AVERTISSEMENT

Au départ, il y a une série de difficultés rencontrées lors de nos ateliers WEB au sein du club. Notre objectif est alors de faire une initiation au web interactif, avec Apache, PHP et MySql. Certains adhérents tiennent à utiliser leur portable ; d'autres les appareils du club, multiboot. Certains sont sous Windows (XP acheté il y a quelques années ; Vista) ou sous Linux (Ubuntu ; Fedora). En bref, multiplicité de systèmes et de navigateurs. En soi, la situation est intéressante.

Malheureusement, les ennuis commencent avant même l'installation d'un serveur Apache en boucle locale, avec les premiers essais de HTML, de feuilles de style et de scripts javascript.

- Firefox refuse obstinément de déchiffrer un modèle de site que nous avion élaboré au début de la décennie (2002) : foin des balises comme FONT et quelques autres. Et certains attributs sont manifestement devenus obsolètes et non reconnus.

- Firefox et Internet Exporer s'obstinent à ne pas interpréter de la même façon les balises de bloc (marging, padding).

- Le passage d'un fichier d'une machine Windows à une machine Linux (et réciproquement) ddonne des résultats bizarroïdes.

- Quant à un essai fait sur une page prise dans un progiciel professionnel par un adhérent informaticien -mais pas orienté web- (je ne citerai pas ce progiciel, vendu plusieurs milliers d'euros), nous nous apercevons que s'il est lisible, ce n'est qu'avec une version bien précise de IE, qu'il mélange allègrement JScript et javascript ; que quant aux feuilles de style, elles n'ont été essayées qu'avec le « bon IE » et donnent du n'importe quoi sur les autres navigateurs, en particulier sous Linux.

Et puis, nos adhérents ne s'avèrent pas nécessairement familiers de l'hexadécimal ou autres subtilités dans les systèmes de mesure du web. Alors ils cherchent sur le net, et là également, on trouve le meilleur et le pire. Heureusement il y a quelques sites sérieux dont il faut proclamer haut et fort les références : **`developper.com`**, **`fr.selfhtml.org`**, **`www.alsacreations.com`** Nous nous en tenons là, avec une mention particulière pour **`fr.selfhtml.org`** qui peut être téléchargé, et devient notre outil de référence, présent sur les appareils de travail. Tout y est : ce qui est obsolète est signalé ; les scripts instables sont « à prohiber »… mais la recherche n'y est parfois pas évidente.

Nous décidons alors de reprendre à zéro la programmation html, depuis le DOCTYPE, les feuilles de style et le minimum vital de javascript. Le document qui suit est le résultat de ce travail : notre volonté est de disposer d'une référence non exhaustive mais sûre, , ainsi que d'une méthode de travail, pour notre production en HTML dynamique. Nous pourrons alors passer de façon propre au web interactif, avec WAMP ou LAMP, sans avoir à se soucier du système d'exploitation, de son âge, de l'âge de la machine, ni du logiciel explorateur (Epiphany, Firefox, les multiples IE), de façon aussi à ce que la clef USB baladeuse autorise des échanges entre machines. (note : L'objectif ultime est de doter un centre d'accueil des sans-logis avec du matériel de récup ; tout ce qu'il lui faut pour fonctionner en économisant le temps des animateurs et en facilitant la communication interne et externe.)

Amiposte-Télécom Mont-Saint-Éloi, février 2009

1 le fichier HTML

fiche 1

objectif.

Il existe de nombreux tutoriel HTML, et pour beaucoup du type "*créez votre site en vingt minutes sans avoir aucune connaissances préalable*". Il vaut mieux les ignorer. Pour les autres, il en existe de trois sortes :

- ceux qui sont dépassés (car la programmation HTML évolue !),
- ceux qui sont "à jour"
- et ceux qui essaient de *concilier* le vieil HTML et la méthodologie d'aujourd'hui.

Malheureusement ceux qui sont à jour (il se disent souvent **web 2**) sont trop complets, trop exhaustifs, très techniques, et souvent rigides sur les méthodes qu'ils recommandent. Ils s'adressent franchement à des spécialistes.

Les tutoriels "*conciliateurs*" embrouillent plus qu'il n'éclairent car ils mélangent des méthodologies qui ne sont pas au même niveau. Par exemple, ils gardent la mise en page par tableau qu'il vaut mieux ne plus utiliser, et ils la complètent par des feuilles de style de placement : ce type de feuille de style doit être l'unique moyen pour faire une mise en page robuste, mais facile à mettre au point.

> L'idée principale des fiches qui suivent est de développer un tutoriel le plus **simple** possible, voire simpliste, volontairement **non exhaustif** ; mais ce texte prend résolument **le parti de la programmation web "actuelle"**. Ce point de vue ne satisfera pas les puristes fignoleurs. En effet certains choix sont fortement simplificateurs, par exemple dans le traitement des tableaux (réduits à l'utilisation de 3 balises). Ou non conforme aux canons actuels, comme par exemple l'utilisation de balises "d'aspect" (qu'en général on ne redéfinit pas, et qui sont à considérer comme des abréviations) comme la balise **<b>** -gras- ou **<i>** -italique- au lieu des balises "sémantiques" **** *-renforcement expressif fort sur le texte-* ou **** *-renforcement faible-* . L'avantage d'utiliser les balises **** ou **** est qu'on peut (qu'on doit) les redéfinir avec des feuilles de style, pour les accorder au sens d'un contexte particulier ; la maîtrise d'œuvre est ainsi complète.

Ce parti pris a au moins deux avantages : il ne charge pas l'apprentissage de méthodes dépassées ni de méthodes peu utilisées. Des méthodes plus fines pourront être acquises une fois les lignes de force de la programmation HTML bien connues.

1. Le fichier HTML.

Un fichier HTML est un texte sauvegardé avec l'extension **.html** (ou **.htm**). Il contient les renseignements permettant à un navigateur (exemple : Firefox, Internet Explorer …) de **l'interpréter** pour afficher une "page" à l'écran, c'est à dire un message graphique (texte et images) selon des normes de présentation (titres, sous-titres, choix de caractères, tableaux…) qui sont choisies ultérieurement par le programmeur ou l'infographiste.

Pour que l'interprétation soit correctement menée, le fichier est doté d'une structure bien définie, en principe intangible ; il faut savoir, cependant, que l'absence d'une commande conduit souvent le navigateur à tenter d'utiliser une commande par défaut. Cette commande par défaut dépend du navigateur choisi, de sa version, de son paramétrage. Dans une telle situation, on n'est jamais sûr du résultat. Par exemple, si on tape un texte à afficher sans aucun renseignement d'affichage, il est affiché, avec des caractéristiques (fonte, dimensions de caractère, largeur de ligne) choisies par le navigateur qu'on utilise et peuvent de plus dépendre de son paramétrage.

2. Les balises.

Le langage HTML a une grammaire. C'est un langage de balise. Une balise commence toujours par le signe '**<**', **immédiatement** suivi du nom de la balise écrit en minuscule. Les noms de balises sont

standardisés. Il est impossible de créer de balises personnelles en HTML.

Après le nom de balise on trouve éventuellement des renseignements appelés **attributs**. La balise est obligatoirement terminée par le signe '**>**'.

2.1. Séparateurs.

* deux balises peuvent se suivre sans qu'il y ait de caractère pour les séparer. Exemple : ** **

* à l'intérieur d'une balise, le séparateur universel est l'espace. S'il y a des attributs, un espace sépare le nom de la balise et les attributs ; dans le cas où il y a plusieurs attributs, on les sépare également par des espaces. Les espaces inutiles ne gênent pas car ils ne sont pas pris en compte ; on peut même les remplacer par un ou plusieurs sauts de lignes.

* par contre une erreur classique est de séparer le signe '**<**' du nom de la balise ; la balise est alors ignorée. Quand le slash (**/**) est nécessaire, c'est toujours après '**<**' ou avant '**>**'. Le '**/**' ne doit pas être séparé du signe '**>**' ou '**<**'**.** Si le nom de balise suit le '**/**', il ne doit pas non plus y avoir de séparation.

Exemples :

correct : ** **

pas correct : **< span id="xyz"> < /span> <img / > </ p>**

* dans le texte HTML, un saut de ligne est interprété comme un espace. Partout où un espace est légal, on peut aller à la ligne : on ne fait que rajouter un espace. Si on veux coller 2 balises, il ne faut donc pas qu'elles soit séparées par un saut de ligne ! Il y aurait un espace en trop et non voulu.

Exemples:

**** : deux images collées

```
<img src="mon_image.jpg" />
<img src="ton_image.jpg" />
```
: deux images non collées, à cause du changement de ligne

```
<img src="mon_image.jpg"
 /><img src="ton_image.jpg" />
```
: deux images collées, même résultat que le premier exemple

* une suite d'espace(s) et/ou passage(s) de ligne est équivalent à un espace unique. Ce n'est pas la peine de multiplier les espaces dans un texte à afficher : un seul espace est affiché ! Par contre, les espaces multiples permettent l'indentation du texte.

note : il y a une exception avec la balise **pre** qui sera étudiée en son temps.

2.2. les types de balises.

Il existe **deux types principaux de balises** et un type **exceptionnel** : les balises simples, les balises normales (qui sont appariées) et le type d'exception qui au départ servait pour les commentaires.

Note : le mot balise est un peu ambigu, puisqu'il sert à désigner un élément générique (*la balise p*) et le symbole graphique effectif du texte HTML (les balises ** ** et ** **). Ceci est sans grande conséquence pour la compréhension de ce qui suit.

2.2.1. les balises simples.

Les balises "simples" se suffisent à elles-mêmes ; leur interprétation ne dépend pas de ce qui suit ou ce qui précède. Pour marquer qu'une balise (symbole graphique) est simple, on la ferme ! Cela consiste à mettre un slash **immédiatement** avant le signe **>**.

Exemples de balises simples :

```
<meta content="text/html; charset=ISO-8859-1" http-equiv="content-type" />
<br />
<br/>
<link rel="stylesheet" type="text/css" href="rescss/global.css" />
<img id="illustrationB" src="resimg/mairie/mairie.jpg"/>
```

Une bonne habitude est de séparer le couple '**/>**' de ce qui précède.

2.2.2. les balises normales vont par paires.

L**es balises "normales"** vont par paires. Exemple : la balise **p** est une balise normale :

```
<p>bla bla bla</p>
```

A une balise **ouvrante** correspond une balise **fermante**. On écrit la balise fermante appariée à une balise ouvrante en rajoutant un slash **après le signe '<'**.

Il est prudent de mettre un "contenu", c'est-à-dire du code entre la balise ouvrante et la balise fermante ; quitte à mettre un code "bidon", un espace, ou mieux, un espace insécable (dans le code ISO, caractère numéro 160). Il y a quelques cas rares où le contenu de balise peut être vide sans inconvénient ou même doit l'être (avec la balise **div** en particulier) ; mais dans un tableau, une cellule vide peut conduire à un affichage imprévu. Dans les exemples ci-dessous, seul le contenu "espace insécable" est montré. Pour les autres exemples pour signifier un contenu quelconque, on a mis des points de suspension.

```
<p> </p>
<p class="barre"> </p>
<div style="clear:left;"></div>
<b>...</b>
<div>...</div>
```

Règle impérative : les balises respectent strictement la règle de l'imbrication. On dit couramment que les balises sont en "poupées russes" : les unités définies par deux couples de balises ne peuvent se chevaucher ; soit elles n'ont rien en commun, soit l'une d'elles est incluse dans l'autre.

```
<p>voici un <b><i>bon</i></b> exemple d'imbrication</p>
```

Pour s'assurer de la bonne imbrication, il est conseillé d'indenter le code.

Exemple:

```
<p>
  texte normal
  <b>
    texte écrit en gras
    <i>
      texte en gras et en italique
    </i>
  </b>
  suite du texte normal
</p>
```

2.2.3. les balises exceptionnelles.

Il y a actuellement deux balises exceptionnelles en HTML. Elles commencent par **<!** et sont d'un usage très spécifique :

§ **la déclaration de type de document** : ce n'est pas à proprement parler une balise HTML, puisqu'elle se situe avant le texte HTML. Elle renseigne le navigateur sur la version de la grammaire HTML qu'il doit utiliser. Son contenu est standard. Comme ce tutoriel se consacre au HTML actuel (pas au vieil HTML, ni au XHTML du futur) la balise doit être :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
```

attention : bien respecter les espaces. Par contre comme le HTML ne fait pas la différence entre les espaces et le sauts de ligne on peut écrire la balise sur une seule ligne ou 2, 3, 4, 5 ou 6 lignes …

```
<!DOCTYPE
      html
      PUBLIC "-//W3C//DTD HTML 4.01//EN"
      "http://www.w3.org/TR/html4/strict.dtd"
>
```

Par contre, les arguments entre quotes (guillemets) ne doivent pas être coupés.

§ **le commentaire :** il commence par **<!--** et se termine par **-->** (pas obligatoirement sur la même ligne).

On peut placer un commentaire partout où une balise est acceptée. Exemples :

```
<!-- ************************************************************ -->
<!--  ceci est un commentaire -->
```

Il n'est pas prévu de commentaire à l'intérieur d'une balise. Dommage ! Il est vrai que tout texte non interprétable dans un corps de balise est ignoré, mais il n'est pas d'usage d'utiliser cette faille pour glisser des commentaires.

Les commentaires ne sont pas imbricables ; mais si on veut "neutraliser" une séquence HTML pour des raisons de mise au point ou mémoriser une démarche, il suffit de la mettre dans un commentaire.

Exemples :

```
<!-- voici un exemple qu'il vaut mieux éviter :
 <td></td> car le contenu est absent, mais pas les surprises ! -->

<!-- ici on va placer le code d'un objet flash -->


<!-- <p>faut-il afficher ce texte ?</p> -->
```

3. Encodage et édition.

3.1. Comment écrire un fichier HTML ?

Comme un fichier 'HTML' est un fichier de texte, en principe n'importe quel éditeur convient ; mais les éditeurs dotés d'outils d'aide à l'édition conviennent mieux. Le bon vieux Notepad de Windows est à laisser aux oubliettes car il n'offre aucune liberté sur le système d'encodage (manière dont les caractères sont encodés). On lui préfère des éditeurs de type PSPad (gratuit, sous Windows) ou Quanta et même gedit (sous Linux). C'est un peu plus pénible à apprendre, mais on regagne rapidement le temps d'apprentissage.

On peut utiliser un éditeur wysiwyg du type Dreamweaver ou Komposer. Mais le temps gagné est illusoire. Le code HTML fourni de façon automatique par ces logiciels répond mal aux exigences d'une bonne structuration, et n'est pas celui qu'attend le programmeur. Seul Amaya, outil de normalisation édité par le w3c et l'INRIA présente toutes garanties ; il est orienté XHTML, peut être utilisé sous Windows comme sous Linux et il satisfait à la licence du logiciel libre. Mais il vaut mieux être aguerri pour l'utiliser.

3.2. Quel système d'encodage adopter ?

Un fichier HTML est constitué de caractères ; c'est un texte ! Il peut référencer des images à l'aide d'expressions textuelles réservées (les balises d'images). Mais il y a une certaine distance entre le **texte que l'on désire écrire,** le code effectif interne à l'éditeur et **le code qui est enregistré sur le disque** : cette distance, c'est **l'encodage**. Chaque caractère est enregistré selon un code binaire.

Au début de l'informatique grand public, on enregistrait les caractères sur 7 bits ; et en pratique, le huitième bit d'un octet restait disponible pour des usages particuliers. Avec ces 7 bits, on pouvait former 128 combinaisons différentes de 0 et de 1, depuis 0000000 jusqu'à 1111111. On disposait donc de 128 caractères, dont les 32 premiers étaient réservés ; ce code s'appelle l'ASCII (prononcer et même écrire : ASKI).

Les codes d'aujourd'hui constituent des extensions de l'ASKI. En gros, on trouve dans l'ASKI, les caractères d'une machine à écrire américaine : majuscules, minuscules, chiffre, ponctuation, parenthésages **( ), [ ], { },** signes mathématiques **+ / > <**, signes commerciaux **&** et **@,** plus quelques signe spéciaux **$ # ^ ** . Mais pas de caractères accentués, pas de guillemets à la « française », pas de guillemets à " l'anglaise "…

La norme ISO étend le codage de l'ASKI à 8 bits, **ce qui autorise 256 combinaisons différentes, donc 256 symboles ou caractères.** On peut alors se permettre de coder les caractères accentués

des langues ouest européennes. Mais c'est insuffisant pour le grec ou le cyrillique (russe). Il existe en fait une série de normes, 15 actuellement, qu'on choisit selon les besoins.

Exemples :

La Norme ISO 8859-1 concerne l'ouest de l'Europe sans le symbole €.
La Norme ISO 8859-15 qui intègre le caractère de l'Euro € et quelques autres.

La norme UNICODE étend encore le codage à un codage sinon universel, au moins assez large pour intégrer le cyrillique, l'arabe, l'hébreu, l'indi, le japonais courant. L'UNICODE est un code sur seize bits, sur deux octets, dit UNICODE UTF-16. Le code comporte 65536 combinaisons, donc 65536 symboles ou caractères sont possibles. **C'est le code de référence actuel (en fait, il existe deux normes UTF-16, dépendantes du système d'exploitation car Microsoft interprète les deux octets dans l'ordre inverse de celui qui est usuel)** ; l'UNICODE est le code utilisé dans le langage Java. Ce n'est pas lui qui est utilisé sur le web.

Le monde Unix a vulgarisé un codage appelé UTF-8 ; extension de l'UNICODE, l'UTF-8 est devenu le codage d'avenir pour le web et les bases de données. En UTF-8, certains caractères sont codés sur un octet (8 bits, dont le premier a la valeur 0). Ces caractères correspondent aux 128 caractères du code ASCII. D'autres sont codés sur 2 octets ayant les 3 premiers bits du premier octet égale à 110 et les 2 premiers bits de l'octet suivant égale à 10. Beaucoup de ces caractères correspondent à l'extension ISO 8859-x. D'autres enfin sont codés sur 3 ou 4 octets. Le nombre de '1' consécutifs, en tête du premier octet indique le nombre total d'octet utilisés. Les octets de suite commencent tous par '10'.

Le navigateur qui reçoit une page a afficher, doit donc connaître le type de codage utilisé lors de la création du fichier afin de restituer la bonne graphie dans le code et sur l'écran.

D'où la nécessité de pratiquer la double exigence suivante :

1. **Choisir un système d'encodage et s'y tenir** : le système d'encodage se paramètre bien sur les éditeurs cités ci-dessus (PSPad, Quanta). Linux utilise par défaut l'UTF-8.

2. **Déclarer dans le fichier HTML quel code on a utilisé** ; il y a une balise pour cela.

Exemple :

```
<meta content="text/html; charset=ISO-8859-1" http-equiv="content-type" />
ou
<meta content="text/html; charset=UTF-8" http-equiv="content-type" />
```

L'oubli d'une des deux règles conduit à des fichiers qui restent corrects du point de vue HTML (il n'utilise que l'ASKI) mais où les caractères affichés sont mal reconnus et donnent des textes illisibles dès qu'on utilise des caractères accentués ou étrangers.

3.3. troisième question : Est-ce aussi simple que cela ?

La réponse est non : d'abord parce que Microsoft a voulu imposer une variante de l'ISO, appelée windows 1250 pour nos pays occidentaux (ou norme ANSI). A proscrire absolument.

Ensuite parce que l'on peut avoir besoin de caractères étrangers (grecs, arabes, hébreux, japonais...), mathématiques ou autres exceptionnellement. Mais là, quel que soit le codage retenu, ce n'est pas simple ; soit parce que l'on ne sait pas rentrer un caractère au clavier portant sur deux octets, soit parce que la fonte d'affichage de l'éditeur (le fichier .TTF pour être simple) ne possède pas ce caractère.

Heureusement, on dispose pour le HTML, d'un recueil de caractères spéciaux, répertoriés, et qui s'écrivent comme une suite structurée de caractères ASKI. **Un code littéral spécifique HTML** est appelé **une entité HTML**.

Par exemple, l'**espace insécable** est répertorié par l'entité HTML **` `** Les entités HTML commencent toutes par un **&** et se terminent toutes par un point-virgule. C'est un peu fastidieux, mais sûr, au moins dans la limite de reconnaissance du navigateur qui sera utilisé (en général la famille Netscape/Mozilla a de bonnes performances en la matière ; pas de problème avec Firefox !).

Exemple avec des guillemets à l'anglaise :

```
&ldquo;  Cette chapelle a été démolie en 1791. &rdquo;
```

affiche : “ Cette chapelle a été démolie en 1791 ”

Les éditeurs du type PSPad ou Quanta peuvent être paramétrés de façon à coder correctement les entités ; par exemple on tape au clavier **é** et le code fourni est **é** C'est là une question de choix personnel. On peut même avoir un paramétrage qui affiche dans la fenêtre d'édition le caractère correspondant à l'entité.

Une table des entités usuelles se trouve en annexe.

3.4. Comment utiliser les caractères < > & " …dans un texte affichable ?

On vient de voir que les caractères **< > & "** ont un usage spécifique dans l'écriture d'un fichier HTML : ils sont réservés aux balises ou caractères spéciaux ; d'autre part la " double quote " figure dans les attributs de balises. Comment les écrire pour qu'ils ne soient pas interprétés comme caractères réservés (on dit parfois, *comment les échapper* ?). En HTML, on a déjà la solution : on les code par un symbole de caractère spécial, une entité HTML. C'est **<** pour inférieur, **>** pour supérieur, **&** pour l'ampersand & et **"** pour la double quote ".

Exemple : `&quot;le caractère & s'appelle ampersand&quot;`
affiche : **"le caractère & s'appelle ampersand"**

Autre exemple, le code suivant :

```
<input type="submit" value="faire un &quot;clic&quot; sur le bouton" />
```

affichera un bouton portant l'inscription suivante:

faire un "clic" sur le bouton

NOTE : en annexe on trouve les tableaux de caractères spéciaux (le code littéral vu ci-dessus ; on trouve aussi l**es codes numériques,** qu'il vaut mieux proscrire ; tous les caractères de code ISO compris entre 32 et 127 et entre 160 et 255 ont un code numérique HTML qui s'écrit avec **&#** suivi de la valeur décimale du code et terminé par un point-virgule. Ainsi @, de code 64 s'écrit avec l'entité numérique **@** il vaut mieux soit prendre l'entité littérale, soit le caractère proprement dit quand c'est possible. Cela marche bien avec les caractères accentués, moins bien avec les ligatures comme oe)

3.5. Majuscules ou minuscule ?

Le HTML est-il sensible à la casse ? La réponse théorique est : **non, mais**. Les incitations actuelles vont dans le sens : **tout en minuscules**. La raison est à chercher dans l'évolution du HTML vers une nouvelle norme (XHTML) qui exige que tout soit en minuscules. Il faut ajouter que le choix d'une attitude uniforme vis-à-vis de la casse conduit à des textes plus lisibles et plus faciles à relire, à modifier et à corriger.

Mais attention : Les entités sont sensibles à la casse !!!

4. Le squelette d'un fichier HTML.

Un fichier html a donc nécessairement la structure suivante :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
            "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <meta content="text/html; charset=ISO-8859-1"
          http-equiv="content-type" />
    . . . . . .
  </head>
  <body>
    . . . . . .
  </body>
</html>
```

La disposition indentée montre bien l'imbrication. Le fichier HTML proprement dit a une structure binaire. La partie de la balise **<head>** contient des renseignement généraux ; au minimum le système d'encodage. La partie de la balise **<body>** est le **corps** du fichier : il comporte tous les éléments affichés dans la fenêtre du navigateur dévolue à la page, ou tout au moins, les références qui permettent cet affichage (comme on le verra en abordant les formulaires, les feuilles de style et le javascript).

Annexe 1

La plus petite unité d'information est le bit. Il peut prendre la valeur 0 ou la valeur 1.
Les ordinateurs manipulent généralement les bits par groupes de 8, appelé octets ; un octet code ainsi de façon binaire un nombre qui peut prendre 256 valeurs, comprises entre 0 et 255. Pour une valeur plus grande, il faut utiliser plusieurs octets.

Pour représenter des caractères dans un fichier texte, on associe un nombre (code numérique) à une lettre, chiffre ou symbole. Par exemple, 65 pour A, 66 pour B, etc...Un des premiers standards crée en 1967 a été le code ASCII (American Standard Code for Information Interchange, code ASCII ou ASKI). Le code ASCII est un code à 7 bits : les codes de 0 à 127 comprennent les caractères non accentués, les nombres de 0 à 9 et certains caractères spéciaux comme indiqué ci-après.

de 0 à 31	*Les 32 premiers codes sont des caractères de contrôle*
de 32 à 63	**!"#$%&'()*+,-./0123456789:;<=>?**
de 64 à 95	**@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_**
de 96 à 127	**`abcdefghijklmnopqrstuvwxyz{|}~**

L'ASCII, bien que codé en fait sur 8 bits, ne définit pas de codes entre 128 et 255 et ne comporte pas les lettres accentuées nécessaires pour la plupart des langages ouest européens.

Comme l'ASCII est très limité, l'ISO (International Organization for Standardization) a crée des standards pour les codes de 128 à 255. Le plus utilisé est le standard ISO-8859-1, connu aussi sous le nom Latin 1. Il inclut les lettres accentuées nécessaires pour la plupart des langages ouest européens comme le français, l'allemand, l'espagnol, le norvégien. Cependant, il y a beaucoup d'autres langages qui ont besoin de codes spéciaux non inclus dans l'ISO-8859-1 comme le grec, le cyrillique.... L'ISO a donc crée une gamme de codes de ISO-8859-1, ISO-8859-2 etc. L'ISO-8859-15 contient l'euro €. L'espéranto est codé en ISO-8859-3 ou Latin 3. Tous ces codes sont des codes à un octet : les 128 premiers codes (de 0 à 127) sont identiques au code ASCII, et les 128 codes suivants (128 à 255) codent les caractères accentués ou spéciaux.

Certains constructeurs ont développé leur propre système de codage, non conformes aux standards internationaux (Ex : IBM EBCDIC, Windows-1252...)

Unicode

Le chinois, le japonais et le coréen ont de très nombreux caractères ; les langages scientifiques (math, chimie) aussi. Ces dialectes nécessitent au moins deux octets pour représenter tous leurs symboles. Avec deux octets, on dispose de 256*256 = 65536 codes. Cette technique a été reprise par le consortium Unicode (Xerox, Apple). L'avantage de l'UNICODE se résume par le slogan : "*un seul nombre pour chaque caractère, quelque soit la plate-forme, le pays, le programme ou le langage,*".

L'inconvénient : La représentation avec 2 octets de l'Unicode, dite UTF-16, double la taille des documents traditionnels européens ou américains. Un autre problème est plus sournois : Tous les constructeurs ne rangent pas les deux octets dans le même ordre en mémoire ; comme pour le oeufs à la coque, il y a les gros-boutistes et les petit-boutistes. C'est ennuyeux pour échanger des données. Il y a donc deux standards dits UTF-16 Little Endian (octet de poids faible puis octet de poids fort) et UTF-16 Big Endian (octet de poids fort puis octet de poids faible). Microsoft Word utilise UTF-16 Little Endian et c'est UTF-16 Big Endian qui est recommandé pour l'Internet.

Et naquit l'UTF-8

Pour résoudre le problème du doublement de la taille des fichiers et de l'incohérence des représentations, Ken Thompson, un des pères d'UNIX, a inventé en 1992 le codage UTF-8, une autre manière de coder les caractères Unicode. L'idée a été de coder les caractères les plus utilisés de 0 à 127 sur un octet, de coder les caractères de 128 à 1023 sur 2 octets et au-dessus de 1023 sur 3 octets (et ce jusqu'à 4 octets) et de retrouver simplement le code UNICODE à partir de celui en l'UTF-8. Le codage UTF-8 devenu un standard de l'Internet, donc de l'informatique, certainement un standard de l'avenir.

Le système de codage est simple. Le bit de poids fort des 128 caractères les plus utilisés (codé sur un seul octet) est toujours 0 ; c'est de l'ASKI ! Lorsque plusieurs octets sont nécessaires, on a la disposition suivante : les bits de poids forts du premier octet sont : 110 pour un codage sur 2 octets, 1110 pour 3 octets et 11110 pour 4 octets. Puis les octets de complément, ont 10 en poids forts.

Si le code UTF-8 d'un caractère est sur un octet, le code UNICODE, le code ASKI, le code ANSI et le code ISO sont identiques. Il existe par ailleurs un mode simple de passage du code UTF-8 au code UNICODE (voir plus loin).

Ces caractéristiques sont intéressantes à plusieurs titres.

* Un texte écrit en ASCII pur (code <128) reste inchangé. Le passage à l'UTF-8 est invisible pour les américains. En revanche un texte écrit en ISO-8859-1 est modifié pour les lettres accentuées représentées en UTF-8 sur 2 caractères. La taille d'un texte avec des accents est donc seulement augmentée de quelques pour cents.

* Le jeu de caractères est "sans-état" [stateless] : la signification d'un caractère ne dépend pas d'un marqueur dans une séquence (bits de fin de caractère par exemple). Un caractère manquant dans la séquence détruit la représentation du caractère mais pas plus, car on peut se re-synchroniser au caractère suivant ; même en cas d'accident, il ne peut y avoir empiètement du code d'un caractère sur le suivant, ce qui n'est pas le cas en UNICODE, où, par exemple, un saut d'octet condamne toute la suite du texte. C'est donc un code très robuste.

* La représentation d'un caractère ne peut pas être contenue dans la représentation d'une chaîne plus grande (un caractère commence toujours par 00, 01 –caractère ASCII pur-, 11 –caractères sur plusieurs octets, **autre chose que 10,** réservé au deuxième octet, éventuellement troisième ou quatrième octet d'un caractère), ce qui permet de faire des recherches de sous-chaînes par comparaison d'octets, comme sur les codages à un octet (avec cependant un problème qui peut exiger une norme supplémentaire ; on peut en effet avec des représentations longues -plus de 3 octets- avoir plusieurs codes pour une même représentation UNICODE ; c'est une faille de sécurité).

Ces caractéristiques en font donc le codage à préférer pour toutes les applications. Il a fallu beaucoup de temps pour que ce codage, nettement supérieur aux autres, s'impose. Par exemple, UTF-8 est devenu le codage natif à partir de la plate-forme .NET et de la version SQL Server 2005 chez Microsoft. Malheureusement il n'est que difficilement compatible avec le PHP, du moins lorsque PHP travaille sur les chaînes de caractères.

annexe 2

L'ASCII étendu. Les 32 premiers codes (caractères de contrôle), de 0 à 31, ne figurent pas ici

32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47

	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
48 **0**	49 **1**	50 **2**	51 **3**	52 **4**	53 **5**	54 **6**	55 **7**	56 **8**	57 **9**	58 **:**	59	60 **<**	61 **=**	62 **>**	63 **?**
64 **@**	65 **A**	66 **B**	67 **C**	68 **D**	69 **E**	70 **F**	71 **G**	72 **H**	73 **I**	74 **J**	75 **K**	76 **L**	77 **M**	78 **N**	79 **O**
80 **P**	81 **Q**	82 **R**	83 **S**	84 **T**	85 **U**	86 **V**	87 **W**	88 **X**	89 **Y**	90 **Z**	91 **[**	92 ****	93 **]**	94 **^**	95 **_**
96 **`**	97 **a**	98 **b**	99 **c**	100 **d**	101 **e**	102 **f**	103 **g**	104 **h**	105 **i**	106 **j**	107 **k**	108 **l**	109 **m**	110 **n**	111 **o**
112 **p**	113 **q**	114 **r**	115 **s**	116 **t**	117 **u**	118 **v**	119 **w**	120 **x**	121 **y**	122 **z**	123 **{**	124 **|**	125 **}**	126 **~**	127 □
128 **€**	129 □	130 **‚**	131 **ƒ**	132 **„**	133	134 **†**	135 **‡**	136 **ˆ**	137 **‰**	138 **Š**	139 **‹**	140 **Œ**	141 □	142 **Ž**	143 □
144 □	145 **‘**	146 **’**	147 **“**	148 **”**	149 **•**	150 **–**	151 **—**	152 **˜**	153 **™**	154 **š**	155 **›**	156 **œ**	157 □	158 **ž**	159 **Ÿ**
160	161 **¡**	162 **¢**	163 **£**	164 **¤**	165 **¥**	166 **¦**	167 **§**	168 **¨**	169 **©**	170 **ª**	171 **«**	172 **¬**	173	174 **®**	175 **¯**
176 **°**	177 **±**	178 **²**	179 **³**	180 **´**	181 **µ**	182 **¶**	183 **·**	184 **¸**	185 **¹**	186 **º**	187 **»**	188 **¼**	189 **½**	190 **¾**	191 **¿**
192 **À**	193 **Á**	194 **Â**	195 **Ã**	196 **Ä**	197 **Å**	198 **Æ**	199 **Ç**	200 **È**	201 **É**	202 **Ê**	203 **Ë**	204 **Ì**	205 **Í**	206 **Î**	207 **Ï**
208 **Ð**	209 **Ñ**	210 **Ò**	211 **Ó**	212 **Ô**	213 **Õ**	214 **Ö**	215 **×**	216 **Ø**	217 **Ù**	218 **Ú**	219 **Û**	220 **Ü**	221 **Ý**	222 **Þ**	223 **ß**
224 **à**	225 **á**	226 **â**	227 **ã**	228 **ä**	229 **å**	230 **æ**	231 **ç**	232 **è**	233 **é**	234 **ê**	235 **ë**	236 **ì**	237 **í**	238 **î**	239 **ï**
240 **ð**	241 **ñ**	242 **ò**	243 **ó**	244 **ô**	245 **õ**	246 **ö**	247 **÷**	248 **ø**	249 **ù**	250 **ú**	251 **û**	252 **ü**	253 **ý**	254 **þ**	255 **ÿ**

Les caractères entre 128 et 159 ne correspondent à aucun standard universel ; il sont dont à éviter. Tous les autres se traduisent aisément en ASKI, ISO, UNICODE,

annexe 3

Caractères spéciaux et entités HTML.

Toutes ces entités de caractères sont reconnues par Firefox et par Internet Explorer 7. Les caractères faisant problème sont très spécifiques (mathématiques et sciences) et ne sont pas repris. Les codes numériques (à éviter) sont dérivés de l'UNICODE.

