et les conditions spécifiées.

Fichier HTML:

```
<link href="css/requetesMedia.css" rel="stylesheet" type="text/css">
```

Fichier **requetesMedia.css :**

```
/* écran large */
@import url("ecranLarge.css") only screen and (color) and (min-width : 1024px);
/* tablette et netbook */
@import url("tablette.css") only screen and (color) and (min-width : 480px) and (max-width : 1024px);
/* mobile intelligent */
@import url("mobileIntelligent.css") only screen and (color) and (max-width : 480px);
```

7.3 Les types de médias

Le type du média doit être donné par un des mots-clés du tableau suivant. Il peut être précédé de l'opérateur **only** (seulement celui-là) ou **not** (tout mais pas celui-là).

Type de média	**Description du média**
screen	Écran ordinateur, tablette, téléphone.
handheld	Petits écrans d'appareils portables. Attention, la plupart des mobiles actuels ne se considèrent pas comme média *handled* mais *screen*.
print	Support paginé et documents écran prêts à l'impression.
projection	Projection par page.
braille	appareil de diffusion tactile en braille.
embossed	Média à impression braille.
speech	Synthétiseurs de parole.
tty	Média utilisant une grille de caractères à chasse fixe.
tv	Média de type télévision.
all	Tous types de médias.

7.3 Les types de médias

Concernant les mobiles et les tablettes, le type de média ***handheld*** paraît le plus adapté, mais en réalité celui-ci est très peu pris en charge par les navigateurs mobiles. Au contraire, le type ***screen*** est plus générique et permet tout autant de cibler les mobiles que les tablettes ou encore les écrans d'ordinateurs de bureau.

7.4 Caractéristiques du support de visualisation

- Un ensemble de critères permettent de cibler très précisément un type de média grâce à ses caractéristiques physiques et techniques .
  - **Les caractéristiques physiques:**
    - width et height : largeur et hauteur de l'espace visualisable; la valeur est un chiffre entier, généralement exprimé en pixels dans le cas des mobiles et des tablettes.
    - device-width et device-height : largeur et hauteur du support de visualisation; la valeur est également un chiffre entier.
    - orientation : orientation du support de visualisation; les valeurs possibles sont les mots-clés *portrait* (vertical) ou *landscape* (horizontal).
    - aspect-ratio et device-aspect-ratio : format d'affichage de l'espace visualisable et support de visualisation; la valeur est une fraction qui peut être, soit la proportion de l'image, par exemple 16/9 ou 4/3, soit la résolution de l'écran, par exemple 1280/720 ou 1024/768.

7.4 Caractéristiques du support de visualisation

- **Les caractéristiques techniques**
  - color : support de visualisation utilisant un affichage en couleur; la valeur est un chiffre entier représentant le nombre de bits par couleur.
  - color-index : support de visualisation utilisant un affichage avec une table de couleurs indexées; la valeur est un chiffre entier représentant le nombre de couleurs contenues dans la table.
  - monochrome : support de visualisation utilisant un affichage en monochrome ou en niveaux de gris ; la valeur est un chiffre entier représentant le nombre de bits par pixel.
  - resolution : résolution du support de visualisation; la valeur est un chiffre entier exprimé, soit en points par pouce (dpi), soit en points par centimètre (dpcm).
  - scan : méthode de balayage des écrans de télévision ; les valeurs possibles sont les mots-clés progressive (balayage pro- gressif) ou interlace (balayage entrelacé).
  - grid : support de visualisation utilisant affichage en grille ou matricielle (bitmap en anglais); la valeur est booléenne, 1 signifie que l'affichage est en grille, 0 signifie que l'affichage est matriciel.
  - webkit-min-device-pixel-ratio : densité de pixels; la valeur est un chiffre entier représentant la densité de pixels du support de visualisation, ce paramètre n'est reconnu que par les navigateurs mobiles utilisant le moteur de rendu HTML WebKit, dont Chrome sur Android et Safari sur iOS.
- **Les préfixes min et max**

  Toutes les conditions, à l'exception de *orientation*, *scan* et *grid*, acceptent les préfixes *min* et *max*. Ces derniers représentent respectivement la valeur minimale et maximale d'une condition, et sont indispensables pour améliorer le ciblage des supports de visualisation. Il est donc possible d'utiliser les conditions *min-width*, *max-device-width* ou encore *min-color*.