Caractères généraux

caractère	code texte	code numérique	commentaire
'	&apos;	&#39;	apostrophe, quote simple
"	&quot;	&#34;	guillemets américains, double quote
&	&amp;	&#38;	et commercial, esperluette, ampersand
<	&lt;	&#60;	inférieur (less-than)
>	&gt;	&#62;	supérieur (greater-than)
€	&euro;	&#128;	euro (monnaie européenne)

			espace insécable
¡	&iexcl;	&#161;	point d'exclamation inversé
¢	&cent;	&#162;	cent (monnaie américaine)
£	&pound;	&#163;	pound (Livre sterling, monnaie anglaise)
¤	&curren;	&#164;	symbole monétaire
¥	&yen;	&#165;	yen (monnaie japonaise)
¦	&brvbar;	&#166;	barre verticale scindée
§	&sect;	&#167;	section
¨	&uml;	&#168;	tréma
©	&copy;	&#169;	copyright
ª	&ordf;	&#170;	indicateur ordinal féminin
«	&laquo;	&#171;	guillemet français ouvrant
¬	&not;	&#172;	crochet de négation
	&shy;	&#173;	tiret de césure optionnelle (permet au navigateur de couper le mot au bon endroit si besoin)
®	&reg;	&#174;	marque déposée
¯	&macr;	&#175;	macron
°	&deg;	&#176;	degré
±	&plusmn;	&#177;	plus-ou-moins
²	&sup2;	&#178;	exposant 2
³	&sup3;	&#179;	exposant 3
´	&acute;	&#180;	accent aigu
µ	&micro;	&#181;	mu (lettre grecque)
¶	&para;	&#182;	paragraphe
·	&middot;	&#183;	point médian
¸	&cedil;	&#184;	cédille
¹	&sup1;	&#185;	exposant 1
º	&ordm;	&#186;	indicateur ordinal masculin
»	&raquo;	&#187;	guillemet français fermant
¼	&frac14;	&#188;	un quart

½	½	½	un demi
¾	¾	¾	trois quarts
¿	¿	¿	point d'interrogation inversé

les caractères accentués sont en fin de page

×	×	×	multiplication
÷	÷	÷	division

attention le Œ et œ **sont à chercher dans les codes accentués et spéciaux.**

ˆ	ˆ	ˆ	accent circonflexe
˜	˜	˜	petit tilde
	‌	‌	antiliant sans chasse
‍	‍	‍	liant sans chasse
–	–	–	tiret demi-cadratin, tiret d'incise
—	—	—	tiret cadratin, tiret de dialogue
†	†	†	obèle
‡	‡	‡	double obèle
•	•	•	gros point médian
‾	‾	‾	tiret en chef (overline, spacing overscore)
™	™	™	marque commerciale (trade mark)
←	←	←	flèche vers la gauche
↑	↑	↑	flèche vers le haut
→	→	→	flèche vers la droite
↓	↓	↓	flèche vers le bas
↔	↔	↔	flèche bilatérale gauche-droite
◊	◊	◊	losange
♠	♠	♠	pique noir
♣	♣	♣	trèfle noir
♥	♥	♥	cœur noir
♦	♦	♦	carreau noir

Monnaies

caractère	code texte	code numérique	commentaire
¤	¤	¤	symbole monétaire
€	€	€	euro (monnaie européenne)
¢	¢	¢	cent (monnaie américaine)
£	£	£	pound (Livre sterling, monnaie anglaise)
¥	¥	¥	yen (monnaie japonaise)
ƒ	ƒ	ƒ	florin (idem fonction)

Caractères alphabétiques accentués et spéciaux

caractère	code texte	code numérique	commentaire
À	À	À	
Á	Á	Á	
Â	Â	Â	
Ã	Ã	Ã	
Ä	Ä	Ä	
Å	Å	Å	
Æ	Æ	Æ	
Ç	Ç	Ç	
È	È	È	
É	É	É	
Ê	Ê	Ê	
Ë	Ë	Ë	
Ì	Ì	Ì	
Í	Í	Í	
Î	Î	Î	
Ï	Ï	Ï	
Ð	Ð	Ð	lettre majuscule islandaise ED
Ñ	Ñ	Ñ	
Ò	Ò	Ò	
Ó	Ó	Ó	
Ô	Ô	Ô	
Õ	Õ	Õ	
Ö	Ö	Ö	
Ø	Ø	Ø	
Ù	Ù	Ù	
Ú	Ú	Ú	
Û	Û	Û	
Ü	Ü	Ü	
Ý	Ý	Ý	
Þ	Þ	Þ	lettre majuscule islandaise Thorn
ß	ß	ß	lettre minuscule allemande s dur

Caractères accentués minuscules et ligatures

à	à	à	
á	á	á	

â	&acirc;	&#226;	
ã	&atilde;	&#227;	
ä	&auml;	&#228;	
å	&aring;	&#229;	
æ	&aelig;	&#230;	
ç	&ccedil;	&#231;	
è	&egrave;	&#232;	
é	&eacute;	&#233;	
ê	&ecirc;	&#234;	
ë	&euml;	&#235;	
ì	&igrave;	&#236;	
í	&iacute;	&#237;	
î	&icirc;	&#238;	
ï	&iuml;	&#239;	
ð	&eth;	&#240;	lettre minuscule islandaise ed
ñ	&ntilde;	&#241;	
ò	&ograve;	&#242;	
ó	&oacute;	&#243;	
ô	&ocirc;	&#244;	
õ	&otilde;	&#245;	
ö	&ouml;	&#246;	
ø	&oslash;	&#248;	
ù	&ugrave;	&#249;	
ú	&uacute;	&#250;	
û	&ucirc;	&#251;	
ü	&uuml;	&#252;	
ý	&yacute;	&#253;	
þ	&thorn;	&#254;	lettre minuscule islandaise thorn
ÿ	&yuml;	&#255;	

Œ	&OElig;	&#338;	ligature majuscule latine OE
œ	&oelig;	&#339;	ligature minuscule latine oe
Š	&Scaron;	&#352;	lettre majuscule latine S avec caron
š	&scaron;	&#353;	lettre minuscule latine s avec caron
Ÿ	&Yuml;	&#376;	

Alphabet Grec

caractère	code texte	code numérique	commentaire

Α	Α	Α	Alpha
Β	Β	Β	Beta
Γ	Γ	Γ	Gamma
Δ	Δ	Δ	Delta
Ε	Ε	Ε	Epsilon
Ζ	Ζ	Ζ	Zeta
Η	Η	Η	Eta
Θ	Θ	Θ	Theta
Ι	Ι	Ι	Iota
Κ	Κ	Κ	Kappa
Λ	Λ	Λ	Lambda
Μ	Μ	Μ	Mu
Ν	Ν	Ν	Nu
Ξ	Ξ	Ξ	Xi
Ο	Ο	Ο	Omicron
Π	Π	Π	Pi
Ρ	Ρ	Ρ	Rho
Σ	Σ	Σ	Sigma
Τ	Τ	Τ	Tau
Υ	Υ	Υ	Upsilon
Φ	Φ	Φ	Phi
Χ	Χ	Χ	Chi
Ψ	Ψ	Ψ	Psi
Ω	Ω	Ω	Omega
α	α	α	alpha
β	β	β	beta
γ	γ	γ	gamma
δ	δ	δ	delta
ε	ε	ε	epsilon
ζ	ζ	ζ	zeta
η	η	η	eta
θ	θ	θ	theta
ι	ι	ι	iota
κ	κ	κ	kappa
λ	λ	λ	lambda
μ	μ	μ	mu
ν	ν	ν	nu
ξ	ξ	ξ	xi

ο	ο	ο	omicron
π	π	π	pi
ρ	ρ	ρ	rho
ς	ς	ς	sigma final
σ	σ	σ	sigma
τ	τ	τ	tau
υ	υ	υ	upsilon
φ	φ	φ	phi
χ	χ	χ	chi
ψ	ψ	ψ	psi
ω	ω	ω	omega

Sciences

caractère	code texte	code numérique	commentaire
%	%	%	pour cent
+	+	+	plus
<	<	<	inférieur (less-than)
=	=	=	égal
>	>	>	supérieur (greater-than)

ª	ª	ª	indicateur ordinal féminin
¬	¬	¬	crochet de négation
°	°	°	degré
±	±	±	plus-ou-moins
²	²	²	exposant 2
³	³	³	exposant 3
µ	µ	µ	mu (lettre grecque)
¹	¹	¹	exposant 1
º	º	º	indicateur ordinal masculin
¼	¼	¼	un quart
½	½	¼	un demi
¾	¾	¼	trois quarts
×	×	×	multiplication
÷	÷	÷	division

ƒ	ƒ	ƒ	fonction (idem florin)
‰	‰	‰	pour mille
′	′	′	prime, minutes, dérivée

″	″	″	double prime, secondes, dérivée seconde
⁄	⁄	⁄	fraction
∂	∂	∂	dérivée partielle
∏	∏	∏	produit n-aire
∑	∑	∑	somme n-aire
−	−	−	moins
√	√	√	racine carrée
∞	∞	∞	infini
∩	∩	∩	intersection
∫	∫	∫	intégrale
≈	≈	≈	presque égal (asymptotic)
≠	≠	≠	différent (not equal)
≡	≡	≡	équivalent
≤	≤	≤	inférieur ou égal (less or equal)
≥	≥	≥	supérieur ou égal (greater or equal)

Les codes des caractères qui en ISO vont de 160 à 255.

On trouve ci-dessous pour chaque caractère son code ISO (en Hex et en décimal), son code UNICODE (en Hex) et son code UTF-8. Il y a ici deux octets seulement à l'UTF-8 ; l'octet qui indique ce nombre (2) est au dessus en Hex et en Binaire. Le second octet est au dessous. On vérifie que si l'on supprime la partie sur-lignée, on trouve, en concaténant le restant, le code UNICODE.

Exemple pour le caractère de code ISO 161 :

```
C2 1100 0010  ----> 0 0010
                              ------> 000  1010 0001 soit 00 A1
A1 1010 0001  ----> 10 0001
```

rappel : décimal, hexadécimal et binaire.

0	1	2	3	4	5	6	7	8	9	A 10	B 11	C 12	D 13	E 14	F 15
0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

CAR.	ISO 8859-1	UNICODE	UTF-8	CAR.	ISO 8859-1	UNICODE	UTF-8
	A0 / 160	00 A0	C2 1100 0010 A0 1010 0000	¡	A1 / 161	00 A1	C2 1100 0010 A1 1010 0001
¢	A2 / 162	00 A2	C2 1100 0010 A2 1010 0010	£	A3 / 163	00 A3	C2 1100 0010 A3 1010 0011
¤	A4 / 164	00 A4	C2 1100 0010 A4 1010 0100	¥	A5 / 165	00 A5	C2 1100 0010 A5 1010 0101
¦	A6 / 166	00 A6	C2 1100 0010	§	A7 / 167	00 A7	C2 1100 0010

			A6 1010 0110				A7 1010 0111
¨	A8 / 168	00 A8	C2 1100 0010 A8 1010 1000	©	A9 / 169	00 A9	C2 1100 0010 A9 1010 1001
a	AA / 170	00 AA	C2 1100 0010 AA 1010 1010	«	AB / 171	00 AB	C2 1100 0010 AB 1010 1011
¬	AC / 172	00 AC	C2 1100 0010 AC 1010 1100		AD / 173	00 AD	C2 1100 0010 AD 1010 1101
®	AE / 174	00 AE	C2 1100 0010 AE 1010 1110	¯	AF / 175	00 AF	C2 1100 0010 AF 1010 1111

CAR.	ISO 8859-1	UNICODE	UTF-8	CAR.	ISO 8859-1	UNICODE	UTF-8
°	B0 / 176	00 B0	C2 1100 0010 B0 1011 0000	±	B1 / 177	00 B1	C2 1100 0010 B1 1011 0001
2	B2 / 178	00 B2	C2 1100 0010 B2 1011 0010	3	B3 / 179	00 B3	C2 1100 0010 B3 1011 0011
´	B4 / 180	00 B4	C2 1100 0010 B4 1011 0100	µ	B5 / 181	00 B5	C2 1100 0010 B5 1011 0101
¶	B6 / 182	00 B6	C2 1100 0010 B6 1011 0110	·	B7 / 183	00 B7	C2 1100 0010 B7 1011 0111
¸	B8 / 184	00 B8	C2 1100 0010 B8 1011 1000	1	B9 / 185	00 B9	C2 1100 0010 B9 1011 1001
º	BA / 186	00 BA	C2 1100 0010 BA 1011 1010	»	BB / 187	00 BB	C2 1100 0010 BB 1011 1011
¼	BC / 188	00 BC	C2 1100 0010 BC 1011 1100	½	BD / 189	00 BD	C2 1100 0010 BD 1011 1101
¾	BE / 190	00 BE	C2 1100 0010 BE 1011 1110	¿	BF / 191	00 BF	C2 1100 0010 BF 1011 1111

CAR.	ISO 8859-1	UNICODE	UTF-8	CAR.	ISO 8859-1	UNICODE	UTF-8
À	C0 / 192	00 C1	C3 1100 0011 80 1000 0000	Á	C1 / 193	00 C1	C3 1100 0011 81 1000 0001
Â	C2 / 194	00 C2	C3 1100 0011 82 1000 0010	Ã	C3 / 195	00 C3	C3 1100 0011 83 1000 0011
Ä	C4 / 196	00 C4	C3 1100 0011 84 1000 0100	Å	C5 / 197	00 C5	C3 1100 0011 85 1000 0101
Æ	C6 / 198	00 C6	C3 1100 0011 86 1000 0110	Ç	C7 / 199	00 C7	C3 1100 0011 87 1000 0111

È	C8 / 200	00 C8	C3 1100 0011 88 1000 8000	É	C9 / 201	00 C9	C3 1100 0011 89 1000 1001
Ê	CA / 202	00 CA	C3 1100 0011 8A 1000 1010	Ë	CB / 203	00 CB	C3 1100 0011 8B 1000 1011
Ì	CC / 204	00 CC	C3 1100 0011 8C 1000 1100	Í	CD / 205	00 CD	C3 1100 0011 8D 1000 1101
Î	CE / 206	00 CE	C3 1100 0011 8E 1000 1110	Ï	CF / 207	00 CF	C3 1100 0011 8F 1000 1111

CAR.	ISO 8859-1	UNICODE	UTF-8	CAR.	ISO 8859-1	UNICODE	UTF-8
Ð	D0 / 208	00 D1	C3 1100 0011 90 1001 0000	Ñ	D1 / 209	00 D1	C3 1100 0011 91 1001 0001
Ò	D2 / 210	00 D2	C3 1100 0011 92 1001 0010	Ó	D3 / 211	00 D3	C3 1100 0011 93 1001 0011
Ô	D4 / 212	00 D4	C3 1100 0011 94 1001 0100	Õ	D5 / 213	00 D5	C3 1100 0011 95 1001 0101
Ö	D6 / 214	00 D6	C3 1100 0011 96 1001 0110	×	D7 / 215	00 D7	C3 1100 0011 97 1001 0111
Ø	D8 / 216	00 D8	C3 1100 0011 98 1001 8000	Ù	D9 / 217	00 D9	C3 1100 0011 99 1001 1001
Ú	DA / 218	00 DA	C3 1100 0011 9A 1001 1010	Û	DB / 219	00 DB	C3 1100 0011 9B 1001 1011
Ü	DC / 220	00 DC	C3 1100 0011 9C 1001 1100	Ý	DD / 221	00 DD	C3 1100 0011 9D 1001 1101
Þ	DE / 222	00 DE	C3 1100 0011 9E 1001 1110	ß	DF / 223	00 DF	C3 1100 0011 9F 1001 1111

CAR.	ISO 8859-1	UNICODE	UTF-8	CAR.	ISO 8859-1	UNICODE	UTF-8
à	E0 / 224	00 E1	C3 1100 0011 A0 1010 0000	á	E1 / 225	00 E1	C3 1100 0011 A1 1010 0001
â	E2 / 226	00 E2	C3 1100 0011 A2 1010 0010	ã	E3 / 227	00 E3	C3 1100 0011 A3 1010 0011
ä	E4 / 228	00 E4	C3 1100 0011 A4 1010 0100	å	E5 / 229	00 E5	C3 1100 0011 A5 1010 0101
æ	E6 / 230	00 E6	C3 1100 0011 A6 1010 0110	ç	E7 / 231	00 E7	C3 1100 0011 A7 1010 0111
è	E8 / 232	00 E8	C3 1100 0011	é	E9 / 233	00 E9	C3 1100 0011

			A8 1010 8000				A9 1010 1001
ê	EA / 234	00 EA	C3 1100 0011 AA 1010 1010	ë	EB / 235	00 EB	C3 1100 0011 AB 1010 1011
ì	EC / 236	00 EC	C3 1100 0011 AC 1010 1100	í	ED / 237	00 ED	C3 1100 0011 AD 1010 1101
î	EE / 238	00 EE	C3 1100 0011 AE 1010 1110	ï	EF / 239	00 EF	C3 1100 0011 AF 1010 1111

CAR.	ISO 8859-1	UNICODE	UTF-8	CAR.	ISO 8859-1	UNICODE	UTF-8
ð	F0 / 240	00 F1	C3 1100 0011 B0 1011 0000	ñ	F1 / 241	00 F1	C3 1100 0011 B1 1011 0001
ò	F2 / 242	00 F2	C3 1100 0011 B2 1011 0010	ó	F3 / 243	00 F3	C3 1100 0011 B3 1011 0011
ô	F4 / 244	00 F4	C3 1100 0011 B4 1011 0100	õ	F5 / 245	00 F5	C3 1100 0011 B5 1011 0101
ö	F6 / 246	00 F6	C3 1100 0011 B6 1011 0110	÷	F7 / 247	00 F7	C3 1100 0011 B7 1011 0111
ø	F8 / 248	00 F8	C3 1100 0011 B8 1011 8000	ù	F9 / 249	00 C9	C3 1100 0011 B9 1011 1001
ú	FA / 250	00 FA	C3 1100 0011 BA 1011 1010	û	FB / 251	00 FB	C3 1100 0011 BB 1011 1011
ü	FC / 252	00 FC	C3 1100 0011 BC 1011 1100	ý	FD / 253	00 FD	C3 1100 0011 BD 1011 1101
þ	FE / 254	00 FE	C3 1100 0011 BE 1011 1110	ÿ	FF / 255	00 FF	C3 1100 0011 BF 1011 1111

2 analyse de la page HTML

tutoriel HTML

fiche 2

introduction

Dans cette fiche et celles qui suivent, on va se centrer sur la programmation **du corps du fichier,** la partie **<body>**. C'est dans cette partie que l'on référencie les textes, les images, les tableaux, les liens… qui sont affichés à l'écran et les propriétés des éléments constituants cet affichage.

note : *Il y a beaucoup d'éléments nouveaux dans cette fiche ; leur compréhension complète n'est pas requise pour passer aux fiches suivantes. On pourra y revenir ultérieurement.*

1. Une question de méthodologie.

1.1. séparer l'intention de la réalisation.

La bonne méthodologie est une **méthodologie de séparation des genres** :

* d'une part analyser le **rôle** joué par chacun des éléments qui constituent la page ;
* d'autre part, rendre compte de la façon dont on procède pour exprimer **graphiquement** chacun des rôles que l'on a dégagé.

Le marquage des rôles relève du HTML ; le graphisme relève des "feuilles de style".

Autrement dit : quand on écrit le corps de fichier, on marque à l'aide de balises le rôle joué par l'unité que l'on balise : *ceci est un titre, ceci est un sous-titre, ceci est un paragraphe, un formulaire, un tableau de données, un lien vers une autre page, une liste numérotée etc*… Le graphisme effectif ne relève pas du HTML et sa mise au point se fait après -ou pour le moins, en dialectique, avec- l'établissement du fichier HTML.

C'est un point de méthodologie très important, et il est regrettable que la plupart des ouvrages élémentaires, qui portent sur la programmation HTML, enseignent ou fassent le contraire. On appelle parfois cette méthodologie la programmation sémantique (La sémantique est l'étude du langage considéré du point de vue du sens).

1.2. Pourquoi séparer le contrôle de la page de son aspect graphique ?

Pourquoi cette méthodologie, que l'on retrouve également dans la programmation des logiciels d'aujourd'hui (souvent écrits en C++ ou en Java) ?

* La première raison est qu'elle permet de sérier les problèmes lors de d'analyse : chercher d'abord **ce que l'on veut signifier**, et ensuite seulement **comment le dire**. Par exemple telle unité de texte est définie comme *paragraphe*, telle autre comme *citation*, telle autre comme une *expression à mettre en valeur* etc… Bien entendu dans la forme qui sera utilisée, on ne fera pas n'importe quoi ; mais c'est un problème de graphisme, à régler séparément.

* La seconde raison est qu'une page où on ne mélange pas **sens et signe** est plus facile à mettre au point, plus facile à modifier, à discuter en équipe. De plus, la majorité de modifications se fait sur l'aspect, la présentation et pas sur le fond de la page (on ne change pas le cahier des charges une fois la page écrite ! ou alors, il vaut mieux recommencer le chantier).

* La troisième raison est que l'on peut alors confier l'aspect proprement graphique à un spécialiste (le graphiste) : c'est ce que font les bonnes maisons d'édition de sites web. Ce faisant, un changement de style n'oblige pas à retourner à la création de la page HTML. Certains sites laissent même l'internaute en ligne, choisir son style d'affichage (couleurs, ambiance de la page, grandeur des caractères, polices etc…). C'est toujours la même page HTML qui est affichée, avec une liaison sur le module

graphique, que l'on peut choisir.

1.3. note importante sur l'histoire du HTML.

Au départ du HTML, il y a la création d'un outil simple qui permet de programmer une unité hypertexte (des textes mis en réseau) adaptée à l'interface graphique des ordinateurs personnels. Dans les années 1990, l'interface graphique devient la règle dans le monde des ordinateurs de bureau (Mac ou PC). Le HTML est donc créé pour répondre à une question d'actualité : comment créer un système de pages hypertextes, adapté aux capacités graphiques des ordinateurs personnels et à la capacité de transmission des lignes téléphoniques locales ? Jusque là, on utilise dans les réseaux (BBS) une interface texte, c'est-à-dire de l'équivalent électronique des textes dactylographiés ou télétypés ; la transmission du signal ne pose pas de problème dans ce cas depuis fort longtemps (travaux de Baudot, mort en 1903).

On peut désormais utiliser des caractères de différentes familles, de différentes dimensions, avec des attributs multiples (épaisseur, inclinaison, couleur, souligné, barré, sur-ligné…), des images, des dispositions en tableau de données etc. Exactement comme dans les livres.

L'idée des promoteurs des premières normes HTML est que **tout** ce dont on doit disposer comme caractéristiques doit être codifié à l'aide de balises : l'orientation première du HTML est d'abord **marquer** (*hyper text markup language*) **l'aspect** recherché : *tel élément de texte doit être réalisé avec telle fonte, telle dimension, telle couleur etc… Tel tableau de données doit être réalisé avec telles dimensions de cases, tel genre, telle couleur, telle épaisseur de traits de séparation, telle disposition (gauche, centre, droite) dans la page etc…***En priorité, on trouve des caractéristiques de présentation**. Ce type de parti pris tient évidemment compte des performances des ordinateurs de l'époque et des capacités de transmission de l'information sur les lignes téléphoniques : ordinateurs assez puissants pour supporter l'interface graphique, pas assez pour réaliser en temps réels des calculs complexes ; lignes de transmission dont le faible débit réclame l'allégement maximum des fichiers à transmettre, mais qui est suffisant pour transmettre des images et supporter un langage interprété.

Les performances évoluent vite et les pages HTML peuvent alors gagner en volume. D'autre part, on peut réaliser une certaine sophistication des éléments graphiques puisqu'on dispose de machines qui calculent de plus en plus rapidement. Les algorithmes de compression évoluent également, surtout pour les besoins de la télévision numérique (MPEG, JPEG...).

Suivant en cela une tendance que l'on retrouve dans tout le mode du logiciel, la méthode d'analyse des pages hypertextes évolue. Petit à petit, avec l'apparition des "**feuilles de style"**, **tout ce qui relève du graphisme pur devient caduc dans la représentation HTML**. Quant à la gestion dynamique (formulaire, liens hypertextes, remplacement d'images, boutons animés…) elle utilise les langages de script comme javascript ou des plugins comme **flash** pour compléter les manques du HTML (contrôles de formulaires, animation élémentaire des pages). Reste spécifique au langage de balise HTML, tout ce qui concerne la structuration de la page, en s'attachant **non plus à l'aspect** que doit avoir chaque élément constitutif, mais à la **signification du rôle** qu'il y joue (*titre, tableau de données, formulaire, liste, regroupement d'une séquence en un élément cohérent –entête de page, pied de page, grandes articulations de la page*-, *isolement d'un particularisme local –expression à mettre en valeur*, *néologisme*- etc).

2. Un exemple d'analyse.

Dans cette section, on va s'attacher à montrer sur un exemple, en quoi consiste **actuellement** l'analyse et la programmation d'une page HTML. L'exemple est un peu formel ; mais son objectif est d'insister sur **le type de questionnement** qui préside à l'établissement d'une page HTML aujourd'hui. Le contenu de cette page n'a pas à être pris en considération, ou seulement de façon marginale.

2.1. La page en mode texte, telles qu'elle serait rendue par une vieille machine à écrire.

```
le codage des images pour le web
" Un bon croquis vaut mieux qu'un long discours. " NAPOLÉON
```

1. Contraintes du codage.

Une image informatique peut être représentée de deux façons :

Soit on fait un programme qui dit comment dessiner l'image : dessiner sur le canevas un rectangle avec telles et telles caractéristiques, un cercle... un pavé de texte...C'est la méthode vectorielle ; pour le web, cette démarche n'est pas généralisée actuellement (avec l'avènement du SVG, format qui reste actuellement marginal, cela va changer).

Soit on décrit l'image comme un tableau de données ; l'atome de ce tableau est le pixel. Une webcam fabrique une image de 640 pixels de large et 480 pixels de haut, soit 307200 pixels. En monochrome, le pixel est codé sur un octet et autorise dont 256 nuances ; en couleur, sur trois octets, et autorise ainsi 1 677 216 nuances. Mais cette petite image, considérée du point de vue du stockage ou de la transmission nécessite 300 kO en monochrome, 1 MO en couleur.

C'est inacceptable en l'état pour l'utilisation sur le web !

D'où la nécessité de compresser les images. Ce n'est pas une exigence spécifique au web, mais le web correspond à un cas d'espèce. En infographie on peut accepter des temps de compression élevés, mais aucune perte de qualité ; si l'imprimeur demande un fichier pdf pour son tirage, on fait une sauvegarde à 200%. Dans le web, le temps de décompression doit être faible, l'information à transmettre faible également, mais on peut accepter une certaine perte de qualité de l'image.

2. Les choix actuels.
Il y a trois modes de compression utilisées sur le web :

*** la méthode par palette (gif, png-palette).**
*** la méthode de la compression de fichier sans perte (gif, png).**
*** la méthode par représentation mathématique de pavés 64x64 pixels (jpeg).**

Ces trois modes seront étudiés dans les pages qui suivent.

2.1. principe de la méthode par palette (format gif ou png-palette).

On choisit en général une gamme de 256 teintes numérotées de 0 à 255, et que l'on code sur trois octets. Chaque pixel est ensuite codé par un octet, qui est le numéro de la teinte la plus approchante. Il y a évidemment perte d'informations, mais quasi indécelable dans l'affichage sur écran. On gagne un facteur 3 dans la taille des fichiers. D'autre choix de dimension de palette sont possibles (16 ou même moins).

2.2. principe de la compression sans perte.

Il existe de nombreux algorithmes mathématiques qui utilisent le fait qu'un fichier d'octets qui code un texte ou une image n'est pas aléatoire, mais présente des régularités que l'on peut exploiter pour le compresser. Ce procédé est utilisé dans le png normal (mais aussi dans les format TIFF, PS, PCX, PDF), et permet encore un léger gain dans les systèmes à palette. Une compression jusqu'à 10 est possible.
Mais en général la performance est plus faible (cela dépend de l'image).

2.3. principe des algorithmes mathématiques.

Le principe est semblables à la décomposition de Fresnel pour les courbes ; mathématiquement complexe, le jpeg autorise le choix du taux de compressions, aux dépens de la qualité d'image. On code par pavés de 4096 pixels ; il apparaît assez rapidement des artefacts (traces dues à la manière de faire le code).

3. Voici une idée des résultats pour une photo couleur de fleurs de 816 x 612 =499 392 pixels

type	brut	png 24b non cp	png pal non cp	png cp max	png pal cp max	gif	jpeg non cp	jpeg cp 10	jpeg cp 30	jpeg cp 50	jpeg cp 80
k-octets	1498	1467	490	1189	420	420	691	346	183	118	78
qualité		max	exc	max	exc	exc	exc	tb	ab	limite	mauv

2.2. Quelques catégories d'analyse.

* les titres et sous-titres.

En premier lieu on trouve le titre général :

```
le codage des images pour le web
```

Il existe 7 balises pour le texte "normal" dont les nom sont **h1, h2, h3 … h6 et p.** Par convention, les balises **h** sont l'équivalent du style "**titre"** des traitements de texte, et "**p"** le style **normal** ou **standard**. Cela ne préjuge pas de la manière de les afficher (les navigateurs on un affichage par défaut qui fonctionne en cas d'absence de précisions apportées par les feuille de style).

h1 est le nom de la balise équivalente au *titre 1* dans les traitements de texte. On aura donc de façon naturelle dans le code de la page les lignes suivantes :

```
<h1>le codage de images pour le web</h1>. . . . .
<h2>1. Contraintes du codage.</h2>. . . .
<h3>2.1. principe de la méthode par palette.</h3>
etc
```

Mais l'interjection qui suit a un rôle particulier dans le texte : ce n'est pas un titre à proprement parler. Dans un traitement de texte, on lui dévoluerait un style particulier. Comme en HTML on ne peut pas définir de balise, on propose de lui affecter la balise **h4**. (en effet, trois niveaux de titres et sous-titres, c'est déjà bien).

```
<h4>C'est inacceptable en l'état pour l'utilisation sur le web !</h4>
```

* les paragraphes.

Quelles sont les parties qui relèvent de la notion de paragraphe ?

En gros, tout ce qui est texte d'explication un peu extensif ; mais on peut objecter que tous les paragraphes ainsi définis, s'ils ont même type de fontes, même saut de paragraphe, n'auront vraisemblablement pas le même retrait. Cela ne concerne pas le choix des balises mais **leur qualification** : ce sont les attributs de la balise "**p**", concernant le mode d'affichage qui différeront. **Cette question n'est donc pas pertinente à ce niveau d'analyse** :

```
<p>D'où la nécessité de compresser les images. Ce n'est pas une exigence spécifique
au web, mais le web correspond à un cas . . . . . doit être faible, l'information à
transmettre faible, mais on peut accepter une certaine perte de qualité de
l'image.</p>
```

* liste.

L'élément suivant nécessite une attention particulière :

```
Il y a trois modes de compression utilisées sur le web :

* la méthode par palette (gif, png-palette).
* la méthode de la compression de fichier sans perte (gif, png).
* la méthode par représentation mathématique de pavés 64x64 pixels (jpeg).

Ces trois modes seront étudiés dans les pages qui suivent.
```

La première et la dernière ligne relèvent de la notion de paragraphe (peut-être de titre pour la première), mais les trois lignes intermédiaires constituent une unité organisée, une liste ; le balisage comporte deux balises : celle indiquant que l'on a une liste et l'autre, un item de la liste. Ce qui donne :

```
<p>Il y a trois modes de compression utilisées sur le web :</p>
<ul><!--début de la liste -->
    <li>la méthode par palette (gif, png-palette)</li>
    <li>la méthode de la compression de fichier sans perte (gif, png)</li>
    <li>la méthode par représentation mathématique de pavés 64x64 pixels
(jpeg)</li>
</ul><!--fin de la liste -->
<p>Ces trois modes seront étudiés dans les pages qui suivent.</p>
```

L'indentation et les commentaire sont uniquement des aides à la programmation.

* liens.

Cependant, il est indiqué que les trois modes doivent être étudiés dans les pages suivantes ; il est donc pertinent de considérer chaque élément de la liste comme un lien vers une page spécifique. On a donc à définir ces liens, par la balise **a** (ancre / anchor / lien ancré). Cette balise a obligatoirement un attribut qui la qualifie, l'attribut **href** (on a mis un argument vraisemblable).

On a en définitive le code suivant, avec imbrication :

```
<ul><!--début de la liste -->

    <li>
      <a href="palette.html">* la méthode par palette (gif, png-palette)</a>
    </li>

    <li>
       <a href="sansperte.html">
          * la méthode de la compression de fichier sans perte (gif, png)
       </a>
    </li>

    <li>
       <a href="mathematique.html">
          * la méthode par représentation mathématique de pavés 64x64 pixels (jpeg)
       </a>
    </li>

</ul><!--fin de la liste -->
```

On commence à voir l'intérêt des modes de disposition avec indentation pour contrôler un texte qui atteint une certaine complexité…

note : l'ajout des indicateurs de navigation (liens ancrés) n'est pas repris dans les corrections.

* citation.

La citation mise en exergue ne relève pas de la catégorie paragraphe, mais en bon HTML de la catégorie "citation", désignée par la balise **blockquote**. En général, une citation est en retrait ; et dans ce cas, si on imbrique plusieurs blockquote, les retraits s'ajoutent… Mais cela non plus ne relève pas du HTML.

```
<blockquote>" Un bon croquis vaut mieux qu'un long discours. " NAPOLÉON</blockquote>
```

* tableau de données.

type	brut	png 24b non cp	png pal non cp	png cp max	png pal cp max	gif	jpeg non cp	jpeg cp 10	jpeg cp 30	jpeg cp 50	jpeg cp 90
k-octets	1498	1467	490	1189	420	420	691	346	183	118	78
qualité		max	exc	max	exc	exc	exc	tb	ab	limite	mauv

Le tableau en HTML est destiné à présenter des données, et **surtout pas à faire de la mise en page** comme on le voit souvent (ce qui conduit *in fine* à des pages mal interprétées par les navigateurs et illisibles).

Il y a au moins trois balises de tableau qui sont obligatoirement présentes : **<table>,** qui encadre le tableau **<tr>** (t*able row* / rang de tableau) qui encadre les lignes et **<td>** (*table data* / donnée de tableau). Il existe d'autres balises de tableau d'utilité plus relative (**tbody, th**), exclues de notre étude. Voici une amorce de la déclaration du tableau :

```
<table>
    <tr>  <!--premier rang -->
        <td>type</td>
        <td>png 24b<br/>non cp</td>
        . . . . . .
    </tr>
    <tr>  <!-- deuxième rang -->
        <td>k-octets</td>
        <td>1498</td>
        . . . . . .
    </tr>
    <tr>  <!-- troisième rang -->
        <td> </td> <!-- pas de vide -->
        <td>max</td>
        . . . . . .
    </tr>
</table>
```

Attention. Toutes les lignes doivent avoir le même nombre de balises td ; sinon on a n'importe quoi au final ! Il existe cependant des attributs spécifiques qui permettent de moduler cette règle, et permettant la fusion de cellules contiguës.

* un peu d'emphase.

Dans le premier paragraphe, on peut être amené à distinguer les deux expressions importantes :

comment dessiner l'image et **image comme un tableau de données**

A la machine à écrire, on soulignerait, où on passerait deux fois le texte pour le rendre plus gras ; dans un article de journal, on utiliserait un caractère plus fort, où l'italique… Il y a de multiple procédés, mais ce n'est pas ce qui est intéressant dans l'analyse HTML : la seule chose à retenir, c'est que l'on veut **marquer ce texte comme important**. La bonne attitude est ici de placer la balise qui sert à attirer l'attention sur une expression à l'intérieur du paragraphe ; cette balise a pour nom **em** (emphase) ; si on veut une marque plus forte, on utilise la balise **strong** (puissant).

```
<p>Soit on fait un programme qui dit <em>comment dessiner l'image</em> : dessiner
sur le canevas un rectangle avec telles et telles caractéristiques. . . .
```

* des images ?

Il est judicieux que la compression dont on parle soit illustrée ; et même peut-être par plusieurs images. On propose par exemple, un échantillon de quatre compressions : png-palette, png, jpeg 30, jpeg 90.

Les échantillons sous forme de fichiers sont proposés. Et pour chacun, il faut mettre une légende. Deux analyses sont possibles. Soit faire un tableau de données ; les images sont des données, tout comme les textes. Soit créer des unités de groupement pour chaque couple image/légende.

Dans le premier cas on a :

```
<table>
    <tr>
       <td><img scr="png-pal.png"></td>
       <td><img scr="png-normal.png"></td>
       <td><img scr="jpeg30.jpg"></td>
       <td><img scr="jpeg90.jpg"></td>
    </tr>
    <tr>
       < td ><p>image avec palette optimisée png</p></td>
       < td ><p>image normale 24 bits png</p></td>
       < td ><p>image jpeg ; compression 30</p></td>
       < td ><p>image jpeg ; compression 90</p></td>
    </tr>
</table>
```

et dans le second :

```
<div>
    <img src="png-pal.png">
    <p>image avec palette optimisée png</p>
</div>
<div>
    <img src="png-normal.png">
    <p>image normale 24 bits png</p>
</div>
<div>
    <img src="jpeg30.jpg">
    <p> image jpeg ; compression 30</p>
</div>
<div>
    <img src="jpeg90.jpg">
    <p>image jpeg ; compression 90</p>
</div>
```

3. Les attributs.

3.1. qualification.

Jusque là on a toujours parlé **balise** : c'est normal puisque la balise est l'élément structurant du code. Mais on a déjà vu la nécessité de **qualifier** les balises par des attributs : par exemple la balise d'image **`img`** est qualifiée par la référence à la source où le serveur peut trouver le fichier d'image. Le qualificatif (= **attribut**) a la forme **`scr="jpeg90.jpg".`**

Les attributs se présentent en général sous la forme **clef/valeur** : la clef est le nom de l'attribut, et la valeur est donnée par une chaîne de caractères, entre quotes (simples ou doubles, mais de façon cohérente : soit deux apostrophes, soit deux guillemets américains), même si c'est une valeur numérique : size="40". Le séparateur (entre la clef et la valeur) est le signe = . Les espaces (ou assimilés) ne sont pas pris en compte (autour du =, avant ou après l'attribut).

L'ordre des attributs qui qualifient une balise est indifférent. Quelques balises : **`img, a, input, form`** ne peuvent pas se passer d'attribut, par exemple l'attribut de source pour **`img`**, ou l'attribut de type pour la balises **`input`**. Dans tous les autres cas les attributs sont facultatifs ; le navigateur utilise des attributs par défaut en cas de manque. Si un attribut est inconnu ou non interprétable (une faute d'orthographe, un attribut inconnu pour le navigateur), il est skippé (sauté) sans avertissement.

recommandation : mettre les clefs d'attributs en minuscules, même si la casse est théoriquement indifférente. Par contre la valeur est sensible à la casse ; par exemple : le fichier "Jpeg90.jpg" n'est pas le même que "jpeg90.jpg", même si sous Windows ils ne sont pas différenciés ; c'est un piège classique : le site fonctionne bien en mode local sous Windows, mais il disjoncte quand on le place chez l'hébergeur qui est sous Unix… Pour les valeurs, essayer d'en rester à l'ASCII pur (pas de caractères accentués) et ne pas utiliser de trait d'union (le moins mathématique), car on ne connaît pas le codage utilisé chez l'hébergeur, et les langages de script peuvent donner un sens particulier au trait d'union. En cas de doute, utiliser l'entité HTML (fiche 1).

3.2. Les quatre attributs universels.

*** identification**

Toute balise peut être identifiée : la syntaxe est du type **`id="`***`valeur`***`"`** ; la règle d'identification impose que la valeur n'est utilisée qu'une fois dans une page donnée. Il s'agit donc toujours d'un **nom spécifique dans la page**. Rien n'interdit que ce même nom soit utilisé dans des pages différentes.

L'attribut d'identification sert essentiellement lorsque l'on veut utiliser des feuilles de style bien particulières puisqu'elles ne s'adressent qu'à un élément par page et, le plus souvent, lorsqu'on veut traiter une balise et son contenu à l'aide d'un script javascript.

*** classe et style**

Toute balise se rapportant à un affichage (**`h, p, img, input, div, span, ul, li, table,`**

tr, td, etc…), c'est-à-dire la majorité des balises peut posséder l'attribut **class** et l'attribut **style.**
La valeur de l'attribut **class** est le nom d'une feuille de style connue dans la page. L'attribut **class** constitue un lien privilégié entre le code HTML et le code des feuilles de style. La valeur de l'attribut **style** est purement et simplement le contenu d'une feuille de style. On se doute que l'attribut **class** s'utilise de manière très courante, alors que l'attribut **style** va à l'encontre de la méthode de séparation du code et du graphisme. En pratique, on garde l'attribut **style** pour la mise au point du texte, au moment des essais (d'où son fréquent emploi dans le tutoriel). On ne devrait jamais le voir en production. On retrouvera le même genre problème avec les scripts javascript qui peuvent se placer dans les balises ; c'est un usage qu'il vaut mieux éviter comme on le verra plus tard, mais pratique lors de la mise au point.

*** title**

Le survol de la plupart des éléments graphiques peut s'accompagner du surgissement d'une petite fenêtre (*popup)* qui comporte un texte court. Cette fenêtre de survol est parfois d'une couleur fluorescente et comporte une indication du genre : *nature de l'élément, action à effectuer*. En l'absence de l'attribut **title,** il ne se passe rien. L'attribut se déclare comme dans l'exemple :

title = "cliquer pour valider". Une tendance actuelle est de simuler l'attribut **title** grâce à l'utilisation de script javascript, car l'attribut **title** a une présentation non paramétrable dans le fichier et donc figée. On peut simuler un petit *popup* qui ressemble à un **title**. On trouvera en fiche 15, dans l'exemple **html15tp6.html** une simulation complète avec javascript.

3.3. Les attributs spécifiques.

Les autres attributs sont spécifiques aux balises ou aux familles de balises : **href** est un attribut pour la balise **a, src** pour la balise **img, name** et **value** pour les balises **input** …

Il faut faire ici une mise en garde sur l'attribut **name** : **il n'appartient, quoiqu'on puisse lire dans les ouvrage sur le HTML, ni à la balise form, ni à la balise img**. L'utiliser dans ces balises peut conduire à des comportements bizarres et de toute façon non adaptés.

Il existe une famille d'attributs dit "**attributs événements"** ; relatifs à la "réactivité" des pages HTML aux événements extérieurs. Au départ du HTML, seuls les **liens** HTML apportaient une touche de réactivité à la page : un clic sur un lien et une page nouvelle était chargée. Assez rapidement, on a vu apparaître les images réactives (balise **map**, aujourd'hui désuète) et les boutons et images de validation de formulaire.

La tendance actuelle est, sauf pour les liens hypertexte et la validation de formulaire, de rejeter la réactivité du côté de javascript : mais là aussi, il y a deux écoles. Soit on fait le lien avec les pages de scripts par la balise **id**, soit en utilisant un attribut spécifique. La première méthode est utilisée par les puristes mais la seconde, plus traditionnelle, a quand même le mérite d'être évidente à mettre en œuvre sans surcharge de code (ni considérations théoriques complexes), et elle est le pendant pour l'interactivité de l'attribut **class** et de l'attribut **style** pour les feuilles de style. C'est cette méthode traditionnelle que l'on va plutôt utiliser ; mais ce n'est qu'une question de transposition que de passer à la nouvelle disposition (voir un exemple dans la fiche 15). Les attributs ont pour nom : **onLoad, onClick, onMouseOver, onMouseOut, onFocus, onBlur, onKeypressed** etc…

Attention : Comme on peut s'y attendre, la fin du chargement d'une page ou d'une image, le déplacement de la souris sur la page, la frappe d'une touche au clavier, le gain où la perte du focus par un élément constituent des événements. Au contraire, **la disparition d'une page n'en est pas un.** Si on ne réalise pas un appel explicite (utilisation d'un lien ancré), on peut quitter une page, voire le navigateur sans que cela soit détectable comme événement. C'est là une limite bien gênante du protocole HTML qui est due à son principe même ; c'est un protocole sans état (la notion de page active n'existe pas pour le serveur).

Annexe.

En guise de TP, on peut examiner la page HTML **html02tp0.html**, qui retient pour les attributs :

id pour les balises ayant une spécificité dans la page : **h4, a, blockquote**

class pour les balises **p** selon le mode d'affichage spécifique qu'elles peuvent avoir (retrait par exemple). On a ainsi **class="normal"**, **classe="retrait"**.

Jouer alors la page dans le navigateur (soit comme fichier, soit comme page web à l'aide d'un serveur local). La correction est donnée en **html02tp0.html**

voici ce fichier de correction :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
 <head>
    <!--  ATTENTION :
    Suivant votre système d'exploitation, votre éditeur de texte
    ou le réglages de votre éditeur, la valeur de  charset  peut être différente
    -->
    <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
    <title> html02corr </title>
  </head>
  <body>
    <h1>le codage de images pour le web</h1>
      <blockquote id="napoleon">
        " Un bon croquis vaut mieux qu'un long discours. " NAPOLÉON
      </blockquote>
      <h2>1. Contraintes du codage.</h2>
      <p class="normal">
        Une image informatique peut être représentée de deux façons :
      </p>
      <p class="retrait"> Soit on fait un programme qui dit comment dessiner l'image
: dessiner sur le canevas un rectangle avec telles et telles caractéristiques, un
cercle, un pavé de texte etc... C'est la méthode vectorielle ; pour le web, cette
démarche n'est pas généralisée actuellement (avec l'avènement du SVG, format qui
reste actuellement marginal, cela va changer).
      </p>
      <p class="retrait"> Soit on décrit l'image comme un tableau de données ;
l'atome de ce tableau est le pixel. Une webcam fabrique une image de 640 pixels de
large et 480 pixels de haut, soit 307200 pixels. En monochrome, le pixel est codé sur
un octet et autorise dont 256 nuances ; en couleur, sur trois octets, et autorise
ainsi 1 677 216 nuances. Mais cette petite image, considérée du point de vue du
stockage ou de la transmission nécessite 300 kO en monochrome, 1 MO en couleur.
      </p>
      <h4 id="inacceptable">
        C'est inacceptable en l'état pour l'utilisation sur le web !</h4>
      <p class="normal"> D'où la nécessité de compresser les images. Ce n'est pas une
exigence spécifique au web, mais le web correspond à un cas d'espèce. En infographie
on peut accepter des temps de compression  élevés, mais aucune perte de qualité ; si
l'imprimeur demande un fichier pdf pour son tirage, on fait une sauvegarde à 200%.
Dans le web, le temps de décompression doit être faible, l'information à transmettre
faible également, mais on peut accepter une certaine perte de qualité de l'image.
      </p>
      <h2>2. Les choix actuels.</h2>
      <p class="normal"> Il y a trois modes de compression utilisées sur le web :
      </p>
      <ul>
        <li>la méthode par palette (gif, png-palette).
        </li>
        <li>la méthode de la compression de fichier sans perte (gif, png).
        </li>
        <li>la méthode par représentation mathématique de pavés 64x64 pixels (jpeg).
        </li>
      </ul>      <p class="normal"> Ces trois modes seront étudiés dans les pages qui
suivent.
      </p>
      <h3> 2.1. principe de la méthode par palette (format gif ou png-palette).</h3>
      <p class="normal"> On choisit en général une gamme de 256 teintes numérotées de
0 à 255, et que l'on code sur trois octets. Chaque pixel est ensuite codé par un
```

```
octet, qui est le numéro de la teinte la plus approchante. Il y a évidemment perte
d'informations, mais quasi indécelable dans l'affichage sur écran. On gagne un
facteur 3 dans la taille des fichiers. D'autre choix de dimension de palette sont
possibles (16 ou même moins).
      </p>
      <h3>2.2. principe de la compression sans perte.</h3>
      <p class="normal">
        Il existe de nombreux algorithmes mathématiques qui utilisent le fait qu'un
fichier d'octets qui code un texte ou une image n'est pas aléatoire, mais présente
des régularités que l'on peut exploiter pour le compresser. Ce procédé est utilisé
dans le png normal (mais aussi dans les format TIFF, PS, PCX, PDF), et permet encore
un léger gain dans les systèmes à palette. Une compression jusqu'à 10 est
possible.<br /> Mais en général la performance est plus faible (cela dépend de
l'image).
      </p>
      <h3>2.3. principe des algorithmes mathématiques.</h3>
      <p class="normal"> Le principe est semblables à la décomposition de Fresnel
pour les courbes ; mathématiquement complexe, le jpeg autorise le choix du taux de
compressions, aux dépens de la qualité d'image. On code par pavés de 4096 pixels ; il
apparaît assez rapidement des artefacts (traces dues à la manière de faire le code).
      </p>
      <h2>3. Voici une idée des résultats pour une photo couleur de fleurs de 816 x
612 =499 392 pixels</h2>
      <table>
        <tr>
          <td>type</td>
          <td> brut</td>
          <td> png 24b<br />non cp</td>
          <td>png pal<br />non cp</td>
          <td>png 24b<br />cp max</td>
          <td>png pal<br /> cp max</td>
          <td> gif</td>
          <td>jpeg<br />non cp</td>
          <td>jpeg<br />cp 10</td>
          <td>jpeg<br />cp 30</td>
          <td>jpeg<br />cp 50</td>
          <td>jpeg<br />cp 80</td>
        </tr>
        <tr>
          <td>k-octets</td>
          <td>1498</td>
          <td> 1467</td>
          <td>490</td>
          <td>1189</td>
          <td>420</td>
          <td>420</td>
          <td>691</td>
          <td>346</td>
          <td>183</td>
          <td>118</td>
          <td>78</td>
        </tr>
        <tr>
          <td>qualité</td>
          <td> </td>
          <td>max</td>
          <td>exc</td>
          <td>max</td>
          <td>exc</td>
          <td>exc</td>
          <td>max</td>
          <td>exc</td>
          <td>tb</td>
          <td>limite</td>
          <td>mauvais</td>
        </tr>
```

```
      </table>
  </body>
</html>
```

* on note l'usage de la balise ouverte <br /> qui signifie le passage à la ligne, sans création de nouveau paragraphe.

* L'essai dans plusieurs navigateurs donne une même allure d'ensemble, mais des aspects de caractères, d'interligne etc. divers.

* Il faut cependant remarquer que globalement, l'effet est assez voisin de celui qu'on attend : bon respect de la notion de titre, sous-titre, paragraphe, citation, mise en page de données etc…

On n'a pas retenu toutes les remarques faites ci-dessus (listes, lien, images) car on reprendra ces points avec plus de détails ultérieurement.

3 le flot HTML et les balises

tutoriel HTML

fiche 3

introduction.

On s'intéresse ici uniquement à la partie affichée du fichier HTML, celle qui est incluse dans les balises **body** (corps**)**. On a vu dans la fiche précédente comment aborder l'analyse et la structuration du corps de la page HTML. Dans cette fiche, on va examiner quelques concepts liés à l'affichage.

note : les fiches du tutoriel ne sont pas une encyclopédie du HTML et de ce qui lui est associé. On renvoie pour cela au site **selfhtml.org** ; on peut le télécharger. Le système de recherche, en page d'accueil est un peu complexe, mais il permet de trouver à peu près tout. Ce site est remarquablement précis et contient très peu d'erreur. Il y a une multitude d'exemples. On pourrait, et c'est le péché lié à son exhaustivité, lui reprocher de ne pas bien signaler **ce qui appartient à l'ancien HTML et ce qui caractérise la programmation web actuelle**.

1. Affichage (display) block et inline.

Les balises portent par défaut un certain nombre de caractéristiques générales de **structure** et d'**affichage**. Par exemple la balise **<p>** (paragraphe) possède les caractéristiques naturelles d'un paragraphe : contenir du texte, l'afficher de façon continue dans la zone qui lui est dévolue (l'espace affichage de l'écran, la case d'un tableau), sauter de ligne en fin de paragraphe avec un interligne espaçant les paragraphes. Les propriétés qui viennent d'être évoquées caractérisent les éléments **block**. On peut aussi limiter l'espace d'affichage à un rectangle prédéfini (voir les feuilles de style). Par opposition, si on considère un caractère, une image, les contenus des balises de gras **<b>** ou d'italique **<i>** , ils s'affichent à la queue leu leu, sans saut de ligne ; on les dit **inline**. En principe, on ne redimensionne pas les éléments **inline** (sauf les images), car ils respectent les caractéristiques de leurs constituants (des textes le plus souvent).

On prendra garde qu'il s'agit là de propriétés par défaut assez naturelles et non de propriétés attachées aux balises : la propriété d'affichage peut toujours être redéfinie ; on verra cela ultérieurement. Il existe une troisième propriété abordable avec les feuilles de style, qui s'appelle la propriété **none** (rien) et qui mutifie la balise ; elle rend inexistante la balise qui la possède : avec **display:none;** il n'y a ni affichage, ni aucune place réservée pour le contenu de la balise… Cette propriété curieuse est cependant fort utile dans le HTML dynamique : la balise existe, elle fait partie du texte HTML chargé par le navigateur et elle n'est pas affichée... tant qu'un script ne vient pas modifier la valeur de l'attribut **display.**

2. Conteneurs.

Sans qu'il s'agisse à proprement parler d'une propriété des balises, la notion de conteneur est cependant un concept utile ; un conteneur est une balise qui "contient" des éléments HTML, textes, images, autres balises (emboîtées) dont certaines peuvent aussi être des conteneurs. Une image n'est pas un conteneur.

Les éléments contenus sont les fils (enfants) du conteneurs, et ils héritent *assez naturellement* de propriétés du conteneur parent : par exemple les caractères d'affichage de texte.

Le super conteneur est le conteneur ** (block)** qui n'a pas de propriétés marquantes autres que celles de pouvoir contenir à peu près tout. Le plus petit conteneur est le conteneur **** (**inline**) qui, en principe ne contient que des éléments inline (texte, voire image) et normalement, aucune balise.

Pour les conteneurs de type block (**div, h1, h2..,p, li**), on peut définir des marges d'affichage, des bordures, des fontes, des couleurs, des dimensions, des données de positionnement etc. Les conteneur inline sont conditionnés par leur contenu ; ils n'ont pas de dimensions

définissables, leurs données de positionnement sont imédiates, mais on peut les colorer, les bordurer, en changer la fonte... Ceci et l'objet des feuilles de style qui seront étudiée plus tard.

Pour avoir une première idée du fonctionnement d'un conteneur, supposons que l'on dispose de la structure suivante :

```
<div>
  <p>bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla</p>
  <p>bli bli bli bli bli bli bli bli bli bli bli bli bli bli bli</p>
</div>
```

Le texte inclus dans les balises **p** se disposera dans les limites (définies dans une feuille de style) de la balise **div**, en héritant éventuellement des styles d'affichage qui peuvent être définis dans la balise **div** (fonte et propriétés, comme la taille, la couleur, le souligné, le gras).

3. flot HTML.

Au début du HTML, les propriétés liées aux balises étaient intangibles. Le balises **block** étaient définitivement des balises **block**. Une page HTML était une séquences d'ordres d'affichage, avec succession sur la page affichée des différents éléments **block**, chacun étant constitué d'éléments **inline**. La première analyse du corps d'une page se fait selon le schéma qui vient d'être évoqué : le "flot HTML" détermine complètement la page affichée...

La situation actuelle est plus riche. En premier lieu, on peut désormais jouer sur l'affichage. Un élément qui a naturellement un comportement block peut se voir imposer un comportement inline. (et réciproquement) ; on peut sortir du flot HTML, ce qui en première approximation revient à superposer ou juxtaposer des éléments **block,** comme si on travailait sur des calques superposables, comme en dessin industriel. Ce dernier point, assez délicat, est appelé le **positionnement** avec sortie du flot et fait l'objet d'une étude particulière. Dans un premier temps, on n'a pas à se préoccuper de ce problème (d'autant plus que les divers navigateurs n'en sont pas au même point dans la prise en charge du positionnement) ; on raisonne toujours en termes de "flot HTML".

4. Les balises de texte.

4.1. Titres et paragraphes.

Rappelons qu'en HTML, il faut s'attacher **au rôle des balises et pas à l'aspect qu'elles confèrent à leur contenu**, même si on ne peut en faire complètement abstraction. Il faut avoir en tête que, dans un document de production, toute balise a son style redéfini par des feuilles de style (un document de production est un document destiné au serveur, mis au point en vue de la diffusion publique).

Il existe 6+1 balises ordinaires : **h1, h2, ..h6** et **p**. Ce sont toutes des balises doubles (attention aux oublis ou mauvaises dispositions, qui conduisent à des affichages inattendus).

Les balise **h** sont dévolues aux titres, avec des numéros décroissants (titre, sous-titre, sous-sous-titre etc), et la balise **p** aux paragraphes. Comme on n'a jamais 6 niveaux de titre, les numéros les plus grands (h4, h5, h6) sont souvent affectées à des textes particuliers (avertissement, copyright…).

4.2. code préformaté.

La balise **pre** constitue **une négation** de ce qui a été dit sur la prise en charge du texte : ***le retour à la ligne et les suites d'espaces sont pris en charge sans changement à l'intérieur d'une balise pre.*** Cette balise provient du temps où le HTML n'avait pas de balise pour les tableaux. Les dispositions en tableaux de données étaient faites en mode texte, en simulant un travail fait à la machine la machine à écrire.

Le texte intérieur à la balise **pre** est affiché tel quel, avec une fonte à espacement fixe. Ce conteneur de genre **block** est destiné aujourd'hui à afficher du code informatique, où tout texte qui doit être reproduit tel quel (rarissime). Il vaut mieux ne pas utiliser de balises incluses dans la balise **pre** qui modifient les dimensions ; l'effet peut être imprévisible (caractéristiques de fonte et respect des coupures de ligne restent prioritaires). Il existe une feuille de style qui permet d'éviter de se servir de la balise **pre (**il s'agit de **white-space:pre;).**

4.3. modifications de texte (inline).

A première vue, les modifications effectuées dans le texte relèvent plus de l'aspect graphique que de la signification. On préférera les balises de modification chargées de sens, et que l'on peut redéfinir en accord avec le graphisme général :

em (emphasis) : le segment de texte est important.

strong : il faut rendre le segment de texte plus évident encore.

sup : il faut mettre en d'exposant.

sub (subscript): il faut mettre en d'indice.

* les balises suivantes sont plutôt des abréviations :

b : lorsque la mise en valeur se fait par la mise en gras de caractères.

i : lorsqu'elle se fait par l'italique; les deux (**b** et **i**) sont cumulables.

* la balise **span** qui est un conteneur inline est utilisé pour les modifications de texte (exemple : texte barré, souligné, surligné, majuscule, minuscule, caractère proportionnellement plus grand ou plus petit, avec les propriétés de feuilles de style **text-decoration** et **text-transform**.).

* **note** : il est fréquent que, dans un texte, on ait à pratiquer une simple mise à la ligne ; on utilise alors la balise simple **
** (ou **
**).

4.4. image.

Il peut paraître curieux que la balise **image** soit présentée dans les balises de texte : c'est que l'image a le même statut qu'un caractère dans le flot HTML. C'est un élément **inline**, pas un **block** ni un conteneur. La forme classique de la balise image est :

```
<img id="monImage"
     class="mesAttributsImage"
     src="belleimage.jpg" />
```

id et **class** sont facultatifs ; la source est obligatoire (son argument est une url, **relative** si l'image est dans le site, ou **absolue** si elle est considérée comme en dehors du site).

L'image de base est le trait horizontal de séparation ; elle est prévue directement dans le langage, c'est la balise simple **hr**, qui est traitée par défaut non pas comme **inline**, mais comme **block** :

<hr />. Cette balise admet évidemment les attributs **id, class, style, title**.

Il existe une autre façon d'afficher une image, théoriquement à réserver aux formulaires, à l'aide d'une balise "de formulaires", au sein desquels elle a un fonctionnement spécifique. La balise fonctionne fort bien hors formulaire, et a même l'avantage d'échapper aux fioritures d'I.E., comme l'apparition d'une barre de traitement de l'image lors du survol d'une image affichée avec la balise "normale".

```
<input type="image"
       id="monImage"
       class="mesAttributsImage"
       src="belleimage.jpg" />
```

On reviendra sur cette façon de procéder assez rare dans le mapping des images (fiche 15).

5. Les ancres.

Le HTML fonctionne sur le mode hypertexte : certains éléments ont la particularité de pouvoir recevoir le focus et d'être activés (touche "entrée"), ou ce qui revient au même d'être cliqués. Dans ce cas, on associe une adresse (sous forme d'une url) à l'élément HTML : une url est un chemin qui indique comment on peut atteindre une page ou un élément de page ; ici, l'url désigne la page que l'on veut à sélectionner pour sa mise à l'écran lorsqu'on clique sur la balise, et éventuellement indique à partir de quelle ligne l'afficher.

Les balises qui permettent de réaliser ces actions sont appelées **ancres**. Ces balises permettent, soit de se déplacer dans la page, soit d'appeler à l'affichage un élément HTML extérieur. Une balise ancre

ou lien ancré est un **conteneur**, comme la balise **span**, de type **inline**.

note : Il existe aussi des balises cibles, dite aussi balises nommées, qui permettent de repérer des endroits particuliers de la pages (comme si on mettait une étiquette). Mais actuellement, on préfère étiqueter une cible par un identificateur de balise. La balise « nommée » tombe donc en désuétude.

* **balise nommée :**

<a name="monetiquette">. Cette balise est désuète ; on obtient le même effet d'étiquette avec les identificateurs de balise : **<p id="monetiquette" ….**

* **appel d'une balise nommée dans la page :**

```
<a href="#monetiquette">texte de l'appel</a>.
<!-- monetiquette correspond maintenant à la valeur de l'un des attributs « id=... » de la page -->
```

* **appel d'une autre page du site :**

```
<a href="../village/monuments.html">les monuments du village</a>
```

* **appel sur une balise nommée dans une autre page du site :**

```
<a href="../village/monuments.html#eglise"
          >les monuments du village : église</a>.
 <!-- la fin de la balise  href  est écrite sur la deuxième ligne ;
      elle   est accolée au texte  pour permettre l'indentation du code
      mais sans qu'il y ait l'ajout d'un espace involontaire -->
```

* **appel avec une url absolue sur un serveur :**

```
<a href="http://lesite/village/monuments.html#eglise">l'église de WXYZ</a>
```

*** cas d'un appel sur des fichiers du disque (travail hors web) :**

```
<a href="file:///lesite/village/monuments.html#eglise">l'église de WXYZ</a>
```

attention : trois slashes pour **file.**

Il n'y a pas d'autre url possible : soit à partir de la page actuelle, soit absolue. On souhaiterait bien entendu un type d'url qui se définisse à partir de la racine du site (dont on ignore le nom lors de sa création, puisque le nom est affecté par l'hébergeur et peut changer si on change d'hébergeur). Ce n'est malheureusement pas possible.

L'utilisation des attributs est délicate avec les ancres ; dans un premier temps, il vaut mieux les éviter.

6. les tableaux de données.

Il y a trois balises pour définir un tableau de données : **table**, **tr** (row = rang), **td** (data = données). Les données peuvent être des éléments HTML quelconques, mais, **quoi qu'on en dise**, pas des tableaux : l'imbrication des tableaux conduit à des résultats imprévisibles. En pratique, il vaut mieux s'en tenir aux tableaux de données, c'est-à-dire aux tableaux où les cellules contiennent des données littérales et/ou chiffrées, et/ou des images. Mais de toute façon, **le tableau ne peut être considéré comme un élément de mise en page** : utiliser les tableaux pour une mise en page complexe devient rapidement ingérable et donne des résultats graphiques stéréotypés et peu lisibles.

On a déjà rencontré la structure de tableau dans la première page ; la balise **table** peut comporter les attributs universels et c'est un **block** ; quand à la balise **td**, il ne faut pas oublier que c'est elle qui est définie comme conteneur (un peu particulier) et que ce sont ses attributs qui permettent de caractériser les éléments graphiques des cellules. De même, il ne faut pas oublier que tous les **tr** contiennent en droit, le même nombre de **td, sauf en cas de fusion de cellules** (voir ci-desous).

Il existe bien d'autres balises de tableau (**tbody, thead, th, tfoot**). Elles ne sont pas nécessaire dans les travaux courants et on conseille d'utiliser plutôt les feuilles de style).

*** La balise td comporte deux attributs HTML spécifiques : rowspan et colspan.**

La syntaxe est de type **rowspan="3"** ou **colspan="2"**. le premier attribut permet de **fondre** des cellules verticalement (ici 3 cellules) et le second des cellules horizontalement (ici 2 cellules). Dans ce

cas, la règle du même nombre de **tr** inclus est infirmée, mais il faut cependant que le compte global y soit (nombre de balises déclarées + nombre de balises fondues).

Exemple :

Voici un tableau avec fusion de cellules ; on examinera bien les balises **td** pour chaque balise **tr**.

<table>
<tr><td>AAA</td><td rowspan="3">BBB</td><td>CCC</td><td>DDD</td><td>EEE</td></tr>
<tr><td>FFF</td><td>GGG</td><td colspan="2">HHH</td></tr>
<tr><td>JJJ</td><td>KKK</td><td colspan="2" rowspan="2">LLL</td></tr>
<tr><td>MMM</td><td>NNN</td><td>PPP</td></tr>
</table>

```
<tr>
  <td> AAA</td>
  <td rowspan="3"> BBB</td>
  <td> CCC</td>
  <td> DDD</td>
  <td> EEE</td>
</tr>
<tr>
  <td> FFF</td>
  <td> GGG</td>
  <td colspan="2"> HHH</td>
</tr>
<tr>
  <td> JJJ</td>
  <td> KKK</td>
  <td colspan="2" rowspan="2"> LLL</td>
</tr>
<tr>
  <td> MMM</td>
  <td> NNN</td>
  <td> PPP</td>
</tr>
```

7. les listes.

La liste est une suite d'items, (en général de textes plus ou moins courts ; rien n'empêche que ce soit des liens hypertextes). Ces textes ont tous un rôle identique ; on montre qu'il s'agit d'une énumération en disposant une puce au début de chaque item. La puce peut être soit un symbole d'ordre (1, 2, 3… ou I, II, III… ou A, B C …) soit un symbole graphique (un rond, un carré, un icone).

note : ne pas confondre

* **un icone** : mot d'origine anglaise qui signifie symbole, matériel ou pas ; *Madona, icone des années 90.* En informatique, c'est une petite image symbolique : *l'icone de Java*.

*** une icône** : tableau religieux, réalisé selon un rite et des formes définies, caractéristique de l'Orthodoxie et des Eglises orientales.

Il y a donc deux genre de listes en HTML : les listes numérotées et les listes à puce. Mais cela ne change rien en ce qui concerne les items. La structure est du type :

Liste numérotée :

```
<ol>
  <li>item 1</li>
  <li>item 2</li>
  <li>item 3</li>
</ol>
```

Liste à puce :

```
<ul>
  <li>item 1</li>
  <li>item 2</li>
  <li>item 3</li>
</ul>
```

La balise li est un conteneur, de type block par défaut.

Mis à part les **attributs universels**, il n'y a pas à se préoccuper des autres attributs qui sont **obsolètes**.

8. Et le reste.

Parmi les balises non décrites, il reste la balise **div** principale balise de regroupement (et donc de mise en page : si on définit un attribut de style à 800 pixels pour **div**, tout l'affichage inclus est dans cette colonne de 800 pixels) et les balises se rapportant plus spécifiquement aux formulaires. Ceci sera vu en détail ultérieurement. Mais auparavant, il faut entrer dans le monde des feuilles de style.

On ne saurait trop inciter le lecteur à construire de petits sites pour s'entraîner. La seule chose à respecter est de se limiter aux balises et attributs évoqués dans le présent texte. Si on prend un modèle extérieur, il faut faire attention aux points suivants :

* pas de balises obsolètes, inemployées ou inutiles : **font, frame, frameset, map...**

* pas d'attributs obsolètes : **width, height** pour les images, **align, cellspacing, cellpadding etc.** Limiter l'emploi de **border** dans les tableaux à une existence transitoire ; **border=1** permet de figurer une bordure avant l'établissement de la feuille de style, on le supprime ensuite dans le site de production.

* pas de mélange des genres. La mise en forme définitive est un autre exercice ; donc pas de couleur, d'alignement, de retrait, de dimensionnement de caractères etc. Ceci est un autre problème.

4 feuilles de style

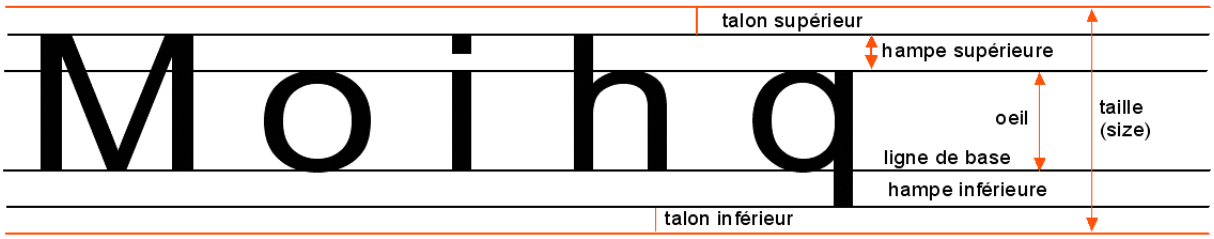
tutoriel HTML

fiche 4

1. les lignes de texte.

1.1. caractères définissant une ligne.

Voici comment se structure une ligne de caractères :



Il s'agit de caractères Lucida Bright. Du point de vue HTML, les données intéressantes sont :

* **la taille du caractère** (font size) : c'est à partir de la hauteur du caractère que se définit la hauteur de la ligne. Par défaut elle lui est égale ; la hauteur de la ligne peut être supérieure à la taille par ajout d'un interligne, ou inférieure, avec risque de chevauchement.

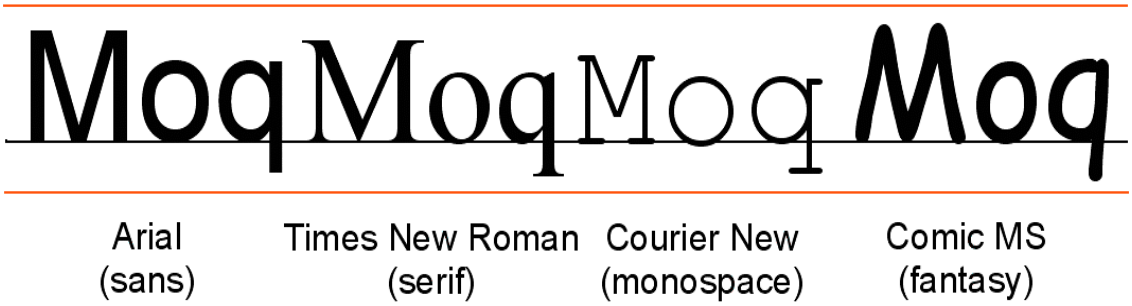
* **la ligne de base** : c'est la ligne qui sert de repère lorsque l'on dispose de fontes différentes ou qu'on mélange caractères et images.

* **l'œil** : sa connaissance a surtout un intérêt esthétique, lorsque l'on est amené à utiliser des fontes différentes sur une même ligne.

On remarquera que les arrondis ne respectent pas bien les limites des lignes de référence ; ceci est une compensation de l'effet d'optique qui fait paraître décalé un arrondi posé sur une ligne de référence. Les arrondis de caractères sont toujours un peu plus hauts qu'ils ne le devraient en stricte géométrie.

1.2. des caractères de même taille sur un ligne.

Voici ce que cela donne, et qui n'est pas sans surprise !

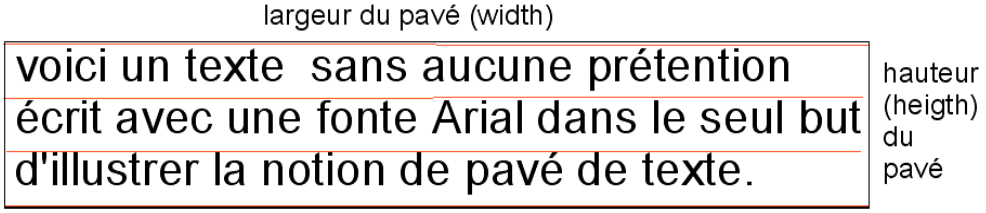


Il s'agit là de quatre familles de fontes régulièrement utilisées et reconnues par le HTML.

1.3. La pavé de texte.

Le pavé de texte est constitué de lignes. Pour simplifier on va supposer qu'il n'y a pas d'interligne,

mais cela ne changerait rien à l'exposé.



1.4. conteneur de texte.

On a déjà rencontré la notion de conteneur; on va en voir ici les caractéristiques. Une image n'est pas un conteneur : une image peut être considéré comme un caractère.

Quelles sont les caractéristiques d'un conteneur de texte ?

* en premier lieu ses dimensions. Mais une fois de plus, Microsoft, voulant échapper aux contraintes du w3c, a eu une attitude qui a varié dans le temps pour définir ce que sont les dimensions d'un conteneur ; il s'est rallié au w3c, mais à condition de le **`dtd`** soit du type **`strict`** ! Les dimensions d'un conteneur sont celle du pavé de texte englobé. Ce n'est pas très naturel, mais c'est la norme.

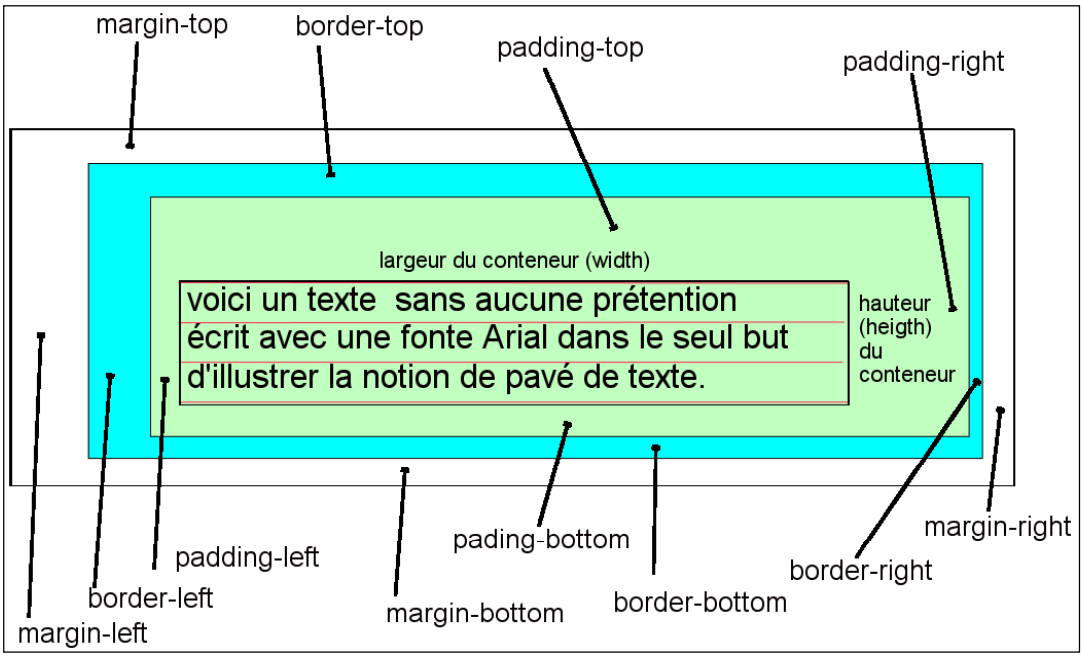
Trois strates entourent le pavé de texte : l'espacement (**`padding`**), la bordure (**`border`**), la marge.

* le **`padding`** est l'espèce d'aération qui entoure le pavé de texte, et évite à l'éventuelle bordure de coller au texte. Il y a donc 4 définitions de padding, une par côté (**`left, top, right, bottom`**). Si le conteneur possède un fond (**`background`**), celui-ci va jusqu'à la bordure et s'applique donc au padding. Malencontreusement, en PAO et traitement de texte, le padding est appelé la *marge*, ce qui peut engendrer des confusions !

* la bordure (**`border`**) qui peut avoir diverses épaisseurs, textures (pointillé, solide, aspect de relief etc), couleur. Il y a donc également quatre définition de bordure.

* la marge peut avoir diverses épaisseurs ; elle est transparente ; elle ne sert qu'à placer le conteneur par rapport à son contenant (page ou autre conteneur). Il y a également 4 définitions de marge.

Un conteneur de texte a un arrière-plan (**`background`**). Dans le meilleur des cas, il est transparent, mais on peut le colorer, ou y définir une image, ou une trame faite d'images. Ce qu'il faut bien voir, c'est que le **`background`** d'un conteneur est fondu avec conteneur et son contenant ; ne pas essayer de geler le **`background`** pour faire défiler son texte dessus ! Même si Microsoft a commis une grossière erreur d'interprétation de la norme qui, sous certaines éditions de I.E. autorisent le texte à glisser sur le fond. (voir fiche 15 pour l'examen du problème).



* nous avons ici un **`background`** vert clair (**`lime`**) , une marge solide et bleue/**`cyan`** ; mais on peut en partie adapter les valeurs à chacun des éléments : une bordure pointillée, l'autre en tiret, avec des couleurs différentes… Le **`padding`** a toujours la couleur de fond et la marge est toujours transparente.

2. L'attribut style et les fontes.

L'attribut HTML **`style` n'est pas à utiliser en production**. Mais au moment de la mise au point, c'est un attribut précieux. Aussi, dans un premier temps, on va utiliser cet attribut pour présenter les feuilles de style, quitte ultérieurement à voir comment transposer en production les concepts vus dans la présente fiche.

2.1. la famille.

* Le premier choix à faire est celui de la famille de fonte. **HTML reconnaît 5 familles de fontes** : **`serif, sans-serif, monospace, fantasy et cursive`**. Etant donné que les polices de fontes sont côté client et que l'on ne peut pas encore importer de fontes dans le fichier HTML (à la différence de PDF ou de Flash), on ne peut savoir avec exactitude quelle fonte sera effectivement disponible ; aussi le w3c a laissé la possibilité de fournir une liste préférentielle et ordonnée de fontes, en la terminant par le nom générique, qui est lui connu par le système d'exploitation.

Voici un exemple de déclaration :

```
font-family:Arial, Helvetica, Verdana, sans-serif;
```

Il est évident que le programmeur est responsable de mettre des fontes de la bonne famille dans la liste. Les trois familles fondamentales sont **`serif, sans`** et **`monospace`**.

-Les fontes à sérifs sont les fontes avec un petit talon, classique en imprimerie ;
-Les fontes sans sérif sont les fontes bâton (ou script) ; ces deux types de fontes sont à espacement variables, c'est-à-dire que chaque caractère a sa propre largeur.
-Les fontes monospace sont des fontes, qui à l'instar de la machine à écrire ou du télétype ont des caractères tous de même largeur. On les utilise pour les textes informatiques car elles permettent l'indentation, et fournissent des sources plus lisibles, surtout si les espaces ont une certaine importance.
-Pour les familles **`fantasy`** et **`cursive`**, les résultats sont très divers dès que la fonte explicite (par exemple *Comic MS*) n'est pas présente sur l'ordinateur.

Fonte Arial, Verdana, sans-sérif
Fonte Times, Garamond, sérif
Fonte Courier, monospace

2.2. La taille.

Il y a deux choix :

font-size la taille du caractère
line-height la hauteur de ligne (qui s'ajuste à la taille si on ne la déclare pas) :

```
font-size: 18px;
line-height: 25px;
```

On étudiera ultérieurement les unités (fiche 14) ; pour l'instant on va utiliser le pixel (px) comme unité.

2.3. La décoration.

Il s'agit du souligné, surligné, normal, barré :

```
text-decoration:underline;
text-decoration:overline ;
text-decoration:normal;
text-decoration:line-through; /* barré */
```

2.4. la graisse et l'italique.

On peut épaissir (ou supprimer cette épaississement), ou mettre en italique, ou les deux.