7.5 Exemples de requêtes de média

- **La taille de l'écran**

Pour cibler les écrans dont le navigateur a une largeur supérieure à 400 pixels, nous écrivons le code suivant :

```
@media screen and (min-width: 400px) {

    /* code du style à appliquer */
}
```

Pour retenir les écrans dont le navigateur a une largeur inférieure à 800 pixels, nous écrivons le code suivant :

```
@media screen and (max-width:800 px) {
   /* code du style à appliquer */
}
```

Pour cibler les écrans dont le navigateur a une largeur comprise entre 400 et 800 pixels nous écrivons le code suivant :

```
@media screen and (min-width: 400px) and (max-width: 800px) {
    /* code du style à appliquer */
}
```


7.5 Exemples de requêtes de média

- **L'orientation et les proportions de l'écran**

Les terminaux portables peuvent fonctionner en mode portrait ou paysage. Pour cibler les appareils dont l'écran est en mode paysage et afficher le texte d'un paragraphe sur deux colonnes nous avons le code suivant :

```
@media screen and (orientation:landscape) {
      p{
        -moz-column-count:2;
        -webkit-column-count:2;
        -ms-column-count:2;
        -o-column-count:2;
        column-count:2;
      }
}
```

Pour cibler un appareil dont l'écran a des dimensions dans une proportion supérieure à 4/3, par exemple plus large que 800 × 600, et afficher le texte d'un paragraphe sur deux colonnes, nous écrivons le code suivant :

```
@media screen and (min-aspect-ratio:4/3) {
      p{
        -moz-column-count:2; -webkit-column-count:2; -ms-column-count:2; -o-column-count:2;
        column-count:2;
      }
}
```

7.5 Exemples de requêtes de média

- **Le nombre de couleurs**

Pour cibler les appareils dont l'écran a un affichage de 1 bit par couleur, nous écrivons le code suivant :

```
@media screen and (min-color:1) {
   /* code du style à appliquer */
}
```

Pour s'adapter aux appareils dont l'écran utilise une table des couleurs comportant un nombre inférieur ou égal à 256 couleurs, nous écrivons le code suivant :

```
@media screen and (max-color-index:256) {
   /* code du style à appliquer */
}
```

Il est possible d'utiliser des conditions sans spécifier leurs valeurs. L'exemple suivant cible exclusivement tous les écrans couleurs.

```
@media only screen and (color) {
   /* code du style à appliquer */
}
```


7.5 Exemples de requêtes de média

Une condition n'accepte qu'une seule valeur. L'exemple ci-dessous cible les écrans avec une orientation horizontale, une largeur minimale d'espace visualisable de 320 pixels, affichant au minimum 2 couleurs, et dont le format d'affichage du support de visualisation est le 16/9 e .

```
@media only screen
      and (orientation : landscape)
      and (min-width : 320px)
      and (min-color : 2)
      and (device-aspect-ratio : 16/9)
```

Bibliographie

- CSS3 Le design web moderne. *Oliveira (de), Vincent*. Edition: Dunod, 2012. ISBN: 978-2-10-058566-3
- HTML5 et CSS3 : Cours et exercices corrigés. *Jean Engels*. Edition: Eyrolles, 2012. ISBN: 9782212134001
- Mémento CSS3. *Goetter,Raphaël*. Edition: Eyrolles, 2013. ISBN: 978-2-212-13665-4
- Premiers pas en CSS3 et HTML5. *Francis Draillard*. Edition: Eyrolles, 2013. ISBN: 9782212136890
- Tout sur HTML5 et CSS3. *Jean-Marie Cocheteau and Laurent Khouri*. Edition: Dunod, 2012. ISBN: 9782100578160