```
font-weight: bold;
font-weight: normal;
font-style: italic;
font-style: normal;
```

L'attribut peut avoir une valeur numérique.

2.5. alignement dans le conteneur (ou l'écran par défaut) :

```
text-align: center;
text-align: left;
text-align: right;
text-align: justify;
```

2.6. couleur.

On peut choisir la couleur du caractère :

```
color: rgb (100, 15, 80);
```

Il y a **trois façons de d'attribuer une valeur à l'attribut couleur (voir fiche 14)** :

- Soit **la méthode rgb** : on donne la valeur (entre 0 et 255), de la composante rouge, puis verte, puis bleue ; par exemple **`rgb(128,64,0)`**. Le noir correspond à **`rgb(0,0,0)`**, le blanc pur correspond à **`rgb(255,255,255`**) et les trois composantes égales donnent un gris.

- Il est souvent pratique pour les mises au point de donner **un nom de couleur** ; le choix est important (215 nuances) mais seuls les fondamentaux sont utilisés : **`white, black, blue, red, green, yellow, lime(vert), cyan, fuschia(magenta)`**. Les autres couleurs proviennent de la définition par Netscape des "couleurs fixes" qui a servi de référence à l'époque des débuts de la technologie couleur.

- Le troisième mode, dit hexadécimal sera étudié ultérieurement (fiche 14). Exemple de code hexadécimal : #a08000;

Il existe d'autres propriétés, comme la possibilité de jouer sur les majuscules, les minuscules etc (propriété **`transform`**). Ou de tenir compte des espaces multiples. Voir les documents plus spécialisés sur ces propriétés peu employées.

3. questions de conteneur.

3.1. la taille (uniquement pour les blocs).

```
width:400px;
height:300px;
```

3.2. le padding.

On peut définir un padding égal pour les quatre côtés, on spécifier pour chacun:

```
padding:15px;
padding-left:30px;
```

3.3. la bordure.

Il y a trois paramètres à fixer : la largeur, la nature et la couleur :

```
border-style: solid;
border-color: rgb (1000,50,0);
border-width:15px;
bord-top-style:dashed;
```

Pour plus de détails, voir le site **selfhtml**.

3.4. la marge.

Il y a un paramètre à fixer , la largeur :

```
margin:50px;
margin-left:20px;
```

La marge est transparente ; ma marge gauche est définie avant la marge droite. Pour un conteneur de largeur définie, faire : margin-left:auto;margin-right:10px;

et pour le centrer : **margin-left:auto;margin-right:auto;**

Les marges peuvent être négatives, et les conteneurs se superposer ! Attention : le conteneur **body** a une marge supérieure.

3.5. le fond.

On peut désirer un fond transparent, coloré, avec une image (ou une trame d'images).

```
background-color: rgb (100,255,0);
background-image: url(monimage.gif);
```

En principe, l'image est répétée, mais il existe la possibilité de ne pas la répéter, ou de la répéter uniquement suivant une direction (X ou Y) Voir le site **selfhtml** pour les détails ; les applications de la répétition sont délicates et très spécifiques. (cf. la fiche **10**).

4. L'attribut style.

4.1. feuille de style.

Une feuille de style est une succession de couples propriété/valeurs ; le point-virgule final est obligatoire. L'ordre est sans importance. En cas d'erreur dans une feuille de style, la lecture s'arrête sur l'erreur et le début est pris en charge normalement. Pour illustrer ceci, le plus simple est de reprendre l'exemple de la fiche 2, dans laquelle on s'impose d'utiliser une seule fonte, une fonte bâton, et de bien dominer la hauteur des caractères.

4.2. héritage.

On a ajouté un conteneur global, **div**. Pour l'instant, il faut savoir que la définition d'une feuille de style pour un conteneur, fait hériter son contenu de tout ce qui est héritable : fontes, dimensions de caractères, couleur etc. Mais bien évidemment pas de ce qui relève du conteneur en tant que tel : dimensions, bordure, padding etc. Le conteneur **div** permet d'afficher sur une colonne margée au lieu de tout l'écran, constituant ainsi une fenêtre d'affichage.

Attention : pour le texte inclus dans les ancres, la décoration par défaut, est le souligné ; si on n'aime pas, il faut explicitement évoquer une décoration **normal**, c'est-à-dire l'absence de toute décoration. Il faut ajouter que les couleurs et le fond sont aussi prédéfinies et n'héritent pas du conteneur éventuel contenant le lien ancré. En pratique si on a le schéma :

```
<div ....>
  <a ....>
    texte
  </a>
<div>
```

le texte hérite de la fonte de div, mais pas de son fond, de sa couleur de caractère, de sa décoration.

4.3. priorités.

Dans un premier temps, il faut avoir présentes à l'esprit quelques règles **pratiques** :

règle 1. Toute balise a une feuille de style par défaut (fixée par le navigateur). Si une propriété n'est pas redéfinie, c'est la propriété par défaut qui est utilisée. Ainsi, la balise **p** a des marges prédéfinies ; elles ne sont pas redéfinies dans l'exercice, ce qui conduit à un aspect légèrement différent selon les navigateurs.

règle 2. S'il y a conflit entre deux propriétés, c'est toujours la dernière qui a été définie qui est prise en compte. Ainsi, le **font-family:sans-serif;** s'applique à toute la page; mais les **font-size** sont redéfinis, de même que les **font-weight**. La règle sous cette forme est un bon aide-mémoire ; cependant il faut la moduler par la règle de priorité suivante (qui ne prendra tout son sens que ultérieurement) :

règle 3. Si on a une clause de style définie par **style**, elle a priorité sur celle définie par **id**, qui elle, a priorité sur celle définie par **class**. Un clause de style définie par **style, id, class** est postérieure à celle définie pour une balise.

4.4. exemple : le code de `html04tp0.html`.

```
<body>
  <div style="width:800px;margin-left:100px;font-family:sans-serif;">
    <h1 style="size:26px;font-weight:bold;"
      >le codage des images pour le web</h1>
      <blockquote id="napoleon" style="size:20px;font-style:italic;"
        >" Un bon croquis vaut mieux qu'un long discours. "
        <span style="font-style:normal;">NAPOLÉON</span>
      </blockquote>
      <h2 style="size:22px;font-weight:bold;">1. Contraintes du codage.</h2>
      <p class="normal" style="size:20px;">
        Une image informatique peut être représentée de deux façons :
      </p>
      <p class="retrait" style="size:20px;margin-left:20px;"> Soit on fait un
programme qui dit comment dessiner l'image : dessiner sur le canevas un
rectangle avec telles et telles caractéristiques, un cercle... un pavé de
texte...C'est la méthode vectorielle ; pour le web, cette démarche n'est pas
généralisée actuellement (avec l'avènement du SVG, format qui reste actuellement
marginal, cela va changer).
      </p>
      <p class="retrait" style="size:20px;margin-left:20px;"> Soit on décrit
l'image comme un tableau de données ; l'atome de ce tableau est le pixel. Une
webcam fabrique une image de 640 pixels de large et 480 pixels de haut, soit
307200 pixels. En monochrome, le pixel est codé sur un octet et autorise dont
256 nuances ; en couleur, sur trois octets, et autorise ainsi 1 677 216 nuances.
Mais cette petite image, considérée du point de vue du stockage ou de la
transmission nécessite 300 kO en monochrome, 1 MO en couleur.
      </p>
      <h4 id="inacceptable" style="font-style:italic; color:rgb(180,0,0);">
```

```
        C'est inacceptable en l'état pour l'utilisation sur le web !</h4>
      <p class="normal" style="size:20px;"> D'où la nécessité de compresser les
images. Ce n'est pas une exigence spécifique au web, mais le web correspond à un
cas d'espèce. En infographie on peut accepter des temps de compression  élevés,
mais aucune perte de qualité ; si l'imprimeur demande un fichier pdf pour son
tirage, on fait une sauvegarde à 200%. Dans le web, le temps de décompression
doit être faible, l'information à transmettre faible également, mais on peut
accepter une certaine perte de qualité de l'image.
      </p>
      <h2 style="size:22px;font-weight:bold;">2. Les choix actuels.</h2>
      <p class="normal" style="size:20px;"> Il y a trois modes de compression
utilisées sur le web :
      </p>
```

* On n'a reproduit que le début du corps de texte. Le fichier entier est **`html04tp0.html`**.

* Attention : ceci n'est pas un modèle de production, mais représente une démarche d'expérimentation : l'attribut style n'est jamais présent en production !

* la balise **`span`** : c'est une balise **`inline`** (elle ne modifie pas la succession des éléments du flot HTML, par exemple en provoquant un saut à la ligne). Elle permet les ajustements de détail. La règle d'usage est qu'aucune balise n'est incluse dans une balise **`span`** (c'est le plus petit conteneur).

Exercice : en guise d'exercice, on peut imposer les contraintes suivantes :

* dominer plus finement les interparagraphes par utilisation du margin-top et du margin-bottom dans les feuilles de style. On rappelle qu'un pixels mesure environ 0,23 mm (pitch) sur un écran normal.

* jouer avec la couleur pour les titres ou sous-titres.

note. On évitera de trop toucher au tableau qui sera examiné ultérieurement..

On n'a pas vu où déclarer les feuilles de style, ni les balises de début et de fin de déclaration des styles, ni comment écrire des commentaires dans la déclaration des styles... C'est l'objet de la fiche qui suit.

5 utilisation des feuilles de style

tutoriel HTML

fiche 5

introduction.

Dans la fiche précédente, on a vu comment établir des feuilles de style (relatives aux textes et conteneurs) et comment les utiliser dans l'attribut **`style`** des balises ouvrantes ou simples. On rappelle une fois de plus que c'est une méthode qui sert dans le débugage, pas en production. On va examiner dans cette fiche les trois méthodes réellement utiles en production.

Les feuilles de styles peuvent

- soit se trouver dans le document html
- soit appartenir à un fichier séparé qui ne contient que des feiuilles de style

On va donc voir comment passer de l'un à l'autre et les avantages du fichier séparé. Dans les deux cas, un système de balisage dans l'en-tête du fichier HTML signale que l'on définit des feuilles de style. On n'est pas obligé de situer le balisage dans l'en-tête : mais un élément de style ne peut être utilisé que s'il a été défini antérieurement ; sinon il est ignoré. Il n'y a de prise en charge d'un élément de style que quand il est rencontré lors de l'analyse du fichier HTML. Ce qui fait qu'il n'est pas interdit de changer un élément de styles dans le cours d'un fichier HTML ; ce n'est évidemment pas une méthode recommandée.

1. la première méthode : redéfinition d'une balise.

1.1. feuille de style par défaut.

On a vu que toute balise possède une feuille de style par défaut, définie dans le navigateur, avec les aléas que cela représente quand on change de navigateur. Cette définition par défaut peut présenter des avantages :

* par exemple, les conteneurs **`h1, h2, …h6, p, div`** ont un affichage de type **`block`** par défaut, ce qui correspond à l'usage habituel : ce sont en effet des blocs, qui se succèdent séquentiellement dans le flot HTML. Il est donc inutile de rappeler à chaque fois la propriété : **`display:block;`**

* de même les conteneurs **`a`** (ancre), **`span`**, **`em`**, **`strong`**, l'image **`img`** sont du genre **`inline`** et c'est leur usage usuel : inutile de rappeler la propriété **`display:inline;`**

* il existe des balises comme **`b`** (gras) ou **`i`** (italique), qu'on ne redéfinit pas car elles ne sont que des équivalences, des raccourcis, de **`<span style="text-weight:bold">`** ou **`<span style="text-style:italic;">`**

1.2. redéfinition.

Malheureusement, si on veut dominer rigoureusement la présentation de la page, les éléments à redéfinir sont nombreux. Ce qui conduit à des reprises fastidieuses mais nécessaires. Pour redéfinir une balise, on énonce son nom et on place les propriété entre accolades.

Exemple.

```
<style type="text/css>
h1 {
  font-family:Arial, Verdana, sans-serif;
  margin-top: 10px ;
  marging-bottom:10px;
  font-size: 28px;
  line-height: 35px;
  padding-top: 6px;
```

```
  padding-bottom: 6px;
}
... autres déclarations ...
</style>
```

On n'a pas repris le **font-weight:bold; color:black; margin-left:0;** etc qui conviennent dans leur valeur par défaut ; par contre les marges haut et bas sont dépendantes du navigateur et il faut les redéfinir.

note : il n'y a pas d'unité si la valeur est nulle ; il y en a nécessairement une sinon, contrairement à ce qui se passe en javascript.

Toutes les définitions de feuilles de style peuvent être faites dans la même **balise `style`** ; ce n'est pas une obligation.

2. la deuxième méthode : cas d'une balise identifiée.

On a vu comment identifier une balise : **`id="mabalise"`**.

Rappels : la casse de la valeur est prise en compte, **`mabalise`** est différent de **`maBalise`**. Et il faut utiliser l'ASKI basique (lettres, chiffres, souligné) dans les identificateurs si on veut éviter des déboires. Même le souligné (underscore _) est à éviter.

On peut créer une feuille de style qui ne fonctionne qu'avec une balise particulière ; il suffit de nommer la feuille de style non plus avec le nom de la balise comme précédemment mais à l'aide d'un # suivi du nom de la balise. Attention : la feuille de style ne peut fonctionner que pour une seule balise, celle qui a le bon identificateur.

exemple.

```
<style type="text/css>
#exceptionnel {
  font-weight:bold;
  color:red;
  font-style:italic;
  font-size:22px;
  background-color:yellow;
}

….. autres déclarations ….
</style>
```

On a là affaire à un segment de texte un peu spécial, qui va déroger par la couleur, le fond etc… tout en prenant les autres éléments dans le balisage d'inclusion :

Exemple :

```
<p>Il faudra prêter <span id="exceptionnel">une attention spéciale</span>
aux recommandations qui suivent :</p>
```

3. La méthode courante : cas des classes.

Dans l'exemple traité en fiche 4, on a pu constater que certaines caractéristiques de style se reproduisent à l'identique d'une balise à l'autre : on dit dans ce cas qu'elles représentent une classe.

Une classe se définit avec nom précédé d'un point ; l'activation de la classe se fait sous la forme **`class="`***`nom de la classe`***`"`** On utilise la classe autant de fois qu'on le veut, dans autant de balises différentes que l'on veut ; les caractères antérieurement définis dans ces balises restent valides, sauf ceux qui sont surchargés, redéfinis par la classe utilisée. En cas de conflit, il faut se rappeler qu'en principe, la dernière définition est prise en compte, supplantant les propriétés de même nom définies auparavant.

4. un exemple.

On se propose de reprendre l'exemple du début, en définissant utilement des feuilles de style (redéfinition de feuille de style pour les balises, méthode par identificateur, méthode par classe).

le code.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
          "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
    <title> html05corr
    </title>
    <style type="text/css">
    div { /* il n'y a qu'un div dans le fichier */
      width:900px;
      margin-left:100px;
      font-family:sans-serif; /* valeur héritable */
      border:2px;
      border-style:dashed;
      border-color:green;
      padding:10px;
    }

    h1 { /* hérite de div pour la fonte, comme les suivantes */
      font-size:28px;
      margin-top:10px;
      margin-bottom:10px;
      text-align:center;
    }

    h2 {
      font-size:24px;
      margin-top:6px;
      margin-bottom:6px;

    }
    h3 {
     font-size:22px;
      margin-top:3px;
      margin-bottom:3px;
      text-decoration:underline;
    }

    p {
      font-size:20px;
      margin-top:3px;
      margin-bottom:3px;
```

```
    padding-top:3px;
    padding-bottom:3px;
    text-align:justify;
  }


  blockquote {
    font-size:20px;
    font-style:italic;
  }

  #auteur {
    font-style:normal;
  }
  .retrait {
    margin-left:20px;
  }

  #inacceptable {
    font-style:italic;
    font-size:20px;
    color:rgb(180,0,0);
    font-weight:bold;
    margin-top:3px;
    margin-bottom:3px;
    text-align:center;
  }

  table {
    border-collapse:collapse; /* les bordure sont superposées*/
  }

  td {
    border:1px;
    border-color:blue;
    border-style:solid;
    padding:10px;
    font-size:18px;
  }
  .titre {
    font-weight:bold;
  }

  </style>

</head>
<body>
  <div>
    <h1>le codage de images pour le web</h1>
```

```
      <blockquote>" Un bon croquis vaut mieux qu'un long discours. "
            <span id="auteur">NAPOLÉON</span>
      </blockquote>
      <h2>1. Contraintes du codage.</h2>
      <p class="normal"> Une image informatique peut être représentée de deux
façons :
      </p>
      <p class="retrait"> Soit on fait un programme qui dit comment dessiner
l'image : dessiner sur le canevas un rectangle avec telles et telles
caractéristiques, un cercle... un pavé de texte...C'est la méthode vectorielle ;
pour le web, cette démarche n'est pas généralisée actuellement (avec l'avènement
du SVG, format qui reste actuellement marginal, cela va changer).
      </p>
      <p class="retrait"> Soit on décrit l'image comme un tableau de données ;
l'atome de ce tableau est le pixel. Une webcam fabrique une image de 640 pixels
de large et 480 pixels de haut, soit 307200 pixels. En monochrome, le pixel est
codé sur un octet et autorise dont 256 nuances ; en couleur, sur trois octets,
et autorise ainsi 1 677 216 nuances. Mais cette petite image, considérée du
point de vue du stockage ou de la transmission nécessite 300 kO en monochrome, 1
MO en couleur.
      </p>
      <h4 id="inacceptable">
        C'est inacceptable en l'état pour l'utilisation sur le web !</h4>
      <p> D'où la nécessité de compresser les images. Ce n'est pas une exigence
spécifique au web, mais le web correspond à un cas d'espèce. En infographie on
peut accepter des temps de compression  élevés, mais aucune perte de qualité ;
si l'imprimeur demande un fichier pdf pour son tirage, on fait une sauvegarde à
200%. Dans le web, le temps de décompression doit être faible, l'information à
transmettre faible également, mais on peut accepter une certaine perte de
qualité de l'image.
      </p>
      <h2>2. Les choix actuels.</h2>
      <p > Il y a trois modes de compression utilisées sur le web :
      </p>
      <p class="retrait"> * la méthode par palette (gif, png-palette).
      </p>
      <p class="retrait"> * la méthode de la compression de fichier sans perte
(gif, png).
      </p>
      <p class="retrait"> * la méthode par représentation mathématique de pavés
64x64 pixels (jpeg).
      </p>
      <p> Ces trois modes seront étudiés dans les pages qui suivent.
      </p>
      <h3> 2.1. principe de la méthode par palette (format gif ou png-
palette).</h3>
      <p > On choisit en général une gamme de 256 teintes numérotées de 0 à 255,
et que l'on code sur trois octets. Chaque pixel est ensuite codé par un octet,
qui est le numéro de la teinte la plus approchante. Il y a évidemment perte
d'informations, mais quasi indécelable dans l'affichage sur écran. On gagne un
facteur 3 dans la taille des fichiers. D'autre choix de dimension de palette
sont possibles (16 ou même moins).
      </p>
      <h3>2.2. principe de la compression sans perte.</h3>
```

```
      <p>
        Il existe de nombreux algorithmes mathématiques qui utilisent le fait
qu'un fichier d'octets qui code un texte ou une image n'est pas aléatoire, mais
présente des régularités que l'on peut exploiter pour le compresser. Ce procédé
est utilisé dans le png normal (mais aussi dans les format TIFF, PS, PCX, PDF),
et permet encore un léger gain dans les systèmes à palette. Une compression
jusqu'à 10 est possible.<br /> Mais en général la performance est plus faible
(cela dépend de l'image).
      </p>
      <h3>2.3. principe des algorithmes mathématiques.</h3>
      <p> Le principe est semblables à la décomposition de Fresnel pour les
courbes ; mathématiquement complexe, le jpeg autorise le choix du taux de
compressions, aux dépens de la qualité d'image. On code par pavés de 4096 pixels
; il apparaît assez rapidement des artefacts (traces dues à la manière de faire
le code).
      </p>
      <h2>3. Voici une idée des résultats pour une photo couleur de fleurs de
816 x 612 =499 392 pixels</h2>
      <table>
        <tr>
          <td class="titre">type</td>  /* titre est prioritaire */
          <td>brut</td>
          <td >png 24b<br />non cp</td>
          <td >png pal<br />non cp</td>
          <td >png 24b<br />cp max</td>
          <td >png pal<br />cp max</td>
          <td >gif</td>
          <td >jpeg<br />non cp</td>
          <td >jpeg<br />cp 10</td>
          <td>jpeg<br />cp 30</td>
          <td >jpeg<br />cp 50</td>
          <td >jpeg<br />cp 80</td>
        </tr>
        <tr><
          td class="titre">k-octets</td>
          <td >1498</td>
          <td > 1467</td>
          <td >490</td>
          <td >1189</td>
          <td >420</td>
          <td >420</td>
          <td >691</td>
          <td >346</td>
          <td >183</td>
          <td >118</td>
          <td >78 </td>
        </tr>
        <tr >
          <td class="titre">qualité</td>
          <td > </td>
          <td>max</td>
```

```
          <td>exc</td>
          <td>max</td>
          <td>exc</td>
          <td>exc</td>
          <td>max</td>
          <td>exc</td>
          <td>tb</td>
          <td>limite</td>
          <td>mauvais</td>
        </tr>
      </table>
      </div>
  </body>
</html>
```

5 fichier externe.

5.1. Syntaxe.

Les feuilles de style peuvent être regroupées en un fichier externe au fichier HTML. Il suffit pour cela d'assurer la liaison entre le fichier HTML et le fichier externe par une balise **link**, dans la partie **head** du fichier HTML.

Exemple.

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

* l'url du fichier est en principe une url relative, c'est-à-dire le chemin relatif entre le fichier html et le fichier de feuilles de style ; exemple : "**../../repCSS/mafeuille.css**" . En effet on place habituellement les fichiers de style sur le même site que les fichiers HTML. Ce peut être aussi un chemin absolu (commençant par **http://**), par exemple si on emprunte un fichier à un autre site, ou encore, si fait le choix des url absolues pour toutes les url du site (c'est utile dans les sites « dynamiques », en association avec le php) .

* Un même fichier HTML peut faire appel à plusieurs fichiers contenant des feuilles de style ; en cas de conflit pour une classe, un identificateur, une redéfinition, c'est la propriété déclarée en dernier lieu est prise en compte.

* le nom du fichier contenant les feuilles de style, doit avoir l'extension .css

Il existe une seconde façon pratiquement équivalente d'introduire un fichier de style ; la syntaxe est :

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

5.2. cache.

Une feuille de style chargée n'est pas incluse dans le fichier HTML : elle lui est simplement liée. Ce n'est pas comme en php où une partie incluse fait partie du fichier qui l'appelle.

Ceci présente un gros avantage : les navigateurs mettent en **mémoire cache** tout ce qu'ils chargent et ne rechargent pas les fichiers qui ont les mêmes références qu'un fichier en cache (même situation sur le serveur, même date).

Cela peut quelquefois présenter des inconvénients lors du débogage (il faut parfois vider le cache avant d'exécuter le fichier). Mais en production, il n'y a que des avantages : le même fichier CSS qui sert pour des pages HTML distinctes d'un même site n'est chargé qu'une fois. On a donc tout intérêt, lors de l'établissement d'un site à "factoriser" les fichiers de style : cela fait gagner en cohérence, puisqu'on est assuré que c'est le même fichier qui est utilisé, et en efficacité puisque le fichier n'est

chargé qu'une fois.

6 questions d'image

tutoriel HTML

fiche 6

introduction.

Le HTML, en attendant mieux, prend en charge l'affichage de trois formats d'images : le format gif, le format jpeg et les formats png. Ce sont des formats bitmap. Il n'y a pas pour l'instant de format vectoriel autorisé : le format SVG qui devrait devenir la norme n'existe que sur quelques navigateurs (par exemple Firefox ; pour programmer en SVG, voir le logiciel Amaya). Après une discussion sommaire sur chacun de ces formats, le sujet portera sur l'utilisation des images en HTML.

1. les formats.

1.1. tableau de pixels.

Une image bitmap est un tableau de pixels (pixel = élément d'image, point constitutif de l'image) : chaque pixel de l'image est représenté par une unité du tableau. Dans sa forme la plus élémentaire, un élément du tableau comporte trois octets, un pour l'intensité de la composante rouge (**`red`**), un pour celui de la composante verte (**`green`**, mais appelé **`lime`** dans le web), un pour celui de la composante bleue (**`blue`**). Comme on peut donner 256 valeurs à l'intensité lumineuse d'une couleur de base (0 à 255), il y a donc 256x256x256 nuances possibles, ce qui est largement suffisant pour une image sur écran (ou imprimée). Mais c'est prohibitif du point de vue de la transmission de l'information, comme cela l'était pour l'enregistrement, au temps où le stockage de l'information était cher. Pour une image monochrome, un seul octet est nécessaire, ce qui ne divise que par 3 le poids des fichiers. La nécessité de compresser l'image est donc apparu très vite en informatique.

1.2. le format couleur gif.

La première compression utilisée sur le web a été le format gif : reprenant une méthode déjà utilisée pour les écrans couleur (le VGA), le principe est

- de se fixer une palette de couleur (une gamme de 256 teintes) qui est une espèce de tableau où chaque teinte est représentée sur trois octets (couleur vraie),
- de référencer chacune des teintes de la palette par un numéro sur un octet (index du tableau, allant de 0 à 255),
- d'affecter à chaque pixel un numéro de la palette.

On y gagne un facteur 3 ; il n'y a pas de flou dans l'image, même si la couleur accordée à chaque pixel n'est qu'approchante. On peut améliorer l'approximation, par exemple en choisissant pour la palette les couleurs les plus fréquentes sur l'image Dans certains cas, au contraire, il faut se fixer une palette de référence, qui reste la même pour toute une famille d'images, comme par exemple quand on traite de graphisme « au trait » (dessins d'architectes, organigrammes)…

On peut ensuite compresser le tableau par une méthode analogue à la méthode ZIP ; le gain reste faible, sauf pour les graphismes au trait.

Malheureusement le format gif reste un format propriétaire et les éditeurs de logiciels doivent payer pour en utiliser les algorithmes : le format gif s'exclut ainsi du monde du libre ou du gratuit (et donc du php et du Java par exemple).

1.3. Le format JPEG.

Le format JPEG est un format de compression basé sur des algorithmes mathématiques complexes. La seule chose à retenir est que le codage se fait par carrés de 64x64 pixels, et que lors de fortes compressions, il peut y avoir des artefacts (formes incongrues, sans rapport avec l'image) qui apparaissent, du plus mauvais effet car ils perturbent en particulier les a-plats monochromes. Ce n'est pas un format pour les images détourées, les graphique au trait etc.

La compression peut être très forte, mais au prix d'une perte d'information. La compression "sans

perte" (qualité photo) correspond à un facteur de réduction de 10 du fichier, ce qui est loin d'être négligeable (c'est la compression utilisée dans les appareils photos numériques). La compression est échelonnée de 0 à 100 (cela ne correspond pas à un facteur réel de réduction du fichier). Pour mémoire, un code 30 correspond à un maintien de qualité acceptable sur le web et un gain d'espace substantiel (qui dépend des images). Au delà, la qualité est plus aléatoire ; elle dépend beaucoup de la nature des images : un graphisme au trait supporte très mal la compression alors que la photo d'un bouquet fleuri la supporte bien.

1.4. Les formats PNG.

Le but des format PNG est double : assurer une qualité maximale à l'image ; fonctionner dans le monde du logiciel libre. C'est malheureusement incompatible avec une compression importante. Il existe deux formats png : le format à palette et le format de compression sans perte (dit aussi par compression d'octets, utilisée en ZIP, que l'on retrouve dans l'ancien PCX, le format TIFF etc).

Le format à palette est de performances identiques au format GIF : il faut le réserver aux graphismes (cartes, plans) où il est très performant.

Le format en vraies couleur "compressé" conduit à une excellente qualité, sans perte d'information, mais au prix de fichiers qui restent conséquents, comparés à ceux du format jpeg.

1.5. transparence.

Les images à palette peuvent disposer d'une nuance transparente (et une seule). La transparence est liée au fichier image. Il ne faut pas confondre la transparence et le facteur alpha qui a une toute autre signification, même si il joue également sur la transparence (voir plus loin)…

2. Les images en HTML.

On distingue les images de fond (background) et les images normales.

Il y a deux types d'images de fond : celles de la page et celles des conteneurs ; les propriétés de feuilles de style et leur comportement peuvent différer.

Il y a deux types d'images normales : celles des images d'illustration et celles des formulaires.

2.1. Images de fond.

* Une image de fond est toujours affichée en vraie grandeur.

```
background-image:url(url de l'image);
```

* elle peut être répétée selon les deux axes (x, y) et c'est la valeur par défaut. Les commandes de répétitions sont :

```
backgroung-repeat: repeat;
backgroung-repeat:repeat-x;
backgroung-repeat:repeat-y;
backgroung-repeat:no-repeat;
```

* la première image peut être positionnée dans la page :

background-position : top;

background-position: center ;

background-position: bottom;

background-position: middle ;

background-position: left;

background-position: right;

background-position: 20px 40px;

Attention à la syntaxe pour la dernière commande, il n'y a pas de séparateur autre que l'espace !

*** pour la page uniquement !**

```
background-attachment:scroll;
background-attachment:fixed;
```

Cette commande permet le défilement du contenu en même temps que le fond (par défaut ; mode **scroll**) ou au contraire la fixation de l'image de fond lorsque la page défile. Cet élément de style ne doit pas s'appliquer aux conteneurs (w3c), même si Microsoft le fait (à tort) dans certains de ses navigateurs.

2.2. Image normale.

Il y a deux façons d'afficher une image :

* **la façon « naturelle » :**

```
<img id="monimage" src="url de l'image" />
```

* **la façon "input" :**

Contrairement à ce qui est écrit un peu partout, cette façon de faire fonctionne hors formulaire ; un peu aberrante, elle est la seule façon d'échapper aux fioritures apportées aux images par I.E. (apparition d'une barre de propriétés que l'on ne peut faire disparaître qu'en modifiant le paramétrage du navigateur)

```
<input type="image"  id="monimage" src="url de l'image" />
```

Les propriétés de style sont les mêmes pour les deux balises (voir la fiche 15).

3. propriétés de style liées aux images normales.

* dimensions : les dimensions peuvent être définies par les commandes **width** et **height** ;

* l'image n'est pas un conteneur.

* c'est un élément **inline**.

* La propriété **padding** n'existe pas pour une image.

* on peut cependant définir la propriété de bordure (**border**) et de marge (**margin**) comme pour les conteneurs. On peut également définir une image comme flottante (**float**) et la positionner de façon absolue (**position: absolute**) exactement comme on le fait d'un type **block**. Cette dernière propriété, importante, sera étudiée plus précisément avec le positionnement.

L'image, comme élément inline peut être ajustée dans un conteneur de mêmes dimensions que l'image (**div, p,** etc) ; on peut alors centrer le **conteneur** par **margin-left:auto; margin-right:auto;** où le caler sur le bord, par exemple par **margin-left:auto; margin-right:5px**; il faut que les deux éléments de marge soient présents, sinon par défaut, **margin-left** vaut 0, et il s'impose prioritairement.

* **vertical-align** : comme une image est un élément **inline**, il est possible de la placer relativement à la ligne de base actuelle. Les valeurs suivantes sont reconnues :

- **top** = aligner en haut.
- **middle** = aligner au milieu.
- **bottom** = aligner en bas.
- **baseline** = aligner sur la ligne de base (ou en bas s'il n'y a pas de ligne de base).
- **sub** = mettre en indice.
- **super** = mettre en exposant.
- **text-top** = aligner sur le bord supérieur de l'écriture.
- **text-bottom** = aligner sur le bord inférieur de l'écriture.

L'usage de ces propriétés est rare ; on peut en avoir besoin par exemple si on veut simuler par une image un caractère exotique (math, grec archaïque, araméen...) dans un contexte latin ; il fut un temps où c'était la principale façon d'insérer une expression grecque, arabe ou hébreu en ligne, quand les caractères dans ces langues n'étaient pas accessibles (rappel : les entités HTML).

Exemple. voici un tableau d'images qui pourrait compléter l'exemple de la fiche 2 ; pour bien faire apparaître la qualité, on a affiché les image à l'échelle 1,5. Fichier : **html06tp0.html**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
                    "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
    <title>
      images
    </title>
<style type="text/css">
.dim {
  width:343px;
  height:341px;
  border: 2px;
  border-style: solid;
  border-color:black;
}

td {
  font-family:Arial;
  border: 2px;
  border-style: solid;
  border-color:black;
  padding: 10px;
  text-align: center;
  font-size: 20px;
}
</style>
  </head>
  <body>
  <h2>images à l'échelle 150%</h2>
    <table>
      <tr>
        <td> <img class="dim" src="_aster.png" /><br/>
               image png normale<br/>127 kO
        </td>
        <td><img class="dim" src="_asterpal.png" /><br/>
               image palette png<br/>47 kO</td>
      </tr>
      <tr>
        <td><img class="dim" src="_aster30.jpg" /><br/>
               image jpeg 30<br/>15 kO</td>
        <td><img class="dim" src="_aster99.jpg" /><br/>
                imagejpeg 90<br/>3 kO</td>
      </tr>
```

```
    </table>
   </h2>
  </body>
</html>
```

7 affichages

tutoriel HTML

fiche 7

introduction.

L'affichage classique respecte les caractéristiques suivantes :

> * les éléments de type **block** sont affichés les uns à la suite des autres, constituant le flot HTML. Essentiellement, on trouve dans cette catégorie las balises **h1, h2 …h6, p, table, form, ul, uo, li, div.**
>
> * les éléments de type **inline** sont affichés côte à côte, constituant des lignes ; lorsque la place manque (bord de l'écran, bord d'un conteneur), il y a passage automatique à la ligne (sauf exception comme avec la balise **pre**). Le passage à la ligne (**wrapping**) se fait sur un espace ou une image (toujours assimilable à un caractère). On trouve dans cette catégorie **span, a, b, i, em, strong, img**...et les balises **input** de formulaire.

Il est possible de modifier les modes d'affichage ; c'est l'objet de cette fiche.

1. le mode display.

1.1. changement de mode.

Lorsqu'un élément est d'un type d'affichage **block** ou **inline**, et qu'il est affiché en tant que tel, il est inutile de le préciser. Mais il se peut que l'on veuille changer le mode d'affichage. Le cas d'école est la liste ; normalement les items d'une liste sont de type **block** et s'affichent donc l'un au-dessous de l'autre. Mais on peut souhaiter disposer des balises de liste (**li**) et les afficher horizontalement ; on impose alors le mode **inline** à l'affichage de la liste.

code et affichage initial.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
          "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
  <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
  <meta name="generator" content="PSPad editor, www.pspad.com">
  <title>inline</title>
  <style type="text/css">
  li {
    border:1px;
    border-style:solid;
    border-color:blue;
    width:250px;
    height:35px;
    font-family:sans-serif;
    font-size:22px;
    padding:5px;

  }
  </style>
  </head>
  <body>
```

```
    <ul>
      <li class="menu">premier item</li>
      <li class="menu">deuxième item</li>
      <li class="menu">troisième item</li>
    </ul>
  </body>
</html>
```

- premier item
- deuxième item
- troisième item

ajout de `display:inline;` à la feuille de style de `li` :

premier item deuxième item troisième item

1.2. les modes principaux.

On vient de voir le passage à **`display:inline;`** Le passage à block est possible (par exemple pour une image). Mais il existe aussi une valeur, **none**, qui efface le bloc, sans aucune réservation d'espace. Dans ce cas, pour récupérer l'affichage, il faut explicitement redéclarer la nature de l'affichage (par un *script* déclenché par un événement par exemple) ; pour mémoire, l'instruction javascript est :

```
document.getElementById("nom du bloc").style.display="block";
```

Le passage de **`inline`** à **`block`** est moins fréquent ; il permet par exemple de placer un paragraphe dans un flot de texte, sans retour à la ligne :

Exemple :

```
<h1> j'insère un paragraphe<p style="display:inline;font-size:20px;">mon paragraphe</p> dans un titre</h1>
```

j'insère un paragraphe mon paragraphe dans un titre

Il permet d'isoler une image du flot de texte où elle est présente. **Fichier exemple** : **`html07tp0.html`**

note : les éléments de type **`block`** peuvent être calés à gauche, à droite, centrés, par rapport à un conteneur éventuel, ou par rapport à l'écran. C'est une question de margin-left ou margin-right ;

Pour centrer, faire :

```
margin-left:auto;
margin-right:auto;
```

Pour caler à droite par exemple :

```
margin-left:auto;
margin-right:0;
```

Ces propriétés remplacent les attributs **`align`** traditionnels des balises **`table, div`** etc..

2. le flottement.

Lorsque l'on dispose d'un conteneur **`block`** de largeur fixée, l'insertion dans le flot HTML suit la règle générale : deux conteneurs déclarés à la suite l'un de l'autre sont affichés l'un derrière l'autre, même s'il y a la place pour les mettre côte à côte. Il y a cependant moyen de rompre l'effet de **`block`** en rendant les éléments flottants.

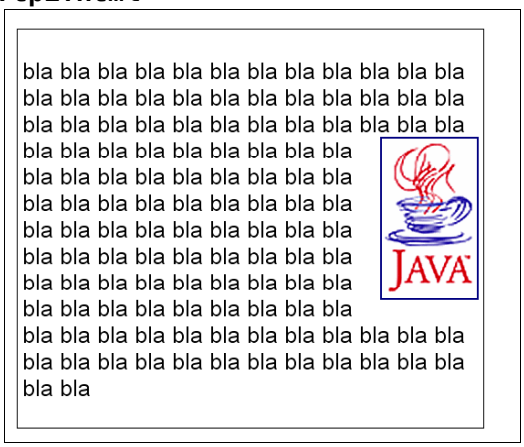
Il y a deux types de flottement : à gauche et à droite. Un élément flottant cherche **la première place d'affichage compatible** avec son argument (**`left`** ou **`right`**). En cas d'absence d'élément gauche ou droite, c'est le conteneur ou l'écran qui sont pris en considération. **Le flot HTML continue à s'enrouler autour de l'élément flottant : celui-ci occupe un espace défini, et les éléments graphiques qui sont en concurrence pour cet espace occupent ce qui reste libre**.

Une image peut être rendue flottante, comme un **`block`**.

2.1. exemple avec une image.

```
<body><div style="width:500px; margin-left:100px; border:1px; padding:5px;
  font-family:sans-serif;font-size:25px;border-style:solid;broder-
color:blue;">
 <p> bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla
bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla
     <img  style="float:right;" src="javalogo.gif" />
      bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla
bla  bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla
bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla
bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla
bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla
 </p>
      </div>
  </body>
```

fichier exemple : `hml07tp1.html`



2.2. exemple avec un conteneur.

```
<body>
  <div style="
      width:500px;
      margin-left:100px;
      border:1px;
      padding:5px;
      font-family:sans-serif;
```

```
      font-size:25px;
      border-style:solid;
      border-color:blue;">
    <p> bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla
     bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla
     <div style="
        float:left;
        width:200px;
        border:5px;
        border-style:solid;
        border-color:red;
        padding:10px;"/>
          toto toto toto toto toto toto toto toto toto
          toto toto toto toto toto toto toto toto toto toto
      </div>
           bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla
           bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla
           bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla
           bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla
           bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla
           bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla
    </p>
   </div>
  </body>
```

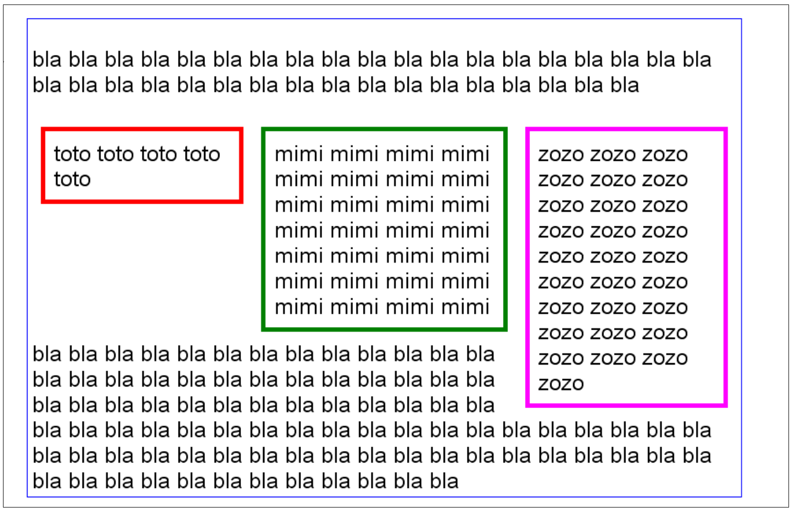
fichier exemple : `hml07tp1.html`

2.3. cas avec plusieurs conteneurs : un tricolonnage.

```
<body>
  <div style=" width:800px;
               margin-left:100px;
               border:1px;
               padding:5px;
               font-family:sans-serif;
               font-size:25px;
               border-style:solid;
               border-color:blue;" >
    <p> bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla
```

```
    bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla
    <div style="float:left;
                width:200px;
                border:5px;
                border-style:solid;
                border-color:red;
                margin:10px;
                padding:10px;" >
      toto toto toto toto toto
     </div>
     <div style="float:left;
                  width:250px;
     margin:10px;
     border:5px;
     border-style:solid;
     border-color:green;
     padding:10px;" >
      mimi  mimi mimi mimi mimi mimi mimi mimi mimi mimi mimi mimi mimi mimi mimi
      mimi mimi mimi mimi mimi mimi mimi mimi mimi mimi mimi mimi mimi mimi mimi
     </div>
     <div style="float:right;
                 width:200px;
                 border:5px;margin:10px;
                 border-style:solid;
                 border-color:magenta;
                 padding:10px; >
      zozo  zozo zozo zozo zozo zozo zozo zozo zozo zozo zozo zozo zozo zozo zozo
      zozo zozo zozo zozo zozo zozo zozo zozo zozo zozo zozo zozo zozo zozo zozo
     </div>
     bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla
     bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla
     bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla
     bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla
     bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla
     </p>
     </div>
  </body>
</html>
```

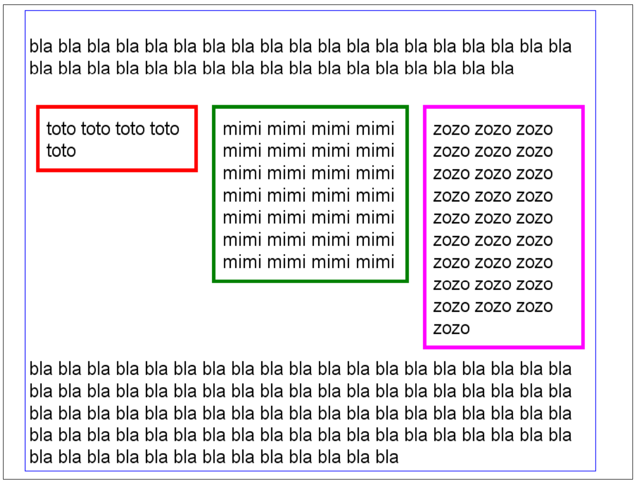
fichier exemple : `hml07tp2.html`



On peut avoir l'intention d'aligner le bas des trois conteneurs ; et vouloir **empêcher que le texte ne continue à s'enrouler sur les conteneurs flottants**. La commande **`clear`** permet d'empêcher l'enroulement à gauche (**`clear:left)`** ou à droite (**`clear:right`**) ou les deux (**`clear:both`**). Seulement, il faut une balise pour porter cette feuille de style. L'usage est d'utiliser la balise **`hr`**, avec une dimension nulle, ou une balise **`div`** à contenu "vide". **L'enroulement est ainsi arrêté pour la balise suivante.**

```
        mimi mimi mimi mimi mimi mimi mimi mimi mimi mimi mimi mimi mimi
      </div>
    <div style="clear:right;height:0;overflow:hidden" />abcd</div>
      bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla
```

fichier exemple : `hml07tp3.html`



Plus simplement, on utilisera la ligne à balise vide :

```
<div style="clear:right;" /></div>
```

3. positionnement.

3.1. Positionnement relatif.

Le positionnement relatif est un peu paradoxal, puisqu'un conteneur est en positionnement relatif …

par rapport à la place qu'il occuperait en l'absence de positionnement explicite ; le positionnement relatif se fait par rapport à la position par défaut. Un conteneur de type **block** a sa place "naturelle" dans le flot HTML. On le positionne relativement quand on le "déplace" en prenant comme repère la place qu'il a naturellement. Le déplacement peut être nul (par exemple si on n'explicite ni **top**, ni **left** ...) ; alors le conteneur occupe l'emplacement par défaut. Cette disposition est importante comme on le verra plus tard, pour le positionnement **absolute**.

Les commande de positionnement usuelles sont **top** et **left.** Exemple :

```
position: relative;
top:40px;
left: 20px;
```

Le conteneur a une position déplacée de 40px à partir du haut et 20px à partir de la gauche, relativement à la position qu'il aurait dans le flot HTML. On pourrait aussi avoir un positionnement à partir de la droite (**right**), ou du bas (**bottom**), moins utiles.

L'intérêt principal du positionnement relatif est que tout positionnement absolute d'un bloc se fait par rapport à un bloc déclaré en position relative (voir en 3.2.).

Note sur les coordonnées : les valeurs pour **top, left, right ou bottom** sont appliquées de façon un peu déconcertante. Voici ce qu'il en est pour left (à adapter pour les autres) : Le départ est **la partie gauche du contenu de la balise englobante** (droite du **padding-left**) ; l'arrivée est **la gauche balise englobée complète** (à gauche du **margin-left**). (voir le fichier **html-top-left.html**)

3.2. positionement absolute.

Le positionnement **absolute** fait sortir le conteneur du flot HTML. Un élément en position **absolute** doit toujours être défini à l'intérieur d'un élément en position **relative**. Ce dernier sert de repère ; on voit ici la nécessité de conteneurs en position **relative** et que l'on ne déplace pas.

Le flot HTML se déroule alors **comme si le conteneur n'existait pas** ; par exemple, le texte du flot naturel se développe "derrière" le conteneur en positionnement **absolute**, et il est donc caché ! On a donc toujours intérêt à placer un conteneur en positionnement **absolute** dans un autre conteneur bien dimensionné appartenant au flot HTML.

On peut donc avoir plusieurs conteneurs avec le même positionnement **absolute**, qui se cachent les uns les autres (on verra ultérieurement l'exemple du diaporama, composé d'images superposées) . On peut affecter une profondeur aux conteneurs en position **absolute** d'un même conteneur en position **relative** ; c'est le **z-index** (un entier positif). Plus le **z-index** est grand, plus le conteneur est "au dessus" ; par exemple, si on définit les z-index 5, 6, 10, 400, le conteneur de z-index 5 est en dessous, celui de z-index 400 est au dessus.

Le positionnement absolu n'a d'intérêt qu'en conjugaison avec le javascript.

3.3. overflow.

Que se passe-t-il lorsque le contenu d'un conteneur dépasse celui de son contenant ?

On peut avoir trois attitudes :

- cacher ce qui dépasse : **overflow:hidden;**

- laisser faire ; le contenu peut dépasser le contenant : **overflow:visible;** valeur par défaut**.**

- faire des ascenseurs ; c'est souvent la bonne solution : **overflow:scroll;** **overflow:auto;** ne place le scroll que s'il est nécessaire.

un exemple avec des ascenseurs :

```
<body>
  <div style ="position: relative; top:50px; left:50px;
```

```
            width:300px; height:300px; overflow:scroll;
            border:2px solid black;";>
    <img src="_phal.png" />
  </div>
</body>
```

4. les listes.

Les feuilles de style relatives aux listes sont beaucoup plus simples ; elles se résument en seulement 3 propriétés :

4.1. type de puce

`list-style-type :`

`decimal` = pour listes **`ol`**: numérotation 1.,2.,3.,4. etc...
`lower-roman` = pour listes **`ol`**: numérotation i.,ii.,iii.,iv. etc...
`upper-roman` = pour listes **`ol`**: numérotation I.,II.,III.,IV. etc...
`lower-alpha` = pour listes **`ol`**: numérotation a.,b.,c.,d. etc...
`upper-alpha` = pour listes **`ol`**: numérotation A.,B.,C.,D. etc...
`lower-latin` = pour listes **`ol`**: numérotation a.,b.,c.,d. etc... y compris le jeu de caractères latin étendu
`upper-latin` = pour listes **`ol`**: numérotation A.,B.,C.,D. etc... y compris le jeu de caractères latin étendu
`disc` = pour listes **`ul`** : rond plein comme puce
`circle` = pour listes **`ul`** : puce circulaire
`square` = pour listes **`ul`** : puce rectangulaire
`none` = pas de puce

4.2. position de la puce et du texte

`list-style-position :`

`inside` = puces et numérotation dans le corps de la liste (décalage sur la droite).
`outside` = retrait à gauche des puces et numérotation (réglage par défaut).

4.3. la puce est une image.

`list-style-image :`

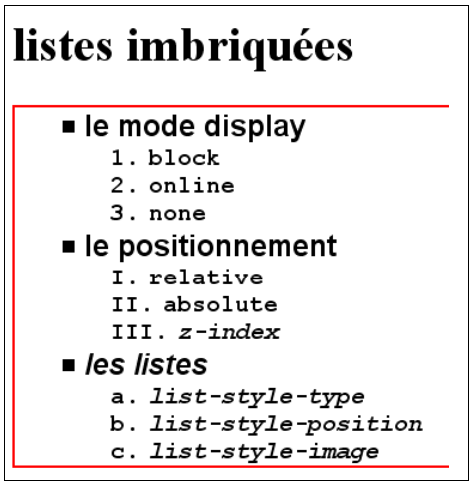
`list-style-image:url(nom_de_fichier)` : permet de remplacer la puce par une image

Les listes s'imbriquent très bien, ce qui permet plusieurs niveaux de hiérarchie.

Exemple: **`html07tp5.html`**

```
<body>
```

```
   <h1>listes imbriquées</h1>
   <ul style="list-style-type:square ;list-style-position:outside;
              font-family:sans-serif;font-size:25px;font-weight:bold;
              border: 2px solid red;">
      <li>le mode display</li>
        <ol style="list-style-type:decimal ;list-style-position:inside;
                   font-family:monospace;font-size:20px;font-weight:bold;" >
          <li>block</li>
          <li>online</li>
          <li>none</li>
        </ol>
      <li>le positionnement</li>
        <ol style="list-style-type:upper-roman ;list-style-position:inside;
                   font-family:monospace;font-size:20px;font-weight:bold;" >
          <li>relative</li>
          <li>absolute</li>
          <li ><i>z-index<i></li>
        </ol>
      <li>les listes</li>
        <ol style="list-style-type:lower-alpha;list-style-position:inside;
                  font-family:monospace;font-size:20px;font-weight:bold;" >
          <li>list-style-type</li>
          <li>list-style-position</li>
          <li><i>list-style-image</i></li>
        </ol>
   </ul>
 </body>
```



5. visibilité.

On a vu qu'un **block** pouvait être occulté ; (**block:none;).** L'espace qu'il aurait occupé est récupéré pour un autre affichage, celui du bloc suivant. Mais il se peut qu'on veuille simplement rendre invisible un objet, par exemple un bouton ou un lien qui ne doivent pas apparaître dans un contexte donné, sans cependant que la mise en forme du la page soit chamboulée.

L'attribut de style qui est lié à la visibilité d'un conteneur quelconque est **visibility** ; cet attribut prend deux valeurs : **hidden** et **visible**. L'espace réservé pour le conteneur est conservé, seul son contenu est devenu non visible, Il est évident que c'est à travers le javascript que ceci apparaît

important, par exemple :

```
document.getElementById("bouton").style.visibility = "visible";
```

8 formulaires

tutoriel HTML

fiche 8

introduction.

Les formulaires jouent un rôle particulier dans le HTML : leur utilité foncière est de pouvoir recevoir un traitement côté serveur (php avec le serveur Apache, java avec le serveur Tomcat etc.).

La fiche présente a donc un intérêt limité, si on ne s'intéresse au côté serveur :

* elle présente les balises et leurs attributs.

* elle se centre sur la manière dont le client perçoit les éléments de formulaire ;

* elle introduit au contrôle des formulaires, qui relève du javascript.

code des exemples : `html08tp0.html`

1. balise de définition d'un formulaire :

1.1. définition d'un formulaire.

```
<form action="URL_traitement"
      id="identificateur_du_formulaire"
      method="nom_de_la_méthode"
      enctype='multipart/form-data'>
```

* **`name`** est désuet ; utiliser **`id`** ; comme la balise **`form`** n'est pas une balise graphique, les attributs **`class, style`** et **`title`** ne sont pas utiles.

* **`action`** : comporte l'url du fichier de traitement (par exemple un fichier php ; mais ce peut être une fonction javascript) ; il peut être pratique dans certains cas (mise au point) de relancer la page actuelle ; dans ce cas on écrit simplement ""; (deux doubles quotes : url vide).

* **`method`** : en pratique, il y a trois méthodes utiles : **`post; get`** et **`file`**.

* **`enctype`** : donne le type MIME de l'encodage ; en pratique on n'utilise cet attribut que pour les upload (chargement de fichier vers le serveur ; la balise **`input/file`** doit être présente).

Les balises dites "de formulaire" on un comportement spécifique lorsqu'elles sont placées entre la balise ouvrante et la balise fermante d'un formulaire.

1.2. les balises input.

* Au début du HTML, les balises de saisie de formulaire sont toutes des balises **`input`** ; ce n'est que pour enrichir l'usage des formulaires qu'on été introduites les balises **`textarea`** et **`select`**.

* Les balises **`input`** possèdent toutes **une définition de type** obligatoire (**`text`** par défaut). Comme ce sont des balises de saisie, elles fonctionnent sur un mode **nom/valeur** et c'est ce couple qui est utilisé par le script côté serveur de traitement du formulaire.

* Il ne faut pas confondre le nom (**`name`**) qui est un attribut de la saisie et **`id`** qui désigne l'identificateur de la balise (attention, dans le vieil HTML il n'en était pas ainsi, ce qui laisse encore traîner des incorrections dans la littérature).

* les attributs **`class , style, title`** sont pertinents pour ces balises.

2. les types de balises input.

2.1. type="text"

C'est la balise de saisie des textes courts (nom, prénom, adresse …).

Ses attributs sont :

* **`size`** : taille de la fenêtre de saisie en caractères ;

* **`maxlenght`** : nombre maximal de caractère qui peuvent être saisis ;

* **`name`** : nom de la saisie (sert uniquement côté serveur ou avec javascript) ;

* **`value`** **c'est un t**exte dans le fichier HTML ; c'est la valeur de cet attribut qui apparaît côté serveur associée au nom.

exemple :

```
<input type="text" id="leNom" class="entree"
                  size="20" maxlength="32" value="" />
```

2.2. type ="password"

C'est rigoureusement la même chose que le type **`text`** ; mais sur l'écran, à l'affichage, les caractères sont replacés pas des astérisques (le mot de passe est illisible).

2.3. type="hidden"

Ce type ne permet pas de saisir des informations et il n'affiche rien ; mais il est traité côté serveur comme un attribut de saisie. Il permet ainsi de véhiculer des données, du serveur vers le navigateur puis du navigateur vers le serveur. Ce type n'a que trois attributs, **`id`**, **`name`** et **`value`**. L'identificateur **`id`** sert dans les scripts javascript, il ne faut pas le négliger !

2.4. type="checkbox"

il s'agit du type : cases à cocher: Les attributs spécifiques sont :

* **`name`** : nom de la balise

* **`value`** : valeur associée au nom

* **`checked`** : coche la case (ou non si il est absent) ; en principe, on ne met pas de valeur à ce nom d'attribut (cela peut changer à l'avenir, d'autant plus que javascript utilise une valeur explicite).

2.5. type="radio"

Les boutons radio fonctionnent comme les checkbox ; mais ils sont regroupés **sous un même nom**, et un seul bouton du groupe peut être marqué (**`checked`**) à la fois ; si on clique un bouton non marqué, celui qui était marqué (s'il existe) perd son marquage au profit du bouton qui vient d'être cliqué. Cliquer un bouton marqué est sans effet.

```
<label><input type="radio" name="groupeRadio"
              value="v1"/>valeur 1<br/>
</label>
<label><input type="radio" name="groupeRadio"
              value="v2" />valeur 2<br/>
</label>
<label><input type="radio" name="groupeRadio"
              value="v3" />valeur 2<br/>
</label>
```

note. seule les balises marquées (**checked)** sont transmises au serveur lors de la validation du formulaire, tant pour **checkbox** qu pour **radio**.

2.6. label pour checkbox et radio.

Les checkbox et radio ont en général une étiquette ; la balise **label** permet que le cliquage de l'étiquette soit similaire au cliquage du bouton radio, ou au cochage (décochage) du checkbox.

```
<label>
  <input type="checkbox" name="n1" value="v1" />valeur1<br/>
</label>
<label>
  <input type="checkbox" checked name="n2" value="v2"
          />valeur2
</label>
```

2.7. type="reset"

On entre ici dans les balises qui se manifestent graphiquement par un bouton ; en cas d'absence de feuille de style, le bouton est celui utilisé par le système d'exploitation.

la balise **reset** permet de revenir à la situation par défaut de l'ensemble du formulaire. Il y a un attribut spécifique, **value** qui donne l'inscription sur le bouton :

```
<input type="reset" value="annuler" />
```

La balise n'est pas transmise lors de la validation.

2.8. type="button"

Comme pour le bouton **reset**, seul l'attribut **value** est intéressant puisque c'est l'inscription qui est sur le bouton. Le bouton a la propriété de ne rien faire et de n'être pas transmis lors de l'évaluation.

Ce bouton est intéressant pour le **javascript**.

2.9. type="submit"

Il s'agit d'un bouton particulier où les attributs **name** et **value** ont le rôle habituel. Mais lorsque le bouton est cliqué, l'ensemble du formulaire est validé, c'est à dire que tous les couples **nom/valeurs** des balises qui sont signifiants (**text, password, radio, submit, hidden**... mais pas **button** ni **reset**) sont envoyés au serveur qui les transmet à la page désignée dans **action**...

Attention. en l'absence de cette balise, **il n'y a pas de validation du formulaire** par un procédé HTML (il peut y avoir validation par un script javascript). En présence de cette balise, la validation se fait indifféremment par la touche **entrée** validant le **input** de type **text, password** et le cliquage de l'**image.** Ceci est souvent un inconvénient, car sauf exception, c'est le bouton **submit** qui doit servir de validation de formulaire. On verra que grâce au javascript, divers modes de validation plus satisfaisants peuvent être mis en œuvre. Il peut y avoir plusieurs boutons **submit** dans un formulaire. Il est évident que dans ce cas, il vaut mieux donner des noms différents aux balises pour que le script serveur puisse s'y retrouver.

2.10. type ="image"

Pour le type image, il faut en plus des attributs **name/value**, un attribut qui donne l'url de l'image.

<input type="image" src="*url_de_l'image***" name="***monNom***" value="***maValeur***" />**

Il faut savoir que si l'on valide en cliquant l'image, le coordonnées de la souris sur l'image sont envoyées au serveur (c'est une forme de *mappage*).

note. Chaque balise peut être présente plusieurs fois dans un formulaire ; Une fois de plus, on rappelle que seule les balises activées sont transmises (balises **radio ou checkbox "checked",**

balise **submit** dont le bouton est cliqué…).

3. la balise textearea.

Il s'agit d'une balise de formulaire ajoutée ultérieurement, et d'une syntaxe différente :
<textarea>… </textarea>

Cette balise permet de saisir un texte long ; le texte par défaut est placé entre les balises ouvrante et fermante. Attention, tous les symboles, y compris les passages à la ligne et les espaces multiples sont transmis à la validation. C'est, avec la balise pre, une exception au principe de compression des espaces et retours de ligne du HTML.

Remarque importante : bien coller les balises si on veut un texte vide par défaut.

Les attributs spécifiques sont :

* **cols** : nombre de colonnes (en caractères fixes) ;

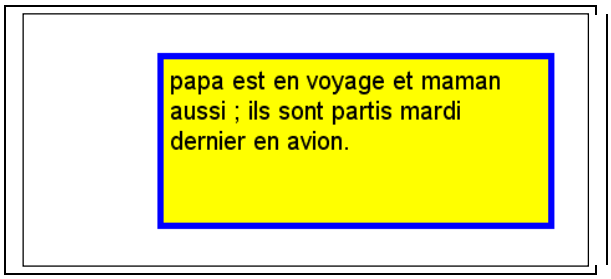
* **rows** : nombre de lignes ;

* **wrap** : passage automatique à la ligne ; en principe on ne déclare pas le wrapping qui est par défaut, sauf si on vient de le supprimer par **wrap="off"**.

En cas d'overflow des barres de scroll apparaissent automatiquement.

Les feuilles de style permettent une présentation choisie.

```
<textarea id="mt" cols="25" rows="4" name="texteSaisi"
       style="border:5px; border-style:solid;
       border-color: blue;
      font-family:arial; font-size=20px;
      background-color:yellow; margin-left: 100px;
      padding:5px;"></textarea>
```



4. la balise select.

La balise **select** a aussi été ajoutée postérieurement au HTML. Elle permet une sélection simple ou multiples d'items d'une liste déroulante.

```
<select id="identificateur" size="5" multiple >
  <option name="idOpt1" value="op un">
    première option
  </option>
  <option name="idOpt2" value="op deux">
    deuxième option
  </option>
  <option name="idOpt3" value="op trois">
    troisième option
  </option>
  <option name="idOpt4" value="op quatre">
    quatrième option
```

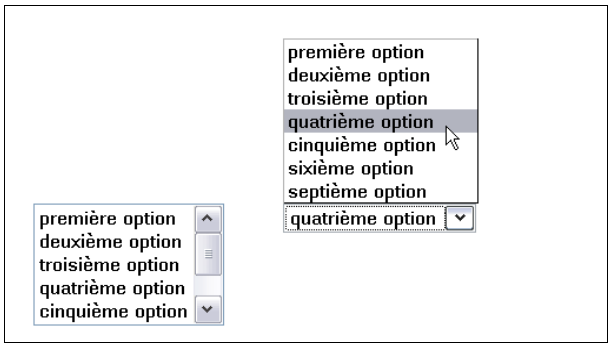
```
  </option>
</select>
```

- si la taille **size** n'est pas posée ni le booléen **multiple**, la liste des textes d'option apparaît comme un menu déroulant ; sinon la liste apparaît dans une fenêtre (avec ascenseur automatique, de taille égale à **size**).

- si le booléen **multiple** est posé, on peut activer une suite d'options en cliquant la première à activer, puis la touche **maj** pressée, en cliquant le dernier. Si on veut activer ou désactiver individuellement, tenir la touche **ctrl** pressée et cliquer l'item choisi.

- on peut ajouter le booléen **selected** aux options à activer par défaut.

- lors de la soumission, c'est (ce sont) le (les) couple(s) formé(s) de **name** et de **value** des balises **option** qui sera (seront) envoyés au serveur.



9 pseudo-formats, balises multiples

tutoriel HTML

fiche 9

introduction.

Sans être très riche en la matière, le HTML est doté d'un minimum de propriétés dynamiques, c'est-à-dire relatives à son interaction avec l'extérieur : un élément graphique peut avoir le focus, être survolé par la souris (hover ou over), être cliqué, garder la trace qu'il a été cliqué (visited), être activé (touche souris enfoncée au cours d'un survol). On reconnaît là les caractères dynamiques liés aux liens ancrés. En principe (w3c) ces caractéristiques devraient se retrouver sur tous les conteneurs. Malheureusement, mis à part le survol (hover), ou les propriétés propres aux éléments de formulaires ou aux boutons, les navigateurs ne les connaissent pas ; par contre, javascript apporte une gestion de toutes les caractéristiques dynamiques que l'on peut souhaiter pour un élément donné. Par exemple la soumission d'un formulaire, la prise ou la perte de focus d'une fenêtre ou d'un bouton, le survol d'un conteneur, le cliquage d'une image etc.

Lorsque les caractères dynamiques sont concrétisables dans des feuilles de style, il faut disposer de styles qui dépendent de l'histoire de l'élément : ces styles sont appelés pseudo-formats. Il ne s'appliquent qu'à des détermination précises de l'histoire de l'élément (survol, clic, focus etc).

1. déclaration des pseudo-formats des ancres.

Il y a quatre pseudo-formats d'ancre :

```
a:link
a:visited
a:hover
a:active
```

Attention : l'ordre dans lequel on définit les feuilles de style est important : en effet, si on définit **`visited`** après **`link`**, le **`link`** devient inutile après un premier clic ; même chose pour **`hover`** défini avant **`link`** ou **`visited`**. C'est une question de priorité.

Autre point important avec les balises d'ancres : elles ont des caractéristiques de style par défaut qui, avant l'avènement des feuilles de style explicites, pouvait avoir un intérêt. Dans les circonstances actuelles elles n'ont plus lieu d'être : **couleur, fond et décoration** doivent être redéfinis pour chaque pseudo-balise en fonction du graphisme de la page, et échapper ainsi à un style universel par défaut (liens soulignés colorés en bleu, rougis s'ils ont été visités).

2. balises multiples.

2.1. balise + identificateur

On peut restreindre l'effet d'une feuille de style aux balises d'identificateur donné ou simplement, à une balise donnée.

`div#monIdentificateur` : la feuille de style s'applique uniquement aux balises **`div`** ayant pour identificateur **`monIdentificateur`**. On se rappelle que la feuille de style peut s'appliquer à plusieurs fichiers HTML.

Une bonne pratique consiste à déclarer les feuille de style attachées à un identificateur sous la forme balise + identificateur (voir l'étude de la section 3).

2.1. balise + classe

le principe est le même, mais s'adresse aux classes **:**

```
div.maClasse
p.maClasse
```

Attention : il n'y a pas de séparateur avant le dièse (identificateur) ni avant le point (classe) !

2.3. inclusion de balises

Si on déclare une feuille de style de la façon suivante :

`div p` { ... }

la feuille de style s'adresse aux paragraphes **`p`** inclus dans un conteneur **`div`**. On peut être plus précis :

div.maClasse p ; la feuille de style s'adresse aux balises p incluses aux balises div ayant pour classe maClasse (attention : pas d'espace avec le point, un espace avant le **`p`**).

Autres cas :
`div#monId p`
`div p.maClasse` : attention à l'espace
`p span#monId`
`p#monId a:link`

2.3. double définition

Il existe de nombreuses autres situations, mais de peu d'usage. Il faut de plus se méfier des définitions multiples (séparées par des virgules) qui sont difficiles à relire. Dans un premier temps, il vaut mieux se contenter des cas cités qui représentent la majeure partie de cas rencontrés. On va cependant citer un exemple :

```
p, blockquote {
  font-family : sans-serif;
  text-align:justify;
  font-size:20px;
}
blockquote {
  font-style:italic;
}
```

La double définition permet de ne changer qu'un attribut pour passer de **`p`** à **`blockquote`**.

3. un exemple d'analyse avec balises identifiées.

3.1. le cahier des charges.

On demande de réaliser une page HTML avec le contenu suivant :

3.1.1. structure.

La page comporte une entête, une barre de menu, un contenu texte et un pied de page.

L'entête montre une image (un fruit), et un bandeau qui sert de titre de la page. Si on clique le bandeau, on appelle une page appelée **`accueil.html`**.

La barre de menu comporte 5 cases, qui sont des hyperliens. Par commodité, dans cet exercice, on laisse le lien vide (= page actuelle).

La partie de contenu comporte un texte en trois parties : titre, sous-titre, textes de paragraphe. Une image illustre ce contenu.

Le pied de page ne contient qu'une indication sommaire (date d'édition).

Il existe de plus une contrainte physique : la partie active de la page doit être centrée et avoir 770 pixels, de façon à pouvoir apparaître sans l'ascenseur du bas même sur les vieux écrans (800x600 pixels).

3.1.2. le sujet.

Le sujet n'a pas d'importance pour l'exemple ; l'accent est mis sur l'analyses HTML et l'analyse

graphique sous forme de feuilles de style, sans souci de recherche esthétique. On suppose que l'on dispose du texte et des images.

On dispose du texte :

```
utilisation des poires
      leur consommation courante
      Les poires sont des fruits couramment consommés tels quels. Le dessert composé d'une poire
au sirop, d'une boule de glace à la vanille, de chantilly et de chocolat chaud est appelé "Poire
Belle-Hélène" ; Le jus de poire fermenté donne une boisson légèrement alcoolisée appelée
poiré.</p>
      Les poires peuvent également être utilisées pour produire de l'eau-de-vie. La plus connue
d'entre elles s'obtient à base de la poire Williams. L'alcool produit est appelé communément
Williamine ou familièrement poire. La bouteille de Williamine contient le plus souvent une poire
en son centre. Pour ce faire, les producteurs introduisent les jeunes pousses de poires encore
accrochées au poirier dans des bouteilles qu'ils suspendent aux branches. En grandissant, la
poire devient impossible à ressortir. Les poires ont notamment des propriétés diurétiques.
```

des images suivantes en format png :

note : ne pas utiliser d'images détourés en format jpeg car l'artefact jpeg apparaît rapidement de manière gênante dans le détourage, que l'on souhaite bien clair !

En guise d'illustration, voici la saisie d'écran d'une réalisation :

3.2. Analyse HTML.

3.2.1. la contrainte de largeur.

La contrainte de largeur (770px) implique, pour le HTML, d'avoir l'ensemble du code regroupé. Une balise div englobante convient pour cela.

3.2.2. l'entête.

L'entête est un conteneur ; peu importe lequel. On va prendre h1 pour rappeler que c'est le titre de la page. Si on met la poire comme image de fond, l'analyse HTML n'a pas à prendre en compte cet élément. La seule chose importante est le lien vers la page **`accueil.html`**. D'où la partie entête :

```
<h1 id="header">
        <a href="accueil.html" title="accueil du site"></a>
</h1>
```

Noter que le lien est vide ! Il n'y a pas de texte ; on pourrait y mettre l'image, mais rien ne nous y oblige.

3.2.3. le menu.

L'analyse HTML est ici plus délicate. Qu'est-ce qu'un menu ? Son constituant de base est le lien ancré : il y a donc 5 liens ancrés avec du texte : *menu 1, menu 2* etc.

Mais il y faut un mode de regroupement pour marquer que ces liens ancrés forment un tout. Et là, il y a trois points de vue possibles (au moins).

- premier point de vue : les liens forment une liste ;
- deuxième point de vue : le menu est un tableau dont les données sont les liens ancrés.
- troisième point de vue : le menu est une séquence (succession) de liens ancrés regroupés dans un même conteneur.

* La liste peut être préférée si on veut insister sur le fait que le menu est un élément de navigation qui permet de choisir parmi **des éléments diversifiés**. La diversification est affirmée par la puce ou l'icone de chaque item ; cette présentation est naturelle dans une liste présentée verticalement, ce qui n'est pas le cas ici.

note : le **comment faire** est exclus de l'analyse HTML. Le code serait alors :

```
<ul  id="menu">
  <li><a href="">menu 1</a></li>
  <li><a href="">menu 2</a></li>
  <li><a href="">menu 3</a></li>
  <li><a href="">menu 4</a></li>
  <li><a href="">menu 5</a></li>
</ul>
```

* le tableau est possible ; notons que le tableau a constitué longtemps un (mauvais) moyen de mise en page et qu'on est tenté d'y avoir recours sans trop y réfléchir.

* la séquence.

Le lien ancré est un conteneur ; le mettre dans un conteneur (**li** ou **td**) lui même contenu dans un conteneur de regroupement (**ul** ou **tr/table**) est une solution qui n'est pas naturelle : comment sémantiquement justifier ces conteneurs englobants ? La bonne analyse est de choisir la séquence :

```
<div id="menu">
  <a href="">menu 1</a>
  <a href="">menu 2</a>
  <a href="">menu 3</a>
  <a href="">menu 4</a>
  <a id="der" href="">menu 5</a>
</div>
```

Il n'y a d'identificateur que sur la dernière ancre ; on pourrait évidemment en mettre sur toutes.

3.2.4. le texte.

Supposons qu'il y ait deux paragraphes ; on a simplement :

```
<div id="contenu">
  <h2>le titre du texte</h2>
  <h3>le sous-titre du texte</h3>
  <p id="initial">paragraphe initial</p>
  <p id="final">paragraphe final<p>
</div>
```

S'il y avait davantage de paragraphes, on insérerait des balises **p** , en principe non identifiées, mais avec un une propriété **class**.

3.2.5. le pied de page.

Il s'agit simplement d'un texte dans son conteneur (peu importe lequel) :

```
<div id="footer">pied de page</div>
```

3.3. le fichier HTML.

Abstraction faite des feuilles de style, on obtient le fichier :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
          "http://www.w3.org/TR/html4/strict.dtd">
<html>
```

```
<head>
  <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
  <title>css et identificateurs</title>

</head>
<body>
  <div id="cadre">

    <h1 id="header">
      <a href="accueil.html" title="accueil du site"></a>
    </h1>

    <div id="menu">
      <a href="">menu 1</a>
      <a href="">menu 2</a>
      <a href="">menu 3</a>
      <a href="">menu 4</a>
      <a id="der" href="">menu 5</a>
      <div id="fin"></div>
    </div>

    <div id="contenu">
      <h2>utilisation des poires</h2>
      <h3>leur consommation courante</h3>
      <p id="initial">
        Les poires sont des fruits couramment consommés tels quels. Le dessert
composé d'une poire au sirop, d'une boule de glace à la vanille, de chantilly et
de chocolat chaud est appelé "Poire Belle-Hélène" ; Le jus de poire fermenté
donne une boisson légèrement alcoolisée appelée poiré.
      </p>
      <p id="final">Les poires peuvent également être utilisées pour produire de
l'eau-de-vie. La plus connue d'entre elles s'obtient à base de la poire
Williams. L'alcool produit est appelé communément Williamine ou familièrement
poire. La bouteille de Williamine contient le plus souvent une poire en son
centre. Pour ce faire, les producteurs introduisent les jeunes pousses de poires
encore accrochées au poirier dans des bouteilles qu'ils suspendent aux branches.
En grandissant, la poire devient impossible à ressortir. Les poires ont
notamment des propriétés diurétiques.
      </p>
    </div>

    <div id="footer">Mont-Saint-Éloi le 13 mai 2008
    </div>

  </div>
</body>
</html>
```

3.4. le cadre.

Voici les feuilles de style qui régissent l'ensemble de la page :

```
body {
    background-color: silver;
    font-family:sans-serif;
    font-size : 18px;
  }
  div#cadre {
    width: 770px;
    margin-left:auto;
    margin-right:auto;
    background-color: white;
  }
```

* le **background** déclaré dans **body** s'applique à la page entière (et pas à l'intérieur du conteneur associé à la balise, comme cela fonctionne avec tous les autres conteneurs).

* les propriétés de fontes de **body** s'imposent à la page entière.

* le cadre a une largeur définie : 770px. Il est centré grâce aux deux balises de marge déclarées à **auto**. Une règle non écrite dans les spécification est de toujours définir conjointement marge gauche et marge droite : en effet, l'analyse de marge se fait en commençant par le marge gauche (puis haut, droite et bas). Si on laisse la marge gauche par défaut à la valeur 0, c'est elle qui s'impose en priorité. Si on voulait une mage droite de 10% on déclarerait : margin-left:auto; margin-right:10%;

* Le **background** s'impose ici à tout le flot HTML : pas aux éléments qui sont sortis du flot , comme les éléments flottants (constituants du menu) ou en positionnement **absolute.**

3.5. détails de l'entête.

3.5.1. Un conteneur.

```
h1#header {
    height:170px;
    background-image:url(poirequeue.jpg);
    background-repeat:no-repeat;
    background-position:left top;
    background-color: rgb(52, 35, 25);
    margin:0;
  }
```

* Par défaut, les conteneurs de type **block** occupent toute la largeur de leur contenant. Il est inutile de définir à 770px la largeur de **h1#header**. Par contre la hauteur doit l'être car, par défaut, c'est le contenu qui impose la hauteur ; si on ne précise pas la hauteur, seule la balise **a** est retenue comme contenu..

* la couleur de fond ne sert que pour ce qui n'est pas l'image de fond ; pour avoir une continuité, la couleur est celle qui domine sur les bords de l'image (calculée dans un logiciel d'édition graphique).

* ne pas oublier le **margin:0;** sinon, on se retrouve avec les marges par défaut de la balise **h1 ;** si on avait choisi la balise **div,** la question ne se serait pas posée. Les autres propriétés graphiques de cette balise **h1** ne sont pas utilisées.

3.5.2. la balise d'ancre comme conteneur dimensionné et positionné.

```
h1#header a {
    display:block; /* pour les dimensions et la position */
    height:90px;
    width: 550px;
```

```
   background-image: url(banniere.jpg);
   background-repeat:no-repeat;
   background-position:left top;
   position:relative; /* pour positionner */
   left:190px;
   top:15px;
}
```

* La balise d'ancre, **a,** est un conteneur, d'affichage **inline** par défaut. Cette propriété l'apparente de ce point de vue à la balise **span** : dans le fil d'un texte quelconque, une partie peut être marquée comme lien. Mais la balise **a** dispose d'une feuille de style par défaut, et de propriétés liées à son rôle de lien. Par exemple, c'est la seule balise disposant de décorations (souligné) et de couleurs variables selon son histoire (lien non utilisé ou déjà visité). Du point de vue des feuilles de style, il est la seule balise qui dispose sur tous les navigateurs de tous les pseudo-formats.

* En premier lieu, on transforme la balise en type **block**. On peut alors lui donner les dimensions de l'image de fond qui sert de titre et lui affecter cette image.

* Le conteneur de la balise **a** est placé par défaut en haut et à gauche de celui de la balise **h1** car il est le premier (et seul) élément inclus. On lui donne donc un positionnement relatif et un déplacement caractérisé par **left:190px; top:15px;**

3.6. Le menu.

* On utilise le même principe que précédemment pour transformer chaque occurrence de la balise **a** en **block** et dimensionner correctement. La bordure est de 1px. Comme on veut que l'ensemble tienne sur la largeur de 770px, il faut se rappeler qu'il y a 6 bordures verticales pour 5 items ; il y a un petit rattrapage à faire (sur le dernier item) et disposer d'une barre supplémentaire (à droite, sur le même item).

* Il faut ensuite faire flotter ces conteneurs dans le conteneur **div#menu**. Il y a trois difficultés dont il faut tenir compte :

- gérer les pseudo-formats et supprimer les décorations. On se contente ici de gérer le survol du conteneur, en le signalant par le changement de curseur (on garde ici la valeur par défaut, le pointeur en forme de main) et un changement de la couleur de fond.

- la couleur du texte est définie spécifiquement pour les balise d'ancre ; il faut donc la redéfinir dans chaque balise d'ancre et ne pas compter sur l'héritage de la balise div englobante.

- ne pas oublier que tant qu'on n'a pas signalé la rentrée dans le flot HTML, le flot HTML est "enroulant" relativement aux balises flottantes. Dans le cas présent, les éléments de style (marge par exemple) des balises qui suivent le menu ne sont pas ceux attendus si le flottement n'est pas interrompu. Classiquement, on interrompt le flottement dans une balise telle que **hr** ou plus sûrement en ajoutant une balise **div** dont la seule caractéristique de style est de d'annuler les conséquences du flottement (l'enroulement) ; le flottement est interrompu dans la balise qui suit la balise d'interruption.

```
<div id="fin"></div>
```

Il y a donc une balise à rajouter au texte HTML. On a là un des rares cas où il faut modifier le fichier HTML à cause du style utilisé !

```
div#menu {
  font-size:20px;
  font-weight:bold;
}

div#menu a {
  /* faire un block avec un inline */
```

```
    display:block;
    float:left;
    width:153px;
    /* caractéristiques d'aspect */
    background-color: #a04070;
    border-style:solid;
    border-color:lime;
    border-left-width:1px;
    border-top-width:1px;
    border-bottom-width:1px;
    border-right-width:0;
    padding-top:5px;
    padding-bottom:5px;
    /* le texte */
    color:white;
    text-align:center;
    text-decoration:none;
  }

  div#menu a:hover { /* survoler */
    background-color: #ff4070;

  }
  div#menu a#der { /* barre pour le dernier et
          dimensions : largeurs + barres 770 = 4 x 153 + 152 + 5 + 1 */
    width:152px;
    border-right-width: 1px;
  }

  div#menu div#fin {
    clear:left; /* NE PAS OUBLIER */
  }
```

3.7. le contenu.

```
div#contenu {
    padding: 0 30px 0 50px; /* haut, gauche, bas et droite.*/
    background-image: url(poirefruit.png);
    background-repeat: no-repeat;
    background-position: 20px 10px;
  }

  div#contenu h2 {
    margin-top:5px;
    margin-left:100px;
    margin-bottom:0;
    padding-left: 15px;
    font-size: 200%;
    color: #009000;
```

```
    border-bottom: 1px solid #009000;/* effet */
    border-left: 5px solid #009000;
  }

  div#contenu h3 {
    margin-top:5px;
    margin-left: 160px;
    padding-left: 15px;
    font-size: 130%;
    color: #009000;
    border-bottom: 1px solid #009000;
    border-left: 5px solid #009000;
  }

  div#contenu p {
    text-align: justify;
    text-indent:2em;
  }

  div#contenu p#initial {
    padding-left:130px;
  }
```

* on note l'utilisation de deux unités non encore rencontrées : le pourcentage et l'**em.**

Voir la fiche 14 pour le détail : le pourcentage s'exprime en fonction de la valeur actuellement en vigueur dans l'élément ; comme ici la taille de fonte est uniformément de 18px (**body**), le pourcentage s'exprime sur cette base de 18px. Si il s'était agi d'une dimension, le pourcentage se serait appliqué aux dimension du conteneur englobant ou sinon de l'écran. Quant à l'**em,** c'est une unité, qu'il ne faut pas confondre avec le nom de balise homonyme ****, égale à la largeur du m minuscule dans la fonte actuelle (unité classique en typographie, pour mesurer les espacements sur la ligne ; le *cadratin* des typographe s'apparente à l'em).

3.8. le pied de page.

```
div#footer {
    text-align:center;
    margin-top:10px;
    padding-top:5px;
    border-top: 1px solid #009000;
    color : #009000;
  }
```

Il n'y a pas de commentaire à faire !

10 le javascript "minimal"

tutoriel HTML

fiche 10

introduction.

Le javascript est un langage à part entière et il n'est pas question de l'étudier ici. Cependant le HTML usuel nécessite, pour avoir un minimum de dynamisme, un petit nombre de propriétés javascript, toujours les mêmes dans les sites courants ; ce qui rend ce minimum javascript incontournable, et son utilisation relève plus de la **recette** que le l'élaboration de formules originales.

1. événements et codes en ligne.

Un script javascript est une action ou un enchaînement **d'actions** qui affectent la page HTML. Avec des scripts javascript, la page n'est plus statique ; on peut commander l'affichage d'une image non prévue à l'affichage initial (exemple : un zoom de l'image), vérifier qu'une entrée d'un formulaire a bien été remplie, visualiser une fenêtre d'alerte. Le script est écrit dans un langage propre, en mode texte, comme le HTML ou les feuilles de style. Il n'y a pas de pré-compilation des scripts javascript comme c'est le cas en Java. La compilation se fait « *on the fly* » (littéralement : au cours du vol).

Il y a deux façon d'exécuter un script javascript :

* Soit on le met dans le fil du fichier HTML ; il est exécuté "dans la foulée", au cours de la prise en charge des items du fichier HTML, comme s'il était lui-même une balise HTML (même dans la partie **head** du fichier).

* Soit on le déclenche à l'aide d'un "événement" (clic de la souris par exemple). La méthode usuelle la plus pratique est de placer le script dans une "fonction" ; la référence à l'événement est placée dans une balises HTML et la fonction est appelée par l'événement lorsqu'il se produit. Par exemple, si on a un bouton, on peut lui associer l'événement onClick (qui se produit lorsque l'on clique le bouton). On lie la fonction à l'événement ; le contenu de la fonction est réalisé lorsque l'on clique sur le bouton. Les événements possibles sont peu nombreux et seront évoqués en leur temps.

attention : bien copier les scripts, en respectant majuscules et minuscules. Ne pas ajouter d'espace là où il n'y en a pas ; les chaînes de caractères sont sur une seule ligne. Bien respecter les balises : ne pas oublier les symboles de fin comme les "quotes", le /> ou le > de la balise.

1.1. exemple 1 : code en ligne.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
          "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
  <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
  <title>javascript en ligne ; bouton</title>
  </head>
  <body>
  <script type="text/javascript">
    document.write("<h1>largeur de l'écran : ");
    document.write(screen.width);
    document.write(" px</h1>");
    document.write("<h1>hauteur de l'écran : ");
```

```
    document.write(screen.height);
    document.write(" px</h1>");
  </script>
  </body>
</html>
```

affiche :

largeur de l'écran :1152 px

hauteur de l'écran :864 px

Il faut retenir que **`document.write()`** écrit directement dans la page HTML, au cours de la lecture par l'analyseur de ladite page et avant que celle-ci soit interprétée par l'afficheur du navigateur. **Tout se passe comme si le texte HTML était complété par les balises et valeurs écrites par le script.** Ceci permet de placer où l'on veut dans la page HTML, des lignes HTML utilisant des propriétés strictement javascript comme le calcul des dimension de l'écran, et qui sont interprétées ensuite.

1.2. Code déclenché par un événement.

Si on crée un bouton, l'événement par défaut est le clic sur le bouton. L'événement s'appelle **`onClick`**, et une bonne habitude consiste à lui associer une fonction, qui est appelée lorsque l'événement se produit. On évite de mettre du code javascript dans le corps des balises, même si cela est autorisé : on obtient ainsi une meilleure lisibilité et il est plus facile de modifier le code ou de réemployer du code emprunté à d'autres pages HTML.

exemple :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
          "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
  <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
  <meta name="generator" content="PSPad editor, www.pspad.com">
  <title>événement</title>
  <script type="text/javascript">
    function alerter() {
      alert ("vous venez de cliquer le bouton");
    }
  </script>
  </head>
  <body>
  <h3> cliquer sur le bouton   
    <input type="button" value ="bouton d'alerte" onClick="alerter()" />
  </h3>
  </body>
</html>
```

* l'usage est de situer le script dans l'en-tête du fichier HTML. On peut très bien le placer dans le corps (après la balise **body**).

* la fonction **alert()** est incluse dans la standard javascript ; elle crée une fenêtre *popup* (jaillissante) avec pour message l'argument de la fonction alert().

Cette fonction est un outil privilégié de mise au point car elle permet au programmeur de contrôler une valeur intermédiaire, un résultat calculé etc. En cas d'erreur, la fonction ne se déclenche pas. Firefox dispose d'un outil intéressant de détection et de signalement des erreurs (pas seulement javascript, mais aussi des feuilles de style) : menu **outils>console d'erreurs.** Les autres navigateurs peuvent également disposer d'outils plus ou moins performants, moins pratiques d'emploi.

2. les événements essentiels.

2.1. les attributs événements.

onLoad : placé dans la balise **body** ; se déclenche à la fin du chargement du fichier HTML, images comprises. Cet événement permet de traiter par javascript les balises du fichier HTML (par exemple les balises de formulaires, les images etc) qui ne sont connues qu'après chargement du fichier HTML. On utilise moins **onUnload**, exécuté en changeant de page (attention, un changement de page est un appel hypertexte sur le site ; il n'y a pas changement de page si on travaille sur un autre onglet (exemple : une page dans une autre fenêtre), ou si on arrête le navigateur.

onFocus : déclenché lorsqu'un élément HTML prend le focus ; par exemple lorsqu'un bouton, une balise **input**, une fenêtre *popup* etc. prend le focus. Si c'est une fenêtre qui prend le focus, l'événement appartient à cette fenêtre (il faut mettre le traitement dans la bonne fenêtre !).

onBlur : déclenché lorsqu'un élément perd le focus. Par exemple si une fenêtre *popup* a le focus et que l'on clique en dehors de cette fenêtre, elle perd le focus. Dans ce cas, on peut invoquer un traitement qui précède immédiatement le changement de focus (par exemple, demander la fenêtre de se fermer elle-même ; voir l'exercice 0 de la fiche11).

onClick : déclenché lorsque l'on clique l'élément graphique de la balise qui comporte cet attribut. Il existe aussi un **onDoubleClick.** Le **onClick** est souvent équivalent, sur un élément (bouton, image, balise **input**) ayant le focus, à la frappe de la touche entrée.

onMouseOver : déclenché lorsque le souris passe de l'extérieur à l'intérieur d'un élément graphique : par exemple si un conteneur possède l'attribut **onMouseOver**, celui-ci se déclenche lorsque le curseur de la souris pénètre sur le conteneur.

onMouseOut : c'est le contraire de **onMouseOver** ; le déclenchement se fait lorsque l'on passe de l'intérieur à l'extérieur de l'élément.

onMouseDown : lorsque la touche de souris est appuyée sur un élément graphique.

onMouseUp : lorsque la touche de souris est relâchée.

onMouseMove : lorsque la souris bouge sur l'élément graphique défini par la balise, qu'une touche soit appuyée ou non.

onKeyPress : lorsqu'on traite une touche comme si on tapait un caractère dans un traitement de texte.

onKeyDown : lorsqu'on appuie sur une touche.

onKeyUp : lorsque, après avoir tenu une touche appuyée, on la relâche.

note : l'orthographe retenue ci-dessus est celle qu'on emploie dans les balises ; les majuscules ne sont pas indispensables. Dans les scripts, il faut supprimer les majuscules (encore une question liée à l'histoire du HTML).

Les principaux événements seront étudiés sur des exemples typiques, c'est-à-dire des exemples à adapter ou simplement à recopier.

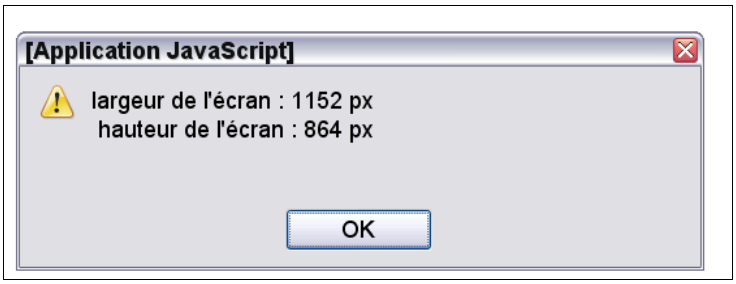
2.2. Exemple.

L'exemple consiste à afficher les dimensions de l'écran au chargement de la page.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
          "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
  <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
  <title>événement</title>
  <script type="text/javascript">
    function dimensions() {
      var affichage ="largeur de l'écran : ";
      affichage += screen.width;
      affichage += " px \n hauteur de l'écran : ";
      affichage += screen.height;
      affichage += " px";
      alert (affichage);
    }
  </script>
  </head>
  <body onLoad="dimensions();">
  <h3>essai de onLoad</h3>
  </body>
</html>
```

* on remarque comment se fait la concaténation de chaînes : avec le signe + . Cet usage n'est pas sans difficulté, puisque la concaténation de deux résultats numériques peut ne pas correspondre à l'attente. Exemple si on fait 123 + 456 , on obtient 569 et non 123456 ; dans ce cas, il faut transformer explicitement les nombres en chaînes par la fonction **toString()**.

* le symbole pour aller à la ligne est **\n**



11 fenêtres popup

tutoriel HTML

fiche 11

introduction.

La création d'une fenêtre *popup* est un exemple d'utilisation du javascript. Une fenêtre *popup* est une fenêtre qui apparaît sur l'écran à la suite d'un événement quelconque réalisé dans la fenêtre active, et sur laquelle elle vient s'afficher ; la fenêtre **`alert`** est une fenêtre *popup*. Sa durée de vie est faible et sa disparition conduit à rendre active la fenêtre qui l'a appelée. Dans le cas où il faut obligatoirement la fermer pour continuer, on dit que la fenêtre est **`modale`** ; une fenêtre modale est donc toujours au-dessus des autres.

1. Image agrandie.

Thème d'étude : On dispose de vignettes sur une page et lorsque l'on clique une vignette, l'image zoomée apparaît dans une fenêtre *popup* adaptée. On va le faire avec 3 images.

1.1. première étape : la page de saisie (exemple `html11tp0a.htlm`)

Pou simplifier, les trois vignettes ont même dimension (150x120 px), et les vignettes ainsi que les trois images sont dans le dossier du fichier HTML. Les url relatives se réduisent donc au nom de fichier.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
                      "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
    <title>
      popup a
    </title>
<script type="text/javascript">
function fpop(adresse,largeur,hauteur) {
   alert (adresse+" / largeur: "+largeur+" / hauteur: "+hauteur);
}
</script>
  </head>
  <body>
    <h1>fenêtre pop</h1>
    <br>
    <table style="border: 1px solid black; margin-left: auto;
                margin-right: auto;">
      <tr>
        <td style="border: 1px solid black; padding: 10px;">
          <img src="_vbesse.png"
               onclick="fpop('_besse.png',450,450);"></td>
        <td style="border: 1px solid black; padding: 10px;">
          <img src="_vphal.png"
               onclick="fpop('_phal.png',450,500);"></td>
        <td style="border: 1px solid black; padding: 10px;">
          <img src="_vvero.png"
```

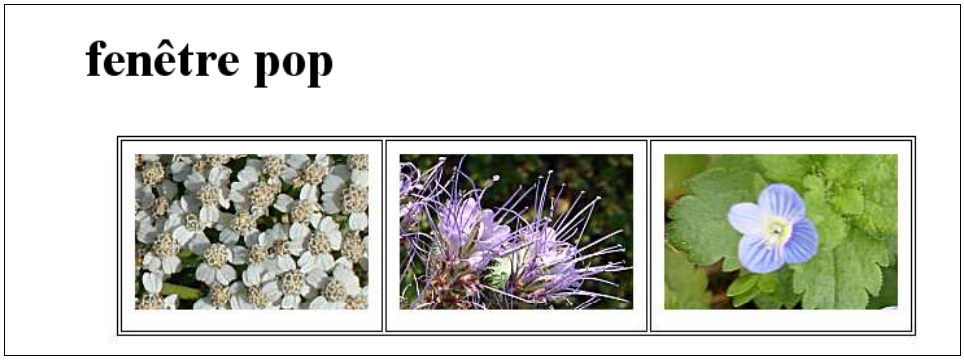
```
            onclick="fpop('_vero.png',570,450);"></td>
      </tr>
    </table>
  </body>
</html>
```

* quand on clique sur une vignette, on appelle la fonction **fpop** (abrégé pour "créer une fenêtre pop"). Comme dans tous le langages, on peut "passer des paramètres" à la fonction appelée ; ici, il y a trois paramètres : le nom du fichier image à afficher en grand, et ses deux dimensions, largeur et hauteur, en pixels.

fpop('_vero.png',570,450): signifie que l'on appelle la fonction **fpop**, avec comme premier paramètre le nom de l'image à afficher : **'_vero.png'** . Comme c'est une chaîne de caractères, on la place entre quotes (les doubles-quotes sont déjà employées dans le texte HTML, on utilise alors des quotes simples). Les deux autres paramètres sont des nombres, largeur et hauteur de l'image (il n'y a pas de quotes).

* théoriquement, c'est la fonction **fpop()** qui doit créer la fenêtre et y placer l'image zoomée. Mais pour réaliser un contrôle, on va se contenter dans un premier temps de la fonction **alert()** vue précédemment pour vérifier le passage de paramètres lors du clic de la vignette.

la page de saisie a l'allure suivante :



et l'appel de fonction donne la fenêtre **alert()** :

1.2. deuxième étape : la création de la fenêtre (html11tp0b.htlm)

```
var lg=screen.width;
var ht=screen.height;

function fpop(adresse,largeur,hauteur) {
   var fenPopLeft=(lg-largeur) / 2;
   var fenPopTop=(ht-hauteur)/2;

   fenPop= window.open("","","width="+largeur+",height="+hauteur
```

```
   +",top="+fenPopTop+",left="+fenPopLeft);
      fenPop.document.write("<html><head></head><body>");
   fenPop.document.write("<img src='"+adresse+"' />");

   fenPop.document.write("</body></html>");
   fenPop.document.close();
}
```

*** fenPop= window.open("","","width="+largeur+",height="+hauteur**
+",top="+fenPopTop+",left="+fenPopLeft);

La fenêtre s'appelle **fenPop()** ; la fonction de création a trois arguments obligatoires, qui sont des chaînes de caractères ; le premier n'est utilisé que si l'on charge une page HTML dans la fenêtre active; ce n'est pas le cas ici (d'où les quotes vides) ; le second n'est pas plus intéressant (il concerne le vieil HTML). Reste le troisième où on place les caractéristiques de la fenêtre, sous la forme nom/valeur, séparées par des virgules.

* Il y a deux choses intéressantes dans notre problème : avoir un espace d'affichage de même dimension que l'image ; cet espace d'affichage devraient être définie avec précision, mais **width** et **height** sont les dimensions de la fenêtre sans la barre de titre (celle avec les symboles de réduction et de fermeture). Ces paramètres de la fenêtre donnent un espace légèrement trop petit, ce qui est peu gênant. On ne peut malheureusement pas être plus précis à peu de frais ; la raison en est que les bordures et la barre de contrôle dépendent du système d'exploitation et de ses réglages, dont les dimensions et la définition de l'écran.

L'autre caractéristique exigée, c'est que la fenêtre ne soit pas n'importe où dans la page ; et même plus précisément qu'elle soit à peu près centrée ; comme on sait définir les dimensions de l'écran, et que l'on dispose des dimensions approximatives de l'image affichée, on peut avec un certaine précision définir l'espacement en haut de l'écran et la marge gauche : il suffit de négliger la barre de titre, ce qui ne tire pas beaucoup à conséquences.

d'où le double calcul :

```
var fenPopLeft=(lg-largeur)/2; (marge gauche)
var fenPopTop=(ht-hauteur)/2; (marge haute)
```

et le report du résultat dans les caractéristiques de l'image :

```
top="+fenPopTop+",left="+fenPopLeft
```

Le mélange de valeurs numériques et de chaînes qu'il faut concaténer crée un joyeux embrouillamini ; mais, avec un peu d'attention, on s'en sort.

* reste à écrire la page HTML grâce à la fonction **document.write()**, en n'oubliant pas de l'appliquer à la fenêtre que l'on vient de créer, **fenPop**. La page est minimale ; pour éviter des concaténations en série, on a appliqué plusieurs fois la fonction successivement, ce qui revient au même.

Ne pas oublier de dire que le document est terminé :

fenPop.document.close(); Cette clause est indispensable : si l'on n'a pas mis les balises HTML au complet, la fonction s'exécute mal ou pas du tout ; en plus, le **close()** indique à javascript qu'il n'y a plus rien à attendre, et restitue le curseur par défaut.

C'est presque ça, à deux inconvénients près :

- l'image n'est pas exactement à sa place (liseré en haut et à gauche) ;
- elle reste à l'affichage même si on ne la voit plus si on revient à la page d'appel.

1.3. troisième étape : le fignolage de la fenêtre (`html11tp0c.htlm`)

Le premier inconvénient provient du fait que la balise **body** définit par défaut un espace à une dizaine

de pixels des bords : pour utiliser correctement la balise **body**, il faut redéfinir sa marge dans une feuille de style (**margin:0;**).

Le second inconvénient est souvent réglé par l'invocation de deux événements : on fait disparaître la fenêtre soit lorsqu'elle perd le focus (clic en dehors de l'image), soit si on la clique (on peut même mettre un **title**…). Évidemment, il vaut mieux éviter de cliquer dans la vignette !

```
  fenPop.document.write("<body style='margin:0;' onBlur='window.close();'>");
  fenPop.document.write("<img src='"+adresse+"' onClick='window.close();'/>");
```

* **attention** : la commande est bien **'window.close();'** ce qui signifie : la fenêtre qui a le focus doit se fermer elle-même, ce qui n'est possible que si la fenêtre a été ouverte par une autre. Une fenêtre appelée avec une url qui été écrite dans le navigateur ne peut pas se fermer ainsi.

2. une fenêtre modale.

2.1. avertissement.

Une fenêtre modale a la bonne idée de toujours avoir le focus, et donc de toujours être au-dessus des autres ; sauf si on clique le bouton, obligatoirement présent, qui sert à la fermer (on peut évidemment fignoler la sortie, par exemple en ayant plusieurs boutons de rôles divers). La fenêtre **alert** est une fenêtre modale.

On pourrait penser qu'à chaque fois que la fenêtre reçoit la commande d'événement **onBlur**, celle-ci soit annihilée en redonnant le focus à la fenêtre modale. Mais javascript interdit ce processus (il conduirait à la perte de maîtrise de l'ordinateur si une personne mal intentionnée utilisait cette forme modale, sans mettre de bouton de fermeture. Ce qui n'empêche pas cette pseudo-solution d'être un peu partout sur les sites consacrés à javascript. Sauf la solution javascript spécifique à I.E. (fonction **showModalDialog()**) aucune solution simple ne fonctionne correctement sous les deux navigateurs de base, Firefox et I.E. Il y a celle que nous proposons ! Et encore son domaine d'action se limite à deux fenêtres : la fenêtre de création et la fenêtre *popup*.

2.2. les données.

Pour l'exercice, on va créer une fenêtre modale sur la page de la fiche 2. La page de la fiche 2 un peu modifiée s'appelle désormais **html11tp1.html,** sa feuille de style **html11tp1.css** et le contenu de la fenêtre *popup* **popmodale.html**. Le code javascript est inclus dans le fichier initial, ou le cas échéant, dans **popmodale.html**.

Le principe de fonctionnement comporte deux démarches essentielles :

- la première est que si la fenêtre *popup* perd le focus au profit de la fenêtre de création, celle-ci refuse le focus et le redonne à la fenêtre *popup*.

- la seconde est que le clic sur le bouton de fermeture ferme la fenêtre popup et redonne le focus à la fenêtre mère.

Une variable flag est mise à 1 lorsque la fenêtre *popup* est active, et 0 sinon.

2.3. création de la fenêtre popup.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
          "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
 <title>fenêtre modale</title>
  </head>
  <body style="margin:0;" >
  <img src="_vero.png" ><br/>
  <div style="width:100px;padding:0;margin-top:15px;margin-left:auto;
```

```
            margin-right:auto;">
   <input type="button" value="fermer" style="color:#a00000;"
    onClick="opener.flag=0; window.close();" />
  </div>
  </body>
</html>
```

* le DOCTYPE est nécessaire à la compréhension des styles w3c par I.E.

* **onClick="opener.flag=0; window.close();"**

  - on n'a pas créé de fonction spécifique ; en production, il aurait fallu le faire !

  - **opener** désigne la fenêtre qui a ouvert la *popup*. Cette fenêtre connaît donc la variable flag, qu'elle met à 0 lors de la fermeture de la *popup*.

  - **window** désigne la fenêtre actuellement active, et donc la *popup*.

Pour l'appeler, ne pas oublier :

```
<h4 id='main' onClick='modale();'>une image modale</h4>
```

2.4. Le script de la fenêtre principale :

```
<script type="text/javascript">
var lg=screen.width;
var ht=screen.height;
var fenPop;
var timer;

var flag=0;

function modale() {
   var fenPopLeft=(lg-570)/2;
   var fenPopTop=(ht-500)/2;

   fenPop= window.open("popmodale.html","","width=570,height=500,top="
       +fenPopTop+",left="+fenPopLeft);
   fenPop.focus();
   flag=1;
}

function controle() {
  if (flag) {
     window.blur();
     fenPop.focus();
  }
}
    </script>
```

* ici aussi, on confond les dimensions de l'image et celles qui servent pour la fenêtre ***popup***.

* **fenPop.focus();**
  flag=1;

En principe la fenêtre créée a le focus ; c'est pour le rappeler que l'on a associé focus et **flag=1;**

2.5. la fonction contrôle()

Dans le fichier de départ, on a pu constater la ligne suivante :

```
<body onFocus="controle();">
```

Au départ, comme **focus** est nul, il ne se passe rien ;

A chaque fois que pour une raison externe (clic sur la page..) la fenêtre tente de prendre le focus, la fonction **contrôle()** est appelée ; elle retire le contrôle à la page actuelle (**window.blur();**) et le redonne à la *popup* qui reste ainsi au-dessus de l'autre.

Ceci ne fonctionnerait pas si une page tierce prenait le contrôle. Notons aussi que si la fenêtre principale venait à disparaître, la fenêtre popup continuerait d'exister (il existe un paramètre javascript à ajouter à la liste des paramètres, **dependent=yes** pour éviter ce phénomène ; le fonctionnement n'est pas garanti sur les navigateurs classiques).

3. fenêtre confirm().

3.1. problématique.

Il existe une autre fenêtre modale que la fenêtre **alert()** et inscrite dans le javascript/HTML : la fenêtre **confirm()**. Alors que la fenêtre **alert()** ne retourne rien lorsqu'on la ferme, la fenêtre **confirm()** retourne un booléen (*vrai, faux*) que l'on peut exploiter dans un script. C'est le type de fenêtre *popup* qui demande "*voulez-vous continuer – oui / non -*".

On propose l'exercice suivant : tous les sites officiels doivent désormais être adaptés au handicap visuel ; le plus souvent, en remplaçant un style à caractères normaux par une style à gros caractères et en remplaçant les images par un simple message de description. On propose de réaliser un schéma de page où une fenêtre **confirm()** permet ce choix.

3.2. principes.

On appelle **normal** et **handicap** les deux états possibles ; pour le cas normal, la page affiche "normalement" ses éléments : caractères de dimension normale, image. Sinon, les caractères sont de grande dimension, et l'image est remplacée par un message descriptif.

Le choix est assuré par un fenêtre **confirm()**. Selon le résultat de la confirmation, un flag est posé signifiant suivant sa valeur "normal" ou "handicap" et la feuille de style convenable est chargée pour la page.

On va donc créer deux feuilles de style (simples pour cet exemple). Et selon la réponse au **confirm()**, c'est l'une ou l'autre qui est chargée. Pour gérer l'image, on va utiliser la propriété qu'on les conteneurs et les images d'être visibles ou pas (**display:none**; ou **display:block;**).

3.3. la présentation "normale"

La Grande Guerre.

D'octobre 1914 à la contre-offensive de mai 1915, la ligne de front passe à proximité immédiate du Mont Saint Eloi. Les Tours constituent un point d'observation privilégié sur le champ de la Première Bataille de l'Artois (1er au 26 octobre 1914) et la ligne de stabilisation (Arras, Lorette, Carency, Souchez, Ablain Saint Nazaire, Vimy, Lens) ; elles sont une cible privilégiée de l'artillerie allemande, qui les ampute de neuf mètres, les laissant ainsi que leur environnement à l'état de ruines.

La deuxième bataille de l'Artois engagée par le Général Foch en mai-juin 1915 repousse la ligne de Front à la crête de Vimy. La Troisième Bataille de l'Artois en septembre échoue et la crête ne sera prise qu'en avril 1917.

Le village connaît alors la vie des zones à proximité immédiate du front : dernier lieu de cantonnement des troupes avant les offensives ; lieu d'équipement pour des attaques comme celles de Vimy par les armées canadiennes, qui demandent une longue préparation logistique ; poste d'observation de la ligne de front par les aérostats et les avions ; et surtout, hôpital de l'arrière immédiat du front. Le cimetière militaire d'Ecoivres en reste le témoin, qui regroupe surtout des soldats britanniques et français mais aussi des soltats des armées alliées venus des cinq parties du monde.

Mont-Saint-Eloi se voit décerner la Croix de Guerre à l'issue des hostilités.

et la présentation "handicap" :

La Grande Guerre.

D'octobre 1914 à la contre-offensive de mai 1915, la ligne de front passe à proximité immédiate du Mont Saint Eloi. Les Tours constituent un point d'observation privilégié sur le champ de la Première Bataille de l'Artois (1er au 26 octobre 1914) et la ligne de stabilisation (Arras, Lorette, Carency, Souchez, Ablain Saint Nazaire, Vimy, Lens) ; elles sont une cible privilégiée de l'artillerie allemande, qui les ampute de neuf mètres, les laissant ainsi que leur environnement à l'état de ruines.

image : un vieux défenseur

La deuxième bataille de l'Artois engagée par le Général Foch en mai-juin 1915 repousse la ligne de Front à la crête de Vimy. La Troisième Bataille de l'Artois en septembre échoue et la crête ne sera prise qu'en avril 1917.

Le village connaît alors la vie des zones à proximité immédiate du front : dernier lieu de cantonnement des troupes avant les offensives ; lieu d'équipement pour des attaques comme celles de Vimy par les armées canadiennes, qui demandent une longue préparation logistique ; poste d'observation de la ligne de front par les aérostats et les

3.4. les feuilles de style

`/* CSS Document normal */`
 	
`div#page {`
	`/* CSS Document handicap */`
	
	`div#page {`

```
  width:800px;
  margin-left:auto;
  margin-right:auto;
  border: 2px  dashed green;
  padding:10px;
}

h2 {
  font-family:serif;
  font-size:30px;
  font-weight:bold;
  text-align:center;
}

p {
  font-family:serif;
  font-size:23px;
  text-align:justify;
}


#imgFloat {
  width:239px;
  height:284px;
  float:left;
  border: 1px solid black;
  margin-right:10px;
}

#imageOui {
  display:block;
}

#imageNon {
  display:none;
}
```

```
  margin:15px;
}



h2 {
  font-family:sans-serif;
  font-size:40px;
  font-weight:bold;
  text-align:center;
}

p {
  font-family:sansserif;
  font-size:30px;
  font-weight:bold;
  text-align:justify;
}

#imgFloat {
  width:239px;
  height:284px;
  float:left;
  border: 1px solid black;
  margin-right:10px;
}

#imageOui {
  display:none;
}

#imageNon {
  font-style:italic;
  display:block;
}
```

3.5. le fichier HTML : `html11tp2.html`

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
          "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
  <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
  <title>la popup confirm()</title>

  <script type="text/javascript">
  var flag = confirm("affichage par défaut : normal\n\n"
```

```
    +"confirmer par O.K. l'affichage NORMAL\n\n"
    +"pour HANDICAP : touche Annuler\n\n");
  if (flag) {
  document.write("<style type='text/css'>@import url(html11tp2n.css);</style>");
  }
  else {
  document.write("<style type='text/css'>@import url(html11tp2h.css);</style>");
  }

  </script>
   </head>
  <body>
    <div id="page">
      <h2>La Grande Guerre.</h2>

      <p> D'octobre 1914 à la contre-offensive de mai 1915, la ligne de front
passe à proximité immédiate du Mont Saint Eloi. Les Tours constituent un point
d'observation privilégié sur le champ de la Première Bataille de l'Artois (1er
au 26 octobre 1914) et la ligne de stabilisation (Arras, Lorette, Carency,
Souchez, Ablain Saint Nazaire, Vimy, Lens) ; elles sont une cible privilégiée de
l'artillerie allemande, qui les ampute de neuf mètres, les laissant ainsi que
leur environnement à l'état de ruines.
      </p>

      <div id="imgFloat">
        <img id="imageOui" src="_vieux.jpg" />
        <h2 id="imageNon">image : <br /> un vieux défenseur</h2>
      </div>

      <p> La deuxième bataille de l'Artois engagée par le Général Foch en mai-
juin 1915 repousse la ligne de Front à la crête de Vimy. La Troisième Bataille
de l'Artois en septembre échoue et la crête ne sera prise qu'en avril 1917.
      </p>

      <p> Le village connaît alors la vie des zones à proximité immédiate du
front : dernier lieu de cantonnement des troupes avant les offensives ; lieu
d'équipement pour des attaques comme celles de Vimy par les armées canadiennes,
qui demandent une longue préparation logistique ; poste d'observation de la
ligne de front par les aérostats et les avions ; et surtout, hôpital de
l'arrière immédiat du front. Le cimetière militaire d'Ecoivres en reste le
témoin, qui regroupe surtout des soldats britanniques et français mais aussi des
soltats des armées alliées venus des cinq parties du monde.
      </p>

      <p> Mont-Saint-Eloi se voit décerner la Croix de Guerre à l'issue des
hostilités.
      </p>
</div>
  </body>
</html>
```

```
 * var flag = confirm("affichage par défaut : normal\n\n"
     +"confirmer par O.K. l'affichage NORMAL\n\n"
```

```
    +"pour HANDICAP : touche Annuler\n\n");
```

La fonction/popup **`confirm()`** accepte comme argument un texte ou le passage à la ligne est **`\n`**. Elle a deux boutons que **l'on ne peut changer** : **`OK`** et **`Annuler`**. Si **`OK`** est cliqué, elle retourne **`true`** (c'est la valeur que prend la variable flag. Si **`Annuler`** est cliqué, c'est false qui est renvoyé.

* On peut alors, de façon classique écrire dans le fichier HTML la commande de chargement de la feuille de style qui convient.

```
if (flag) { document.write("<style type='text/css'>@import
            url(html11tp2n.css);</style>");}
  else { document.write("<style type='text/css'>@import
            url(html11tp2h.css);</style>"); }
```

On a choisi ici le chargement par **`url`** au lieu de **`link`** ; c'est sans conséquence. En guise d'exercice, on peut faire le chargement par **`link`**.

Tout ce qui concerne les conteneurs de texte doit se dérouler de façon classique. Il n'en est pas de même pour l'image.

```
*  <div id="imgFloat">
```

On constitue un conteneur flottant, prédimensionné, qui contient à la fois l'image et le texte de remplacement (***image : un vieux défenseur***). Mais selon la feuille de style, l'affichage (**`display`**) n'est pas le même : l'image est affichée comme **`block`** pour le **`normal`**, cachée (**`hidden`**) pour le **`handicap`** ; c'est le contraire pour le conteneur de texte. Rappelons que l'image peut être indifféremment affichée (**`display`**) comme **`block`** ou **`inline,`** puisqu'elle est seule dans son conteneur. Et de plus, le **`display:none`** ne fait aucune réservation d'espace, contrairement à la commande **`visibility`** qui réserve l'espace et n'aurait pu convenir.

12 récupérer les balises pour le javascript

tutoriel HTML

fiche 12

introduction.

Lorsqu'un fichier HTML est chargé dans le navigateur, il est mis sous une forme (un modèle) qui lui permet de prendre en compte les feuilles de style, les scripts javascript etc. Le format interne permet à travers des scripts d'avoir accès à l'information et de la modifier : par exemple, un script javascript peut avoir pour fonction de changer un élément de style (une couleur de fond), récupérer la valeur d'une balise **`input`** pour la contrôler (*a-t-elle bien été remplie ?*). On a vu précédemment que l'un des souhaits du programmeur pouvait être de se charger de la validation d'un formulaire sans utiliser la balise **`submit`**, pour contrôler les données envoyées ou pour éviter une validation prématurée (touche entrée dan un **`input`** de type **`text`** ou **`password`**).

Le **modèle interne** du navigateur est fondé sur la représentation des balises. Quelle que soit l'action que l'on veut réaliser, il faut donc récupérer la représentation de la balise en jeu. Tant pour des raisons historiques que formelles, cette récupération peut prendre plusieurs formes dont certaines son désuètes (la récupération par nom de balise).

1. Récupération par identificateur : getElementById().

Depuis le début du tutoriel, nous avons montré l'importance le l'identificateur (**`id="monIdentificateur"`**), qui est unique dans la page, et sert donc bien à identifier les balises **de façon univoque**, contrairement aux noms, par exemple, un même nom pouvant/devant être donné à plusieurs balises, ou aux tags (type de la balise : **`body, input`** etc), où il faut passer par des tableaux javascript (on peut récupérer tous les éléments de même tag, le résultat est stocké dans un tableau indexé). Aussi, nous ne garderons que le système de reconnaissance de la balise par son identificateur : pour certains, cela peut paraître un peu lourd, mais c'est sûr, facile à mettre en œuvre, rapide d'exécution et facile à contrôler.

Pour récupérer une balise (on dit plutôt un **`Element`** dans ce cas), il faut évoquer la fenêtre où l'on se trouve **`(window`** en général, sauf pour les *popup* crées ; on a le droit d'omettre l'identificateur **`window`**), le document (qui s'appelle toujours **`document`**), puis demander au document par la fonction **`getElementById()`** de restituer l'**`Element`** dont le nom est passé en argument.

Par exemple voici une balise :
`<p id="monHistoire">bla bla bla bla bla bla bla bla bla bla bla bla</p>`

Pour la récupérer dans un script on écrit :
`window.document.getElementById("monHistoire")`

ou plus simplement :
`document.getElementById("monHistoire")`

On peut alors soit placer cet **`Element`** dans une variable javascript, ou travailler directement dessus, soit avec ses propriétés (son style par exemple), soit ses fonctions (**`submit()`** pour un formulaire).

2. Un exemple d'application. (`html12tp0.html`)

entête

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
          "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
  <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
```

```
   <title>getElementById()</title>.
```

le script est placé en fin d'étude.

l'écran de départ.

```
</head>
  <body>
  <h1 id="monTitre">essai de GetElementById()</h1>
  <h2 style="text-decoration:underline;display:inline;">texte blaBla : </h2>
  <p id="monBlaBla">bla bla bla bla bla bla bla bla bla</p>
  <h2 style="text-decoration:underline;display:inline;">taper le texte :
  </h2> 
  <input id="maSaisie" type="text" size=20 maxLength=35 value="" /><h2><br />
  cliquer le bouton pour  appeler le script : 
  <input  type="button" value="validation" onClick="gebi();" /><br />
  <hr></h2>
```

On distingue 4 zones dans l'écran :

* le titre, dans une balise **h1** standard, identifiée **"monTitre"**
* le texte bla bla …. dans une balise **p** standard identifiée **"monBlaBla"**
* la zone de saisie de texte, sans nom, identifiée **"maSaisie"**
* le bouton de validation de type **input/button**

l'écran après exécution du script

```
  <div id="result" style="display:none;">
    <h2 style="text-decoration:underline;">le blaBla : </h2>
    <h2 id="monBlaBlaInn" style="color:blue;"> </h2>
    <h2 style="text-decoration:underline;">la valeur input : </h2>
    <h2 id="maSaisieVal" style="color:red;"> </h2>
  </div>
</body>
</html>
```

Après validation :

* le titre est passé en rouge ;

* après la zone de saisie est apparue une nouvelle zone identifiée **"result"**, avec deux éléments :

* en bleu, dans une balise h2 identifiée **"monBlaBlaInn"**, le texte bla bla …

* en rouge, dans une balise h2 identifiée **"maSaisieVal"** la valeur saisie en **input/text**

le script.

```
<script type="text/javascript">
  function gebi () {
    var imp = document.getElementById("maSaisie").value;
    if (imp=="") {
      alert ("la balise input n'a pas été renseignée");
      return;
    }
    document.getElementById("monTitre").style.color="red";

    var blaBla=document.getElementById("monBlaBla").innerHTML;
    document.getElementById("monBlaBlaInn").innerHTML=blaBla;

     document.getElementById("maSaisieVal").innerHTML=imp;
     document.getElementById("result").style.display="block";
    }
```

```
</script>
```

Reste à voir le fonctionnement de la fonction activée par le bouton de façon classique.

* **document.getElementById("maSaisie")** les expressions ayant cette structure récupèrent l'**Element** identifié (ici, la balise **input/text**). Le terme qui suit soit désigne une propriété de l'élément (qui peut elle aussi avoir des propriétés), soit une fonction de l'élément (on la reconnaît à la présence de parenthèses, avec parfois des paramètres). On connaît bien les propriétés de la balise **input/text : name, value, size, maxlength**. La différence avec l'écriture HTML est qu'ici, la casse est importante.

Dans l'exemple cité, la propriété **value** (la saisie) est placées dans une variable javascript appelée **imp,** nom complètement arbitraire. On contrôle que la rubrique a bien été renseignée (**if (imp==""**) { (**attention au double signe égale**). Si la valeur est restée vide, la fonction **alert()** est invoquée et la fonction arrêtée (**return;**).

* **document.getElementById("monTitre").style.color="red";**

On a ici affaire à une balise conteneur, **h1**, qui a donc un **style**, qui lui-même a la propriété **color**. **style** est la propriété "feuille de style", dont on sait qu'un de ses attributs est **color**. Reste à affecter comme pour une feuille de style classique la valeur de la couleur. La syntaxe est spécifique à javascript : **=** *au lieu des* **:** *; valeur en chaîne de caractères* ***quotés****, alors que dans la feuille de style, on se contente d'écrire la référence de la couleur.*

*** var blaBla=document.getElementById("monBlaBla").innerHTML;**

Pour les balise avec contenu, la propriété **innerHTML** désigne la valeur du contenu ; on en fait ensuite ce que l'on veut : la modifier, l'affecter à une autre balise (en javascript), l'utiliser comme paramètre de fonction etc. Ici, on l'affecte à une autre balise d'affichage :

document.getElementById("maSaisieVal").innerHTML=imp;

Tout comme on affecte la valeur **imp** :

document.getElementById("maSaisieVal").innerHTML=imp;

* mais tout ceci a été fait dans un conteneur **div** non affiché (**display:none;**) Il reste à commander l'affichage de ce conteneur, c'est-à-dire modifier une propriété de style :

document.getElementById("result").style.display="block";

Autant la propriété **innerHTML** est peu employée, autant les propriété de formulaire et les styles sont des propriétés importantes dans l'utilisation élémentaire de javascript. La fiche qui suit leur seront consacrées.

13 formulaires, feuilles de style, images

tutoriel HTML

fiche 13

introduction.

On a esquissé dans la page précédente la façon de passer d'un formulaire à javascript, et comment on utilisait javascript pour imposer une propriété de feuille de style. Dans l'esprit pratique qui préside à ce tutoriel, on va examiner des cas concrets tels qui se présentent le plus souvent.

note. Comme on utilise systématiquement la méthode de l'identificateur (**`getElementById()`**), on peut se trouver en divergence importante avec des méthodes par ailleurs fort respectables, non désuète, voire d'avant-garde en matière d'articulation HTML/javascript/CSS. Les seules méthodes à rejeter pour les formulaires sont celles fondées sur l'attribut **`name`** pour la reconnaissance des balises : l'attribut **`name`** est à réserver pour les couples nom/valeur des balises de saisie. Il faut noter que rien n'interdit que la valeur de **`id`** et celle de **`name`** soient les mêmes : il n'y a pas d'ambiguïté possible.

1. la balise form.

Supposons qu'une balise **`form`** soit réduite à : **`<form id="maBaliseForm">.`** Il n'y a là rien de non conforme aux spécifications HTML. En cas d'absence de **`method`** c'est par défaut la méthode "**`get`**" qui est employée. Mais pour un usage strict de la balise, il faut au moins renseigner l'attribut **`action`** et l'attribut **`method`** avant toute validation (**notes** : un **`input/image`** remplace le bouton de validation ; **`action=""`** rappelle la page actuelle). Il faut rappeler que la balise **`form`** supporte l'événement **`onSubmit`**, et le code javascript appelé par cet événement est exécuté avant la soumission du formulaire, qui peut d'ailleurs être interrompue par le code javascript associé à **`onSubmit`** (par exemple à l'issue d'un contrôle non satisfaisant sur un élément du formulaire).

1.1. rappel sur le formulaire.

Supposons que l'on ait le formulaire suivant (exemple **`html13tp0a.html`**) :

```
<form id="maBaliseForm">
  <h2>cliquer l'image</h2>
    <input type="hidden" name="nom" value="Jean Mercier">
    <input type="image" src="_aster.png" name="aster" value="code_png">
</form>
```

Le clic sur l'image produit comme url :
`file:///D:/_html/code/html13tp0a.html?nom=Jean+Mercier&aster.x=86&aster.y=81&aster=code_png`

La page actuelle :
`file:///D:/_html/code/html13tp0a.html`

Les paramètres transmis par le méthode get :
`nom=Jean+Mercier&aster.x=86&aster.y=81&aster=code_png`

Cette méthode n'est pas recommandée.

1.2. le formulaire modifié par javascript. (dans `html13tp0b.html`)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
          "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
```

```
  <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
  <title>formulaire</title>
  <script type="text/javascript">
    function init() {
      document.getElementById("maBaliseForm").action="html13tp0br.html";
      document.getElementById("maBaliseForm").method="post";
     /* document.getElementById("maBaliseForm").method="get"; */
    }
  </script>
  </head>
  <body onLoad="init();">
  <h1> essai de formulaire </h1>
  <form id="maBaliseForm">
  <h2>cliquer l'image</h2>
    <input type="hidden" name="nom" value="Jean Mercier" />
    <input type="image" src="_aster.png" name="aster" value="png" />
  </form>

  </body>
</html>
```

le fichier appelé est : **`html13tp0br.html`**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
          "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
  <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
  </head>
  <body>
    <h2> page appelée par html13tp0b.html </h2>
    <h2> l'url est : </h2>
    <h2 style="font-family:monospace">
    <script type="text/javascript">document.write(location.href)</script></h2>
  </body>
</html>
```

* rappel : **`location.href`** désigne l'url complet du fichier.

Avec la méthode **`post`**, aucun argument n'est passé par l'url.

Avec la méthode **`get`**, la transmission des arguments est complète par l'url, dans la limite du possible (128 caractères en tout).

2. le problème de la soumission.

2.1. soumission différée.

Le formulaire est soumis (en principe, envoyé au serveur) par la balise **`input/submit`** ou la balise **`input/image`**. Mais la présence de l'un de ces deux éléments dans le formulaire fait qu'avec les balises **`input/text`** ou **`input/password`** la validation se fait si on tape la touche entrée. C'est un gros inconvénient lorsque certaines rubriques de saisie doivent obligatoirement être enseignées (cas du **`login/password`** par exemple) et qu'un contrôle préalable côté client s'impose.

Un autre problème lui est lié qui est celui de la vraisemblance de la saisie : saisie trop courte pour être un login ou un mot de passe, adresse mail sans le @ obligatoire et unique. Il peut s'avérer utile qu'un minimum de contrôle soit fait avant validation, voire qu'il y ait une double validation.

2.2. soumission sans balise input/submit. `html13tp1.html`

Voici un fichier de formulaire qui ne comporte pas de balise **`submit`** (ou d'**`input/image`**, ce qui est équivalent). C'est un bouton ordinaire qui réalise la soumission ; on doit constater qu'il faut obligatoirement passer par ce bouton pour réaliser la soumission (pas de touche entrée sur la balise **`input/text`** ou **`input/password`**).

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
          "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
  <title>formulaire et contrôle</title>
  <script type="text/javascript">

  function valider () {
    document.getElementById("formEnvoi").submit();
  }

  </script>

</head>
<body>
  <h1> submit différé </h1>

  <form id="formEnvoi" action="" method="get" >
    <p>entrer le login: 
      <input type="text" size="20" maxlength="20" name="login" value="" />
    </p>
    <p>entrer le mot de passe: 
      <input type="password" size="20" maxlength="20" name="pass" value="" />
    </p>
    <p>entrer votre mail: 
      <input type="text" size="30" maxlength="30" name="e-mail" value="" />
    </p>
    <p> contr&ocirc;lez vos données: 
       <input type="button" value="contr&ocirc;le" onClick="valider();"/>
    </p>
  </form>
  <hr />

</body>
</html>
```

* **`document.getElementById("formEnvoi").submit();`**

C'est l'appel de soumission, équivalente au bouton **`input/submit`** à la différence près qu'il ne peut y avoir de soumission intempestive avec les balises **`input/text`** ou **`input/passsword`**.

* contrôler l'url d'appel qui doit ressembler à :

file:///D:/_html/code/html13tp1.html?login=jean+mercier&pass=6022&e-mail=jean.mercier%40free.fr

le & sépare les saisies ; %40 est le @ ; le + est l'espace

2.3. un exercice de saisie différée. `html13tp2.html`

On se propose de saisir une login (6 caractères au minimum), un mot de passe (également 6 caractères au moins) et une adresse mail (contrôle minimum : le @ est présent). **La saisie est indépendante d'un formulaire**. Lorsque les contrôles sont positifs, on présente les saisies globalement et on demande de valider. On doit donc copier les saisies dans un nouveau formulaire (avec balises **input/hidden**).

On pourrait bien entendu contrôler beaucoup d'autres choses : pas d'espace dans un mot de passe, rien que des caractères autorisés dans l'adresse mail etc… Cela complexifie les contrôles mais ne change rien à la structure.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
          "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
  <title>formulaire et contrôle</title>
  <script type="text/javascript">


  function effacer() {
    document.getElementById("affichage").style.display="none";
  }

  function controler () {
    /* le login */
    var vlog = document.getElementById("log").value;
      if (vlog.length <6) {
        alert("le login est incorrect");
        return;
      }
      /* le mot de passe */
      var vpass = document.getElementById("pass").value;
      if (vpass.length <6) {
        alert("le mot de passe est incorrect");
        return;
      }
      /* le mail */
      var vmail = document.getElementById("mail").value;
      if (vmail.length <8) {
        alert("le e-mail est incorrect1");
        return;
      }
      var p = vmail.indexOf("@");
      if (p<4 || p>(vmail.length-3)) {
          alert("le e-mail est incorrect2");
          return;
```

```
      }
  /* tout est correct faire les copies */
    document.getElementById("flog").value=vlog;
    document.getElementById("fpass").value=vpass;
    document.getElementById("fmail").value=vmail;
    document.getElementById("innLog").innerHTML=vlog;
    document.getElementById("innPass").innerHTML=vpass;
    document.getElementById("innMail").innerHTML=vmail;
    document.getElementById("affichage").style.display="block";
  }

  </script>

  <style type="text/CSS">
  .envoi {
    font-size:25px;
    color:#a00000;
    font-weight:bold;
  }

  div#affichage {
    display:none;
  }
  </style>

</head>
<body>
  <h1> contrôle de formulaire </h1>
  <!-- ------------------- -->
  <!-- saisie des éléments -->
  <!-- ------------------- -->
  <p>entrer le login: 
     <input id="log" type="text" size="20" maxlength="20"
            value="" onFocus="effacer();" />
  </p>
  <p>entrer le mot de passe: 
     <input id="pass" type="password" size="20" maxlength="20"
            value="" onFocus="effacer();" />
  </p>
  <p>entrer votre mail: 
     <input id="mail"type="text" size="30" maxlength="30"
            value=""   onFocus="effacer();" /></p>
  <p> contr&ocirc;lez vos données: 
     <input type="button" value="contr&ocirc;le" onClick="controler();"/>
  </p>

  <hr />
  <!-- ----------------------------------- -->
  <!-- affichage et validation du formulaire -->
  <!-- ----------------------------------- -->
```

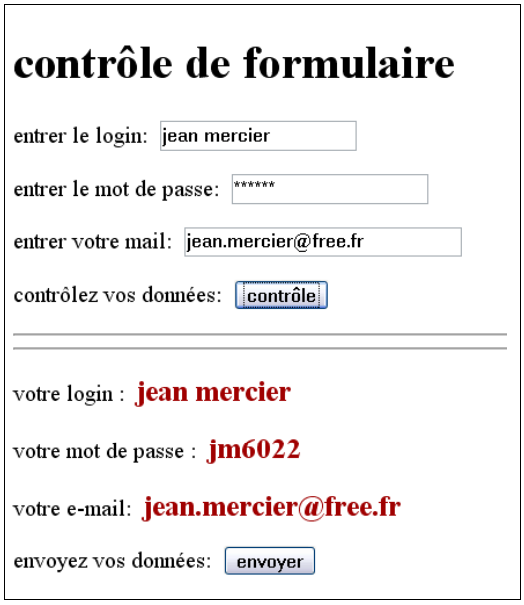
```
  <hr />
  <div id="affichage">

    <form id="envoi" action="" method="get" >
      <input id="flog" type="hidden" name="login" value="">
      <p>votre login : 
        <span id="innLog" class="envoi"></span>
      </p>

      <input id="fpass"type="hidden" name="password" value="">
      <p>votre mot de passe : 
        <span id="innPass" class="envoi"></span>
      </p>

      <input id="fmail" type="hidden" name="email" value="">
      <p>votre e-mail: 
        <span id="innMail" class="envoi"></span>
      </p>

      <p> envoyez vos données: 
        <input type="submit" value="envoyer" />
      </p>
    </form>
  </div
</body>
</html>
```



quelques règles sur les formulaires

Une fois identifiée la balise par **`document.getElementById("monId")`** on peut changer les attributs classiques en leur affectant une valeur convenable ; la règle à retenir est que le nom de l'attribut doit être en minuscules pour se conformer aux règles actuelles. La valeur affectée peut être une chaîne de caractères, un nombre ou un booléen.

Exemple :

la balise : **`<input id="cbx" type="checkbox" checked /> boîte à cocher`**

script javascript : **`document.getElementById("cbx").checked=false;`**

En pratique, c'est l'attribut **`value`** qui est le plus étudié dans les scripts javascript, soit pour contrôler une valeur, soit pour la modifier.

quelques règles sur les styles

On ne peut que modifier un élément de style à la fois sur une balise identifiée (l'attribut **`class`** n'est pas pris en compte) ; de même, on n'a accès en javascript qu'aux éléments de style définis par javascript, pas à ceux définis dans les feuilles de style. Ce qui n'empêche pas de modifier un élément déjà défini dans une feuille de style : la balise prend en compte la modification, la feuille de style reste inchangée.

En matière de règles de passage à javascript, on retiendra :

- si un élément de feuille de style comporte un tiret , on le supprime et on remplace le caractère suivant par une majuscule.

  `background-color` devient backgroundColor ;

  `font-family` devient **`fontFamily.`**

- les affectations sont des chaînes et sont identiques à celles du format CSS :

  `width:10px;` devient **`style.width="10px";`**

  `display:block;` devient **`style.display="block";`**

  `vertical-align="text-top";` devient **`style.verticalAlign="text-top";`**

  `font-weight=900;` devient **`.style.fontWeight="900";`**

  `color:rgb(56,180,35);` devient **`style.color="rgb(56,180,35);"`**

  `background-color:#00ffff; devient style.backgroundColor="#00ffff";`

3. chargement d'image.

3.1. les deux types de problèmes.

Javascript permet de traiter deux types de problèmes courants en HTML :

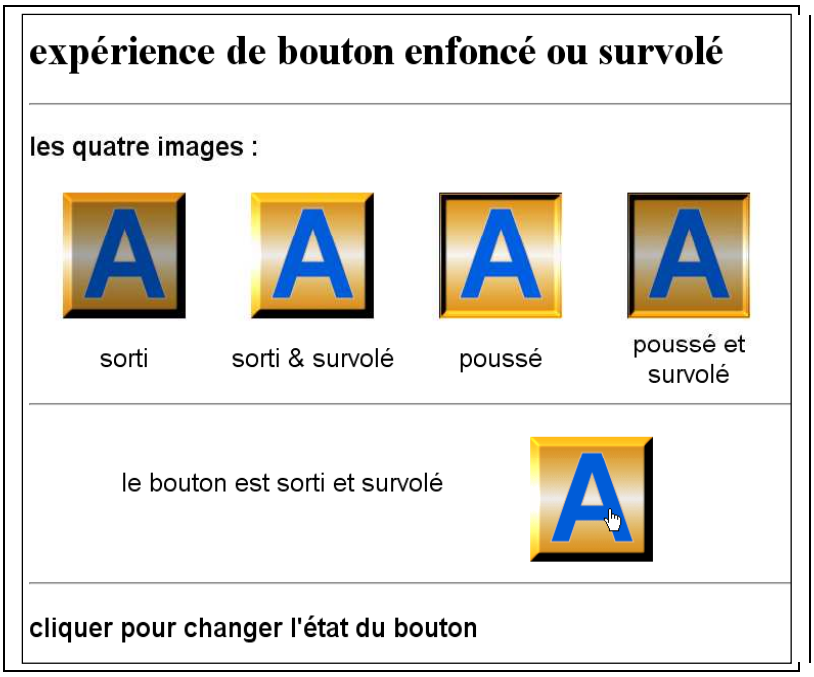
- le premier problème est lié **au temps de chargement** des images en HTML. Il est long ! Les images s'affichent par morceaux, et même en s'appuyant sur les techniques spécifiques (progressivité, entrelacement ; cf. voir les logiciels graphiques comme Gymp, PaintShopPro, Adobe Photoshop), cette situation peut s'avérer gênante (cas des diaporamas par exemple) : la solution est de prendre le temps de stocker toutes les images de la page avant leur affichage

- le second problème est plus "esthétique" : l'habitude est de signaler le survol d'un bouton par la souris (changement de bordure, de couleur de fond etc), et de signifier par une simulation graphique l'enfoncement d'un bouton. En fait, dans les deux cas, il s'agit de préparer plusieurs images de même dimensions pour la même balise, et d'afficher l'une ou l'autre des images. Le choix de l'image est commandé par la survenue d'un événement (survol, sortie, touche enfoncée…).

3.2. simulation d'un bouton survolé ou enfoncé.

Dans la partie médiane de l'illustration ci-dessous, on a figuré un bouton, accompagné d'un commentaire. Le bouton dispose de 4 états : sorti non survolé, sorti et survolé, enfoncé et survolé, enfoncé et non survolé. On a construit une image pour figurer chaque état (partie supérieure de l'illustration). Les images sont de mêmes dimensions, 120x120 pixels ; les boutons d'une application usuelle seraient plus petits !

La construction de la page est soumise aux exigences évoquées ci-dessus. Le préchargement des images en premier lieu (même si ici elles ne sont pas très lourdes). Puis l'affichage fonctionnel de la page (réalisé par a fonction **`init()`**).



3.3. image HTML et image javascript.

3.3.1. image en HTML.

On a vu ce qu'est une image définie par une balise HTML. L'attribut essentiel d'une image HTML est sa source (attribut **`src`**) ; si on n'explicite pas cet attribut, il n'y a pas d'image ! Si par un script javascript on change la source, c'est l'image de la nouvelle source qui est prise en compte immédiatement …

Que fait le navigateur HTML lorsqu'il rencontre cet attribut ? Il lance le chargement du fichier, et laisse **le système d'exploitation traiter la mise du contenu de l'image dans un cache prévu pour cela**. Le navigateur passe alors à la suite de l'analyse du fichier HTML.

Que peut-il alors se passer ?

> - le navigateur peut arriver en fin d'analyse sans que toutes les images soient dans le cache ; en général, le navigateur traite l'information au fur et à mesure qu'elle arrive ce qui donne un affichage erratique de la page. **La seule chose importante à savoir est que le navigateur attend que tout soit en place pour déclencher un événement onLoad.**
>
> - mais il se peut aussi que **l'image soit déjà dans le cache**. Le navigateur est agencé pour contrôler cette présence en comparant l'url des fichiers du cache et celle du fichier demandé ; ainsi la même image copiée à des adresses différentes est considérée comme deux images distinctes. Dans le cas où l'image est dans le cache le système n'a rien à faire, et le navigateur s'y retrouve quand même. Ceci représente évidemment un gain de temps important sur le chargement.

Dans tous les cas, les éléments de style de l'image sont respectés : dimensions (**`width, height`**),

bordures, marges, display (par défaut, c'est **inline**), float.

3.3.2. Image en Javascript.

Il existe aussi un type image en javascript, qu'il vaut mieux **ne pas confondre** avec le type HTML. Pour obtenir un image javascript, il faut d'abord la créer par un opérateur javascript, et la placer dans une variable. Les champs classiques relatifs à l'image javascript sont les dimensions (qui s'appellent aussi **width** et **height**), et la source (qui s'appelle aussi **src**). Le schéma classique de création est :

```
monImage = new Image ; /* créer une image par l'opérateur new */
monImage.src ="url de l'image"; /* source de l'image */
```

L'effet de l'affectation (**monImage.src ="*url de l'image*";**) est le même que pour une image HTML : **le code de l'image est chargé dans le même cache**. S'il a besoin de l'image, le navigateur peut utiliser le contenu du cache. Pour traduire ceci on parle de "pré-chargement de l'image".

Il y a un gros avantage esthétique au pré-chargement : déclencher l'affichage lors du **onLoad** conduit à un flashage de l'image, c'est-à-dire une mise à l'écran instantanée. L'usage est classique : on pré-charge l'image, et ensuite, par un script javascript, on identifie la source HTML et la source javascript. C'est devenu un simple jeu d'écriture.

3.4. le code. html13tp5.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
    <title>
      bouton enfoncé
    </title>

<script type="text/javascript" >

var flagSorti=true;

/* stockage des quatre images */
var imgBoutonSorti = new Image ;
    imgBoutonSorti.src="boutonSorti.gif";
var imgBoutonSortiOver = new Image ;
    imgBoutonSortiOver.src="boutonSortiOver.gif";
var imgBoutonPousse = new Image ;
    imgBoutonPousse.src="boutonPousse.gif";
var imgBoutonPousseOver = new Image ;
    imgBoutonPousseOver.src="boutonPousseOver.gif";

/* chargement des quatre commentaires */
var texteBoutonSorti ="le bouton est sorti";
var texteBoutonSortiOver ="le bouton est sorti et survolé";
var texteBoutonPousse ="le bouton est poussé";
var texteBoutonPousseOver ="le bouton est poussé et survolé";

/* chargement des variableiables images*/
var variableBouton;
var variableBoutonSorti; var variableBoutonSortiOver;
```

```
var variableBoutonPousse; var variableboutonPousseOver;
var variableComment;

function init() {
  variableBouton=document.getElementById("bouton");
  variableBouton.src = imgBoutonSorti.src ; /* identification */

  variableBoutonSorti=document.getElementById("boutonSorti");
  variableBoutonSorti.src = imgBoutonSorti.src ;

  variableBoutonSortiOver=document.getElementById("boutonSortiOver");
  variableBoutonSortiOver.src = imgBoutonSortiOver.src ;

  variableBoutonPousse=document.getElementById("boutonPousse");
  variableBoutonPousse.src = imgBoutonPousse.src ;

  variableboutonPousseOver=document.getElementById("boutonPousseOver");
  variableboutonPousseOver.src = imgBoutonPousseOver.src ;

  variableComment=document.getElementById("comment");
  variableComment.innerHTML = texteBoutonSorti;
}

function survol() {
  variableBouton.style.cursor="pointer";
  if (flagSorti) {
    variableBouton.src = imgBoutonSortiOver.src;
    variableComment.innerHTML = texteBoutonSortiOver;
  }
  else {
    variableBouton.src = imgBoutonPousseOver.src;
    variableComment.innerHTML = texteBoutonPousseOver;
  }
}

function dehors() {
  variableBouton.style.cursor="default";
  if (flagSorti) {
    variableBouton.src = imgBoutonSorti.src;
    variableComment.innerHTML = texteBoutonSorti;
  }
  else {
    variableBouton.src = imgBoutonPousse.src;
    variableComment.innerHTML = texteBoutonPousse;
  }
}

function cliquage () {
  if (flagSorti) {
```

```
    variableBouton.src = imgBoutonPousseOver.src ;
    variableComment.innerHTML =texteBoutonPousseOver;
  }
  else {
    variableBouton.src = imgBoutonSortiOver.src ;
    variableComment.innerHTML =texteBoutonSortiOver;
   }
  flagSorti = !flagSorti;
}

</script>
<style type="text/css">
td {
  width:180px;
  font-family:sans-serif;
  font-size:25px;
  text-align:center;
}
h3 {
  font-family:sans-serif;
  font-size:25px;
  font-weight:900;
}

#comment {
  font-family:sans-serif;
  font-size:25px;
  float:left;
  width:400px;
  margin-top:50px;
  margin-left:50px;
  overflow:hidden; /* par précaution*/

}

#bouton {
  float:left;
  margin-top:20px;
  margin-bottom:20px;
}

hr {
  margin-top:10px;
  margin-bottom:10px;
  clear:left;
}
</style>

  </head>
```

```
  <body onLoad="init();">
    <h1>expérience de bouton enfoncé ou survolé </h1>
    <hr>
    <h3>les quatre images : </h3>
    <table>
      <tr>
        <td>
          <img id="boutonSorti"/></td>
        <td>
          <img id="boutonSortiOver"/></td>
        <td>
          <img id="boutonPousse"/></td>
        <td>
          <img id="boutonPousseOver"/></td>
      </tr>
      <tr>
        <td>
          sorti</td>
        <td>
          sorti & survolé</td>
        <td>
          poussé</td>
        <td>
          poussé et survolé</td>
      </tr>
    </table>

    <hr>

      <dir id="comment"></dir>
      <img id="bouton"
           onMouseOver="survol();"
           onMouseOut="dehors();"
           onClick="cliquage();" />
      <hr>
      <h3>cliquer pour changer l'état du bouton </h3>
  </body>
</html>
```

* la première remarque à faire est que les images sont préchargées selon la méthode décrite ci-dessus. Les balise HTML d'image n'ont pas l'attribut **`src`** : cela ne gène en rien l'analyse HTML. Ce n'est que lorsque tout est prêt que l'affichage est déclenché par l'événement **`onLoad`** qui appelle la fonction **`init()`** qui elle, définit les sources, ce qui donne un affichage flashé, et la possibilité d'utiliser les images préchargées.

* dans la première partie du fichier HTML, on utilise un tableau, ce qui est logique, les images constituant des données, aussi bien que les textes. Si on préfère le flottement, prévoir un conteneur.

* dans la seconde partie (le bouton survolé), on a utilisé la méthode classique du **`float`**, avec un élément "**`comment`**" dimensionné, ce qui permet d'avoir une image fixe, même si le texte de l'élément "**`comment"`** est variable (et peut-être imprévisible selon les types d'ordinateurs ou de système

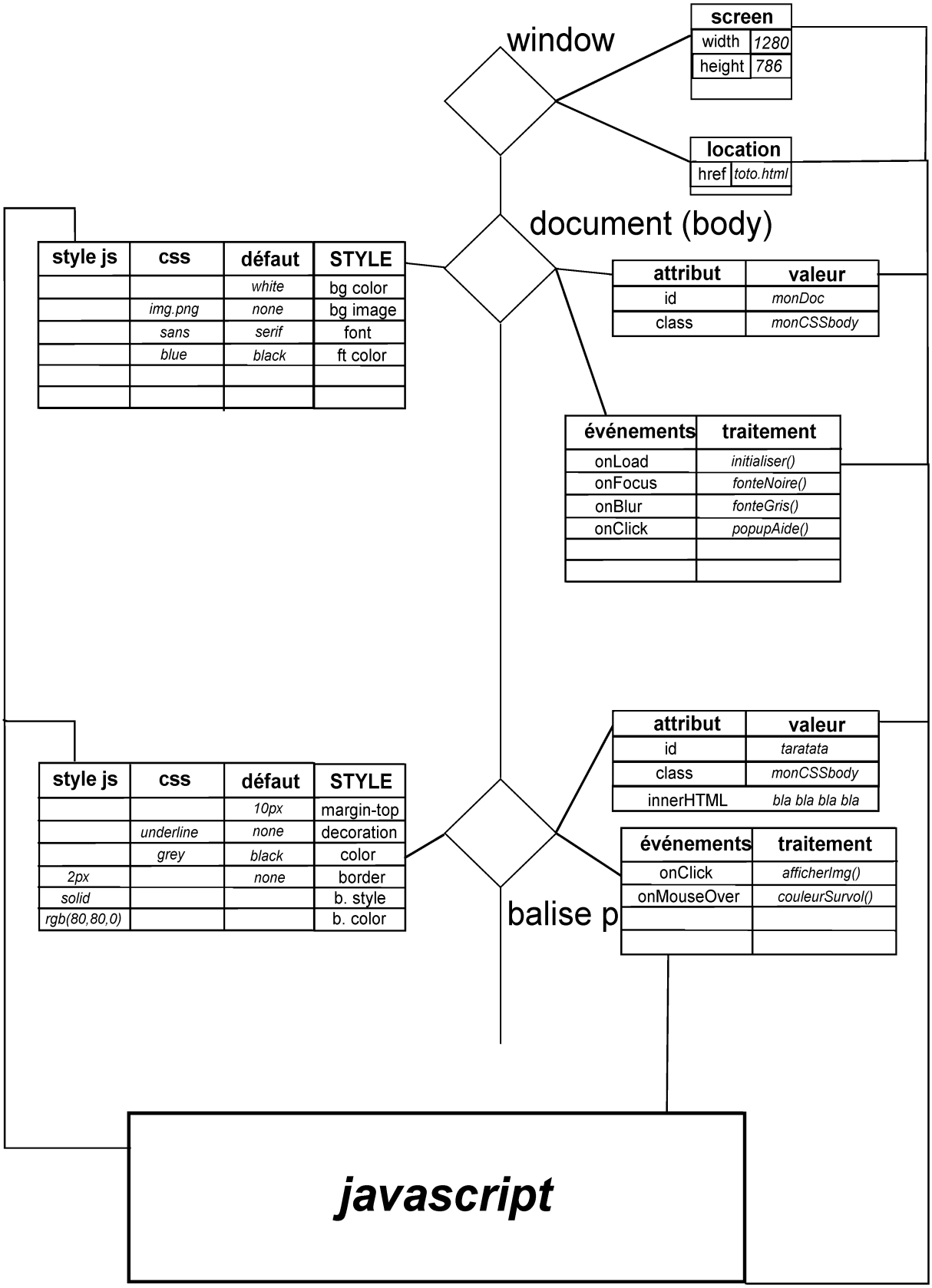
d'exploitation).

aide mémoire.

La page suivante comporte une image "aide-mémoire". Son rôle est de rappeler les niveaux d'intervention de javascript dans la représentation de la page HTML :

On a figuré l'élément racine **`window`**, avec les propriétés **`screen`**, **`location,`** **`document`**. Les propriétés sont symbolisées dans des petits tableau à deux colonnes (nom/valeur). On trouve ainsi **`window.location.href`** qui est la propriété url avec pou valeur "**`toto.html`**".

On a fait de même avec **`document`** et une balise **`p`**. On a fait figurer des propriétés événement (**`onload,`** **`onClick`** ...), de style, et de style créées par javascript. Il faut bien se rappeler que javascript n'a pas accès aux feuilles de style directement : il est capable de créer des propriétés de style qui se superposent à celles existantes ; alors qu'il a parfaitement accès aux propriétés comme **`screen`** (en lecture), **`location`** (en lecture/écriture), ou l'attribut **`innerHTML`** (lecture/écriture).


window
screen
width 1280
height 786
location
href toto.html
document (body)
style js | css | défaut | STYLE
white | bg color
img.png | none | bg image
sans | serif | font
blue | black | ft color
attribut | valeur
id | monDoc
class | monCSSbody
événements | traitement
onLoad | initialiser()
onFocus | fonteNoire()
onBlur | fonteGris()
onClick | popupAide()
attribut | valeur
id | taratata
class | monCSSbody
innerHTML | bla bla bla bla
style js | css | défaut | STYLE
10px | margin-top
underline | none | decoration
grey | black | color
2px | none | border
solid | b. style
rgb(80,80,0) | b. color
balise p
événements | traitement
onClick | afficherImg()
onMouseOver | couleurSurvol()
javascript

14 les dimensions, les couleurs

tutoriel HTML

fiche 14

1. dimensions et unités.

Jusque là, nous avons utilisé comme unité une valeur qui semble naturelle en HTML, le pixel, puisque le destinataire essentiel des fichiers HTML est l'écran.

Mais ce n'est pas la seule façon de mesurer ; en effet, dans la vie courante, on mesure avec le unités du système métrique. Dans le monde de la PAO, les unités sont spécifiques, le point, le pica etc. D'autre part, on peut aussi vouloir faire les évaluations en fonction des conteneurs ou de l'écran (exemple : zoomer une image pour occupe le maximum de l'écran) , dont les dimensions ne sont pas connues du programmeur, puisque ce sont des dimensions côté client.

Cela nous a conduit jusque-là, à définir pour les conteneurs de page des dimensions fixes qui s'appliqueront à tous les écrans. C'est malheureusement la principale façon de réaliser une mise en page précise au pixel près, à défaut d'être la plus élégante, ni la mieux adaptée à chaque cas d'écran.

Abr	mention	Signification	Exemples
pt	**absolue**	**point** **Unité de mesure typographique.** **1 point correspond à 1/72 de pouce industriel, soit environ 0.03528 mm**	font-size:12pt; line-height:14pt;
pc	**absolue**	**Pica.** **Unité de mesure typographique. 1 pica correspond à 12 points. Le pica était la dimension usuelle du caractère sur les machines à écrire.**	**line-height:1.2pc;** **font-size:1pc;**
in	**absolue**	**pouce (inch)** **Unité générale de longueur dans les pays anglo-saxons. 1 pouce industriel/métrique correspond à 2.54 centimètres exactement.**	**margin-left:1in;** **border-width:0.1in;**
mm	**absolue**	**millimètre**	**margin-bottom:10mm;** **width:70mm;**
cm	**absolue**	**centimètre**	**font-size:1cm;** **top:2.54cm;**
px	**absolue/relative**	**pixel** **Dépend de la densité en pixels (pitch ; le pitch de référence est 0.28 mm)du mode de sortie; relative à l'écran, mais absolue sur un écran donné.**	**margin-right:60px;**
em	**relative**	**par rapport à largeur du m minuscule dans la police actuelle de la balise.**	**font-size:1.2em;** **line-height:1.5em;**
ex	**relative**	**par rapport à la hauteur de la minuscule x de cet élément. Peu utilisé.**	**font-size:1.3ex;**
%	**relative**	**pourcentage** **relative à la taille de l'élément proprement-dit, ou à la taille de l'élément parent ou à un contexte plus général (écran, conteneur)**	**font-size:10pt;** **line-height:120%;** ***soit 12pt***

2. les couleurs.

2.1. les 16 couleurs de base

grey #808080 rgb(128,128,128)	black / #000000 rgb(0,0,0)
maroon #800000 rgb(128,0,0)	red #ff0000 rgb(256,0,0)
green #008000 rgb(0,128,0)	lime #00ff00 rgb(0,256,0)
olive #808000 rgb(128,128,0)	yellow #ffff00 rgb(256,256,0)
navy #000080 rgb(0,0,128)	blue #0000ff rgb(0,0,256)
purple #800000 rgb(128,0,0)	fuchsia/magenta #ff00ff rgb(256,0,256)
teal #008080 rgb(0,128,128)	aqua/cyan #00ffff rgb(0,256,256)
silver #c0c0c0 rgb(192,192,192)	white #ffffff rgb(256,256,256)

attention : **`aqua = cyan ; fuchsia = magenta ;`** le vert de référence des afficheurs est **`"lime"`** et non **`"green"`** comme il aurait été normal...

2.2. écriture rgb

Une couleur se définit par l'intensité de chacune de ses trois composantes positives (teinte sur écran, obtenue en juxtapsant les lumiphores, par opposition à la définition en peinture ou en imprimerie où les teintes sont obtenues par superposition des couches colorées) : rouge (red), verte (green/lime) et bleue (blue). Cette intensité se mesure sur une échelle de 0 à 255, et peut donc être codée sur un octet. Ceci s'applique aux feuilles de style : rgb (125, 32, 64).

Netscape avait créé une palette de 216 teintes qu'il avait appelé les teintes "sûres". Bien que ce point de vue soit dépassé, il est intéressant de connaître le tableau des teintes sûres ; l'échelon pour chaque nuance est de 51, ce qui représente 6 échelons de rouge, 6 de vert et 6 de bleu. La table des couleurs Netscape permet de se repérer dans l'échelle des teintes.

Voici ce tableau : **de pavé en pavé, la composante rouge augmente de 51 ; de ligne en ligne, c'est la composante verte et de case en case, c'est la composante bleue**. Dans chaque case on a mis les trois composantes (échelle 0 .. 255) dans l'ordre r, g, b.

0, 0, 0	0, 0, 51	0, 0, 102	0, 0, 153	0, 0, 204	0, 0, 255
0, 51, 0	0, 51, 51	0, 51, 102	0, 51, 153	0, 51, 204	0, 51, 255
0, 102, 0	0, 102, 51	0, 102, 102	0, 102, 153	0, 102, 204	0, 102, 255
0, 153, 0	0, 153, 51	0, 153, 102	0, 153, 153	0, 153, 204	0, 153, 255
0, 204, 0	0, 204, 51	0, 204, 102	0, 204, 153	0, 204, 204	0, 204, 255
0, 255, 0	0, 255, 51	0, 255, 102	0, 255, 153	0, 255, 204	0, 255, 255

51, 0, 0	51, 0, 51	51, 0, 102	51, 0, 153	51, 0, 204	51, 0, 255
51, 51, 0	51, 51, 51	51, 51, 102	51, 51, 153	51, 51, 204	51, 51, 255
51, 102, 0	51, 102, 51	51, 102, 102	51, 102, 153	51, 102, 204	51, 102, 255
51, 153, 0	51, 153, 51	51, 153, 102	51, 153, 153	51, 153, 204	51, 153, 255
51, 204, 0	51, 204, 51	51, 204, 102	51, 204, 153	51, 204, 204	51, 204, 255
51, 255, 0	51, 255, 51	51, 255, 102	51, 255, 153	51, 255, 204	51, 255, 255

102, 0, 0	102, 0, 51	102, 0, 102	102, 0, 153	102, 0, 204	102, 0, 255
102, 51, 0	102, 51, 51	102, 51, 102	102, 51, 153	102, 51, 204	102, 51, 255
102, 102, 0	102, 102, 51	102, 102, 102	102, 102, 153	102, 102, 204	102, 102, 255
102, 153, 0	102, 153, 51	102, 153, 102	102, 153, 153	102, 153, 204	102, 153, 255
102, 204, 0	102, 204, 51	102, 204, 102	102, 204, 153	102, 204, 204	102, 204, 255
102, 255, 0	102, 255, 51	102, 255, 102	102, 255, 153	102, 255, 204	102, 255, 255

153, 0, 0	153, 0, 51	153, 0, 102	153, 0, 153	153, 0, 204	153, 0, 255
153, 51, 0	153, 51, 51	153, 51, 102	153, 51, 153	153, 51, 204	153, 51, 255
153, 102, 0	153, 102, 51	153, 102, 102	153, 102, 153	153, 102, 204	153, 102, 255
153, 153, 0	153, 153, 51	153, 153, 102	153, 153, 153	153, 153, 204	153, 153, 255
153, 204, 0	153, 204, 51	153, 204, 102	153, 204, 153	153, 204, 204	153, 204, 255
153, 255, 0	153, 255, 51	153, 255, 102	153, 255, 153	153, 255, 204	153, 255, 255

204, 0, 0	204, 0, 51	204, 0, 102	204, 0, 153	204, 0, 204	204, 0, 255
204, 51, 0	204, 51, 51	204, 51, 102	204, 51, 153	204, 51, 204	204, 51, 255
204, 102, 0	204, 102, 51	204, 102, 102	204, 102, 153	204, 102, 204	204, 102, 255
204, 153, 0	204, 153, 51	204, 153, 102	204, 153, 153	204, 153, 204	204, 153, 255
204, 204, 0	204, 204, 51	204, 204, 102	204, 204, 153	204, 204, 204	204, 204, 255
204, 255, 0	204, 255, 51	204, 255, 102	204, 255, 153	204, 255, 204	204, 255, 255

255, 0, 0	255, 0, 51	255, 0, 102	255, 0, 153	255, 0, 204	255, 0, 255
255, 51, 0	255, 51, 51	255, 51, 102	255, 51, 153	255, 51, 204	255, 51, 255
255, 102, 0	255, 102, 51	255, 102, 102	255, 102, 153	255, 102, 204	255, 102, 255
255, 153, 0	255, 153, 51	255, 153, 102	255, 153, 153	255, 153, 204	255, 153, 255
255, 204, 0	255, 204, 51	255, 204, 102	255, 204, 153	255, 204, 204	255, 204, 255
255, 255, 0	255, 255, 51	255, 255, 102	255, 255, 153	255, 255, 204	255, 255, 255

2.3. écriture hexadécimale pour les feuilles de style.

L'écriture hexadécimale est une écriture "condensée" de l'écriture rgb() : on remplace l'entête par # et chaque composante est écrite en hexadécimal, sans séparation. Voici le tableau de correspondance entre hexadécimal et décimal : en maigre, le décimal, en gras l'hexadécimal (toujours deux chiffres).

0	**00**	32	**20**	64	**40**	96	**60**	128	**80**	160	**a0**	192	**c0**	224	**e0**
1	**01**	33	**21**	65	**41**	97	**61**	129	**81**	161	**a1**	192	**c1**	225	**e1**
2	**02**	34	**22**	66	**42**	98	**62**	130	**82**	162	**a2**	194	**c2**	226	**e2**
3	**03**	35	**23**	67	**43**	99	**63**	131	**83**	163	**a3**	195	**c3**	227	**e3**
4	**04**	35	**24**	68	**44**	100	**64**	132	**84**	164	**a4**	196	**c4**	228	**e4**
5	**05**	37	**25**	69	**45**	101	**65**	133	**85**	165	**a5**	197	**c5**	229	**e5**
6	**06**	38	**26**	70	**46**	101	**66**	134	**86**	166	**a6**	198	**c6**	230	**e6**
7	**07**	39	**27**	71	**47**	103	**67**	135	**87**	167	**a7**	199	**c7**	231	**e7**
8	**08**	40	**28**	72	**48**	104	**68**	136	**88**	168	**a8**	200	**c8**	232	**e8**
9	**09**	41	**29**	73	**49**	105	**69**	137	**89**	169	**a9**	201	**c9**	233	**e9**
10	**0a**	42	**2a**	74	**4a**	106	**6a**	138	**8a**	170	**aa**	202	**ca**	234	**ea**
11	**0b**	43	**2b**	75	**4b**	107	**6b**	139	**8b**	171	**ab**	203	**cb**	235	**eb**
12	**0c**	44	**2c**	76	**4c**	108	**6c**	140	**8c**	172	**ac**	204	**cc**	236	**ec**
13	**0d**	45	**2d**	77	**4d**	109	**6d**	141	**8d**	173	**ad**	205	**cd**	237	**ed**
14	**0e**	46	**2e**	78	**4e**	110	**6e**	142	**8e**	174	**ae**	206	**ce**	238	**ee**
15	**0f**	47	**2f**	79	**4f**	111	**6f**	143	**8f**	175	**af**	207	**cf**	239	**ef**

16	**10**	48	**33**	80	**50**	112	**70**	144	**90**	176	**b0**	208	**d0**	240	**f0**
17	**11**	49	**31**	81	**51**	113	**71**	145	**91**	177	**b1**	209	**d1**	241	**f1**
18	**12**	50	**32**	82	**52**	114	**72**	146	**92**	178	**b2**	210	**d2**	242	**f2**
19	**13**	51	**33**	83	**53**	115	**73**	147	**93**	179	**b3**	211	**d3**	243	**f3**
20	**14**	52	**34**	84	**54**	116	**74**	148	**94**	180	**b4**	212	**d4**	244	**f4**
21	**15**	53	**35**	85	**55**	117	**75**	149	**95**	181	**b5**	213	**d5**	245	**f5**
22	**16**	54	**36**	86	**56**	118	**76**	150	**96**	182	**b6**	214	**d6**	246	**f6**
23	**17**	55	**37**	87	**57**	119	**77**	151	**97**	183	**b7**	215	**d7**	247	**f7**
24	**18**	56	**38**	88	**58**	120	**78**	152	**98**	184	**b8**	216	**d8**	248	**f8**
25	**19**	57	**39**	89	**59**	121	**79**	153	**99**	185	**b9**	217	**d9**	249	**f9**
26	**1a**	58	**3a**	90	**5a**	122	**7a**	154	**9a**	186	**ba**	218	**da**	250	**fa**
27	**1b**	59	**3b**	91	**5b**	123	**7b**	155	**9b**	187	**bb**	219	**db**	251	**fb**
28	**1c**	60	**3c**	92	**5c**	124	**7c**	156	**9c**	188	**bc**	220	**dc**	252	**fc**
29	**1d**	61	**3d**	93	**5d**	125	**7d**	157	**9d**	189	**bd**	221	**dd**	253	**fd**
30	**1e**	62	**3e**	94	**5e**	126	**7e**	158	**9e**	192	**be**	222	**de**	254	**fe**
31	**1f**	63	**3f**	95	**5f**	127	**7f**	159	**9f**	191	**bf**	223	**df**	255	**ff**

exemples **`rgb(124,156,35)`** se traduit en **`#7c9c23`** (la casse est indifférente).
`fuschia = rgb (255,0,255) = #ff00ff`

Il existe une notation hexadécimale abrégée (3 chiffres hex) qui est déconseillée.

2.4. le facteur alpha.

Il ne faut pas confondre le facteur alpha et la transparence. Sur une image (GIF ou PNG-palette), on peut rendre une teinte transparente ; cela peut être intéressant pour une image "détourée", où seul la partie non détourée du motif est visible. Le facteur alpha est un indice de transparence qui s'applique à chaque pixel : avec un facteur alpha de 0, l'image est transparente (en fait elle n'existe plus), à 1, elle est opaque ; c'est le jeu sur la variation du facteur alpha qui permet de réaliser des effets de fondu avec de scripts adaptés.

Malheureusement, si tous les navigateurs autorisent la transparence grâce aux feuilles de style, la méthode pour y arriver n'est pas la même selon qu'on est du côté de Firefox ou du côté de I.E. On renvoie aux articles spécialisés pour traiter ce problème.

15 exemples pour inciter à aller plus loin

tutoriel HTML

fiche 15

1. un problème de scrolling

1.1. introduction

La présente fiche traite du problème de l'image de fond et du filigramme : on dispose d'un ensemble (page, séquence de texte…) qui est affiché sur un fond (background) fait d'images (répétées ou non). La séquence est suffisamment longue pour nécessiter des ascenseurs pour la visionner. Deux éventualités se présentent : soit on veut que le texte et le fond se déplacent conjointement (filigramme) ; soit on veut que le texte se déplace sur un fond qui reste fixe.

Le w3c a donné des recommandations pour prendre en charge ces exigences, et I.E. n'a pas respecté la norme : d'où les multiples forums où les internautes s'étonnent que le défilement correspond à leur attente sous I.E. mais ne fonctionne pas avec Firefox et autres navigateurs. En général, c'est plutôt le contraire qui se produit. Malheureusement, l'ignorance de la norme conduit à des propositions inefficaces, voire complètement illogiques (jeter un coup d'oeil sur les forums !).

C'est que la norme w3c s'applique à la **page HTML**, et que I.E. l'a implémentée dans des domaines où elle n'avait rien à voir (traitement de l'**`overflow`** des conteneurs par exemple) et où elle crée des incompatibilités avec les autres navigateurs. Dans la norme w3c il faut avoir conscience que le **`background`** ne remplit pas les mêmes exigences selon qu'il est relatif à la page ou à un conteneur par exemple l'arrière-plan déclaré dans la balise body peut dépasser la limites du conteneur de corps de page ou de son contenu, et s'appliquer aux marges alors que ce n'est pas le cas pour les autres conteneurs.

Pour clarifier la question, on va traiter deux exemples : l'un relatif à la page HTML, l'autre à l'**`overflow`** des conteneurs.

1.2. un fond fixe et un texte défilant.

1.2.1. cahier des charges.

Pour cet exemple, on va prendre une image de **`background`** non répétée, en haut et à gauche de la **page** et un texte suffisamment long pour être défilant. La contrainte est que l'image reste fixe dans la fenêtre.

1.2.2. le code.

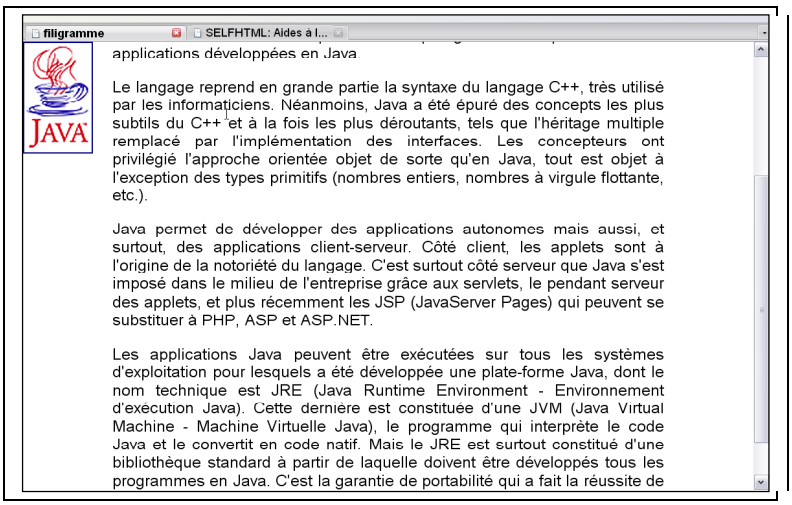
```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
    <title>
      filigramme
    </title>
    <style type="text/css">
#arriereplan { /* s'applique à la balise body */
  background-image: url(javalogo.gif);
  background-repeat: no-repeat;
  background-attachment: fixed;
```

```
}
.leTexte {
  font-family: Arial, sans;
  font-size: 25px;
  margin-left: 130px;
  margin-right: 130px;
  text-align: justify;
    </style>
  </head>
  <body id="arriereplan" >
    <p class="leTexte"> Java est à la fois un langage de programmation
informatique orienté objet et un environnement d'exécution informatique portable
créé par James Gosling et Patrick Naughton employés de Sun Microsystems avec le
soutien de Bill Joy (cofondateur de Sun Microsystems en 1982), présenté
officiellement le 23 mai 1995 au SunWorld.
    </p>
    <p class="leTexte"> Le langage Java a la particularité principale que les
logiciels écrits avec ce dernier sont très facilement portables sur plusieurs
. . . . . .
      partie de texte coupé
. . . . . .
programmes en Java. C'est la garantie de portabilité qui a fait la réussite de
Java dans les architectures client-serveur en facilitant la migration entre
serveurs, très difficile pour les gros systèmes.
    </p>
  </body>
</html>
```

```
* background-attachment: fixed;
```

C'est la ligne intéressante : le **`background-attachment`** est défini pour **la page**.

1.2.3. aspect de la page affichée.



L'image de fond (le logo java) reste fixe lors du défilement de la page.

le fichier HTML : **`html15tp0.html`**

2. Le problème avec les conteneurs.

2.1. la tentation.

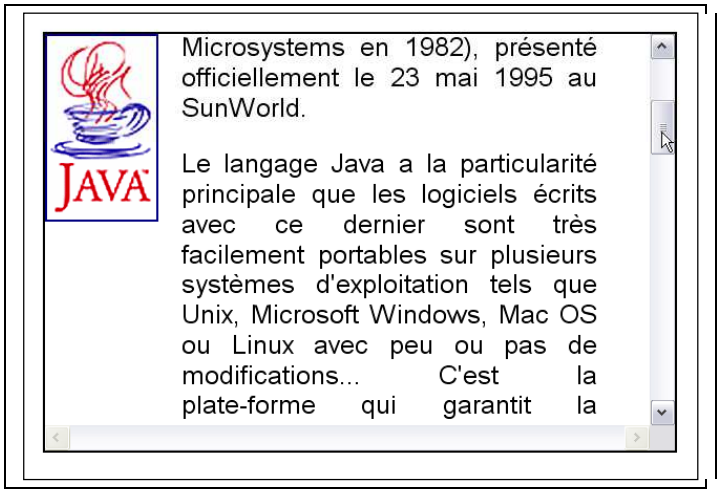
Si on veut réaliser la même chose avec un conteneur, en utilisant le scroll d'**`overflow`**, on est tenté d'utiliser la même méthode : pour l'élément affiché, on définit un fond attaché et un texte trop grand qui nécessite d'utiliser le scroll d'**`overflow`**. Malheureusement, la norme du w3c ne prend pas en charge cette situation, et seul I.E. ne résiste pas à la tentation de la contredire, créant une confusion qu'on ne s'explique pas lorsque l'on change de navigateur (cf : voir les forums sur ce sujet). C'est que, selon la norme, **le traitement de l'overflow à l'aide du scroll s'applique à l'élément qui a provoqué l'overflow, soit ici le texte**. Il faut s'y prendre autrement, et la notion d'attachement est alors tout à fait inutile.

On propose de réaliser l'affichage précédent, mais dans un conteneur de 600x400 px.

2.2. Le code. `html15tp1.html`

```
. . . . . .
<style type="text/css">
#arriereplan {
  background-image: url(javalogo.gif);
  border: 2px solid black;
  background-repeat: no-repeat;
  width:600px;
  height:400px;
  margin:100px;
  overflow:scroll;
}
.leTexte {
  font-family: Arial, sans;
  font-size: 25px;
  margin-left: 130px;
  margin-right: 50px;
  text-align: justify;
    </style>
  </head>
  <body>
    <div id="arriereplan">
    <p class="leTexte"> Java est à la fois un langage de programmation
informatique orienté objet et un environnement d'exécution informatique . . . .
. . . . . .
Java dans les architectures client-serveur en facilitant la migration entre
serveurs, très difficile pour les gros systèmes.
    </p>
  </div>
  </body>
</html>
```

* **le scroll ne s'appliquant qu'au contenu du conteneur. I**l n'y a pas besoin de **fixer** le fond qui l'est naturellement.



3. bouger le fond.

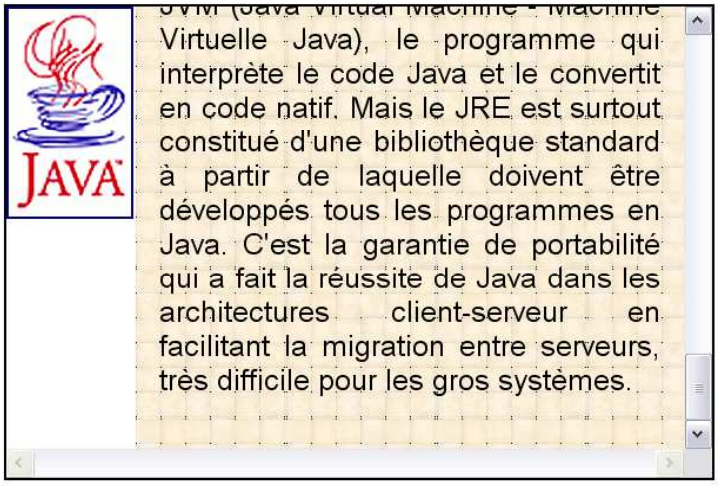
3.1. Et si…

Et si dans le cas précédent, on voulait qu'il y ait une image de fond se déplaçant avec le texte ? La réponse est assez logique : comme le texte est dans des conteneurs (balise **p**), il n'y a qu'à donner un fond à ces conteneurs : il se déplacera avec le texte, puisque soumis au scroll ! Il y a cependant un petit inconvénient : seule la partie concernée par le fond du conteneur **p** comporte l'image de fond. Pas les marges naturelles du conteneur p ! Dans ce cas, un second conteneur spécial pour le fond, le conteneur "**tresse**" peut être utile ; voici les modifications intéressantes du texte source.

3.2. Le code. `html15tp2.html`

```
<style type="text/css">
#arriereplan {
  background-image: url(javalogo.gif);
  border: 2px solid black;
  background-repeat: no-repeat;
  width:600px;
  height:400px;
  margin:100px;
  overflow:scroll;
}
#tresse {
  background-image: url(tresse.jpg);
  margin-left: 110px;
  padding-top:20px;
  padding-bottom:20px;
  }
.leTexte {
  font-family: Arial, sans;
  font-size: 25px;
  margin-left:20px;
  margin-right: 20px;
  text-align: justify;
}
    </style>
  </head>
```

```
  <body>
    <div id="arriereplan">
    <div id="tresse">
    <p class="leTexte"> Java est à la fois un langage de programmation
informatique orienté objet et un environnement d'exécution informatique portable
créé par James Gosling et Patrick Naughton employés de Sun Microsystems avec le
. . . . . .
```

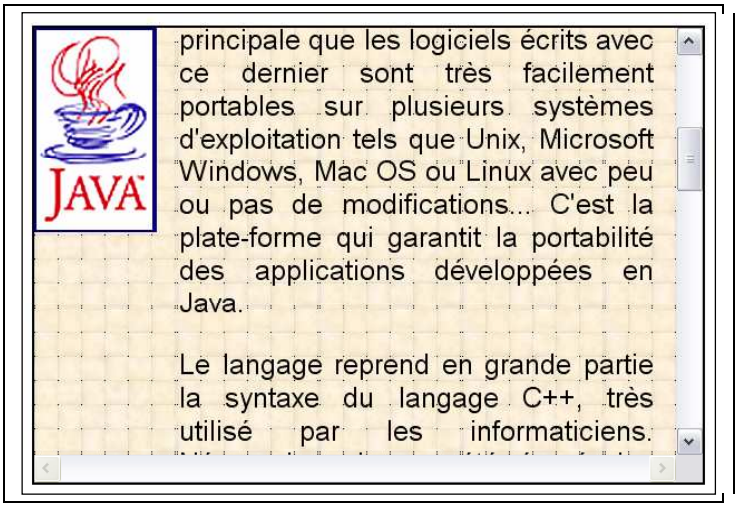


exercice

reprendre l'exercice précédent, avec les exigences suivantes :

* le logo reste fixe ;

* le fond de tresse est mobile, mais occupe tout l'espace du conteneur global.



indications:

Utiliser les **`z-index`** sur des conteneurs en position absolue.

solution : **`html15tp3.html`**

4. Mappage d'une image ou d'un conteneur.

Le HTML ancien possède dans son arsenal de balises "**les images réactives**", complément aux liens ancrés. A une image est superposé un **mappage**, ensemble d'aires rectangulaires, polygonales ou

circulaires, A chaque figure est associée une adresse (une url) ; quand on clique la figure, on réalise un travail identique à celui d'un lien ancré : appel d'une page ou déplacement dans la page, voire action **`javascript:`** ou **`mailto:`** Ce procédé, fort utilisé dans les cartes de Noël des années 2000 semble actuellement tombé en désuétude.

On peut cependant avoir besoin d'un procédé de ce type, et la bonne méthode est l'utilisation d'un script. Tout serait pour le mieux si, une fois de plus, I.E. ne se distinguait des autres navigateurs dans sa gestion des événements. Il y a donc deux modes de gestions des événements : le mode Netscape et le mode I.E. Heureusement, le code qui gère les événements est court !

4.1. cahier des charges du problème.

On dispose d'une image. Lorsque le curseur est sur l'image, il a la forme d'une croix et en dehors, c'est le curseur par défaut. Lorsque l'on clique, une fenêtre **`alert()`** indique les coordonnées sur l'image du centre de la croix lors du clic.

4.2. Méthodes.

* Lors d'un clic souris sur un élément graphique, le navigateur produit un événement, qui est **un objet javascript ; le clic** peut être programmé pour appeler une fonction définie par l'utilisateur. Malheureusement, si la fonction utilisateur est appelée par les deux navigateurs, la récupération de l'objet événement est différente : pour Netscape, cet objet est passé en paramètre de la fonction utilisateur, alors que pour I.E. L'événement est une propriété de l'objet global **`window`** (racine du modèle javascript de la page) ; aucun paramètre n'est passé à la fonction, ce qui fait que si on spécifie un paramètre, celui ci reste indéfini (casté en **`false`** au besoin).

* Les champs non plus ne sont pas les mêmes : pour I.E. les champs **`offsetX`** et **`offsetY`** désignent le coordonnées du curseur au moment du clic dans un repère lié à l'élément graphique pour lequel le gestionnaire d'événement (**`onclick)`** a été défini. C'est simple !

Pour Netscape, les coordonnées **`layerX`** et **`layerY`** sont données par rapport au **`layer (`**calque**`)`** qui porte l'élément graphique pour lequel le gestionnaire d'événement (**`onclick)`** a été défini. C'est un peu plus compliqué, d'autant plus que les éléments **`layer`** ont été abandonnés ; mais on dispose d'un remplaçant très efficace : **pour les navigateurs de la famille Netscape (Firefox...), un élément layer est un élément sorti du flot HTML avec un positionnement `absolute`**. La page d'affichage graphique est l'élément **`absolute`** par défaut. Si on ne prend aucune précaution, on a donc pour coordonnées du point de clic la position de ce point dans la page. On transforme donc par feuille de style la position par défaut de l'image (en principe, dans le flot HTML) en position **`absolute`**. Mais on sait que celle-ci doit être définie par rapport à un conteneur du flot HTML déclaré en position **`relative`**. **C'est la règle**. Ce conteneur est défini par une balise de conteneur ; le plus simple est de prendre la balise **`div`**, qui n'a qu'un attribut explicite par défaut qui nous intéresse : c'est un **`block`**, cela nous convient.

* reste la problème de l'affichage de l'image. Pas de problème avec la famille Netscape, la balise img est parfaite. Malheureusement I.E. affiche une barre de menu *popup* sur l'image, ce qui est particulièrement gênant. On est alors contraint d'utiliser pour éviter cet artifice, une autre méthode, celle qui est utilisée pour les formulaires, qui, quand n'est pas insérée dans la balise **`form,`** a le même comportement que **`img`** (à la différence près qu'il n'y a pas de barre popup sous I.E.).

4.3. Le code. `html15tp4.html`

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
          "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
  <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">

  <title></title>
  <script type="text/javascript">
```

```
 function click_XY (evenement) {
 if (evenement) { /* solution FireFox */
   x1 = evenement.layerX;
   y1 = evenement.layerY;
 }
 else { /* solution IE */
   x1= window.event.offsetX;
   y1= window.event.offsetY;
}
alert ("x = "+x1+"   y = "+y1);
 }

 function curseurover() {
   document.getElementById("promo").style.cursor="crosshair";
 }

 function curseurout() {
   document.getElementById("promo").style.cursor="default";
 }

 function init() {
   document.getElementById("promo").onclick = click_XY;
   document.getElementById("promo").onmouseover = curseurover;
   document.getElementById("promo").onmouseout = curseurout;
 }
 window.onload = init;
 </script>
 </head>
 <body>
 <h2>essais événements et coordonnées</h2>
 <div style="position:relative;height:640px;width:1000px;">
   <input type="image" id="promo" src="_vero.png"
        style="position:absolute;left:10px;cursor:crosshair;" />
 </div>
 <h2>fin de l'image</h2>
 </body>
</html>
```

* faire attention au DOCTYPE (éviter les **dtd** archaïques).

* **window.onload = init;** c'est l'équivalent plus moderne de **<body onLoad="init()">**

* on a disposé les styles dans les balises pour la démonstration ; en production, il faudrait faire un fichier CSS et un fichier JS.

* **if (evenement) {** Si c'est un navigateur Netscape, **evenement** est passé en paramètre et il est casté en **true**. Si c'est I.E. il n'y a pas de paramètre, ce que javascript accepte très bien (la signature n'a pas à être respectée en javascript). **evenement** est donc indéfini (**indefined**) et il est casté en **false.**

* **evenement.layerX**; on rappelle que layerX est l'abscisse sur le layer (ici, l'image, en position **absolute**).

* **window.event.offsetX**; l'événement **event** se rapporte à l'élément où est défini le gestionnaire d'événement, soit ici l'image. On obtient bien la même valeur sur les deux navigateurs.

* **function init() {** ... On ne peut définir des attributs relatifs à un élément que si cet élément existe déjà ; pour cela il faut que le corps du HTML soit chargé (la fonction **window.onload** se déclenche alors.

* **height:640px;width:1000px;** comme on définit l'image en position **absolute**, il lui faut un conteneur en position relative. Il faut de plus que le conteneur soit assez grand pour contenir toute l'image et rejeter le reste du flot HTML après lui ; sinon, ce qui reste du flot HTML normal passe derrière l'image (ici les sous-titre « fin de l'image »).

* **cursor:crosshair;** A première vue, cela ne sert à rien ; sauf qu'il faut prévoir le cas où la fonction **curseurover()** n'est pas appelé. Avec I.E. il peut se produire que le curseur soit au milieu de l'image lors du chargement ; le curseur doit alors être le bon !

4.4. une autre illustration de la même méthode.

Le fichier **html15tp6.html** reprend le cas étudié dans la fiche 9, mais réalise une simulation stylée du **title** sur l'image de l'entête.

On trouvera ci-dessous les variantes :

4.4.1. dans le fichier HTML :

```
<h1 id="header">
      <p id="titleSimul">cliquer pour aller sur accueil.html</p>
      <a id="ancre" href="accueila.html"></a>
    </h1>
```

4.4.2. dans les styles :

```
h1#header {
    height:170px;
    background-image:url(poirequeue.png);
    background-repeat:no-repeat;
    background-position:left top;
    background-color: rgb(52, 35, 25);
    margin:0;
    position:relative;
  }

  h1#header a {
    height:90px;
    width: 550px;
    background-image: url(banniere.png);
    background-repeat:no-repeat;
    background-position:left top;
    display:block; /* pour position et dimensions */
    position:relative; /* pour positionner */
    left:190px;
    top:15px;
  }
  h1#header p#titleSimul {
    border:2px solid red;
    background-color:yellow;
    font-family: cursive;
```

```
    font-size:16px;
    font-weight:normal;
    color:blue;
    padding:5px;
    position:absolute;
    z-index:1;
  }
```

4.4.3. le code javascript :

```
<script type="text/javascript">
  var objetAncre;
  var objetSimul;
  var objetStyle;

    function init () {
      objetAncre = document.getElementById("ancre");
      objetSimul = document.getElementById("titleSimul")
      objetStyle = document.getElementById("titleSimul").style;
      objetAncre.onmousemove = titleMove;
      objetAncre.onmouseout = titleOut;
      objetStyle.display ="none";
    }

function titleMove(evenement) {
      var x1, y1;
      if (evenement) { /* w3c*/
        x1 = evenement.layerX+205;
        y1 = evenement.layerY-25;
      }
      else { /* IE */
        x1= window.event.offsetX+205;
        y1= window.event.offsetY-10;
      }
      objetStyle.top=y1+"px";
      objetStyle.left=x1+"px";
      objetStyle.display = "block";
     }

    function titleOut() {
      objetStyle.display="none";
    }

    window.onload = init;
  </script>
```

* Le « **`title` »** est une balise **p** qui s'affiche lorsque le curseur se déplace sur le conteneur du lien ancré, suit le curseur dans ses déplacements, et s'efface (**`display:none`**) lorsque l'on sort du lien ancré.

Note : on doit prendre garde à ce que le curseur soit en dehors du **`title`**, car sinon, il s'applique à la balise **p** et pas au lien ancré **a** : la forme n'est pas la même, les attributs relatifs aux événements

(comme onMouseOver, onMouseOut...) sont également affectés.

5. feuilles de style à la demande

5.1. introduction.

Le chargement d'une feuille de style peut se faire par une balise link :
`<link rel="stylesheet" type="text/css" href="mafeuillerouge.css" />`

Le chargement de feuille de style d'une page HTML peut être dynamique, c'est à dire non programmé de manière fixe dans la page HTML. On l'a rencontré dans la fiche sur la *popup* **`confirm()`**.. On exclut la solution où le **serveur** a le moyen de définir le chargement de la feuille de style, par exemple à travers un script php ; on se situe délibérément du côté client.

On propose la programmation du thème suivant : le fichier de feuille de style à charger peut-il être défini dans l'url ? On avait vu une autre façon de faire, utilisant un script intégré dans le flot HTML.

L'url aurait alors l'allure suivante :

`file:///d:/mon_site/html15tp5.html?css=mafeuillerouge.css`

avec **`mafeuillerouge.css`** comme nom de fichier de feuille de style (supposée dans le même répertoire pour éviter les problème d'url).

5.2. Principe.

Le nom de la feuille de style est passée en paramètre dans l'url : il suffit de le récupérer à parti du **`href`** de la page. L'url de la page est connu dès le début de l'appel de page : on peut donc le récupérer et l'analyser dans un script de la partie **`head`** du fichier HTML.

Une fois connu le nom du fichier de style css, on écrit la balise de lien qui lui correspond dans la page en cours d'exécution ; la balise écrite devient la ligne suivante du fichier HTML en train d'être analysé. Elle est immédiatement exécutée. Le corps de la page se conforme ainsi à la feuille de style chargée.

Avec la système de bouton, on recharge la page, mais en changeant le paramètre d'url.

5.3. Le code. `html15tp5.html`

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
          "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
  <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
  <title>feuille de style variable</title>
  <script type="text/javascript">
   function rouge () {
     location.href="html15tp5.html?css=mafeuillerouge.css";
   }
   function vert () {
     location.href="html15tp5.html?css=mafeuilleverte.css";
   }
   function bleu () {
     location.href="html15tp5.html?css=mafeuillebleue.css";
   }

  c=location.href
  d=c.search("css=");
  if (d!=-1)
  var fichier=c.substring(d+4,c.length);
  document.write('<link rel="stylesheet" type="text/css" href="'+
```

```
  fichier+'" />');

  </script>
  </head>
  <body>
     <h1>essai de css variable</h1><br />
    <input type="button" value=" rouge " onClick="rouge();" />  
    <input type="button" value=" vert " onClick="vert();" />  
    <input  type="button" value=" bleu " onClick="bleu();" />
  </body>
</html>
```

*** d=c.search("css=");**

La méthode **search()** retourne –1 en cas d'échec ; le décompte se fit à partir de 0 en cas de succès.

Si d vaut 47, c'est que la chaîne "css=" commence au caractère 48 du le comptage des caractères.

*** c.substring(d+4,c.length);**

- le premier paramètre est le numéro du premier caractère retenu dans la chaîne.

- le second paramètre constitue une marque d'arrêt ; il n'est donc pas nécessaire qu'il y ait un caractère à cet indice ! Ce qui est heureux, car **c.length** donne la longueur de la chaîne et le dernier caractère a pour indice **c.length-1**.

Note. on s'est contenté de redéfinir la balise h1, ce qui est suffisant pour la démo !

fichier de code : **html15tp5.html**