



OFPPT

ROYAUME DU MAROC

مكتب التكوين المهني وإنعاش الشغل
Office de la Formation Professionnelle et de la Promotion du
Travail

RESUME THEORIQUE & GUIDE DES TRAVAUX PRATIQUES

MODULE N° : 17

TITRE DU MODULE : Création de sites web

SECTEUR : TERTIAIRE

SPECIALITE : TSDI

Niveau : TECHNICIEN Spécialisé

Juin 2006

REMERCIEMENT

La DRIF remercie les personnes qui ont contribué à l'élaboration du présent document.

Pour la supervision :

MME.BENNANI WAFAE
M. ESSABKI NOURDDINE

DIRECTRICE CDC TERTIAIRE & TIC
CHEF DE DIVISION CCFE

Pour la conception :

- ABDELILAH JELLAL

Formateur animateur au CDC Tertiaire & TIC /DRIF

Pour la validation :

Les utilisateurs de ce document sont invités à communiquer à la DRIF toutes les remarques et suggestions afin de les prendre en considération pour l'enrichissement et l'amélioration de ce programme.

**Said Slaoui
DRIF**

MODULE 17 : Création de sites web

OBJECTIF OPERATIONNELS DE PREMIER NIVEAU DE COMPORTEMENT

COMPORTEMENT ATTENDU

Pour démontrer sa compétence, le stagiaire doit **créer un site web** selon les conditions, les critères et les précisions qui suivent :

CONDITIONS D'EVALUATION

- Travail individuel effectué avec un ordinateur équipé d'un outil de développement de site web (type web expert)

CRITERES GENERAUX DE PERFORMANCE

- Utilisation des commandes appropriées.
- Respect du temps alloué.
- Respect des règles d'utilisation du matériel et logiciel Informatique.

**OBJECTIF OPERATIONNELS DE PREMIER NIVEAU
DE COMPORTEMENT**

**PRECISIONS SUR LE COMPORTEMENT
ATTENDU**

**CRITERES PARTICULIERS DE
PERFORMANCE**

A. Créer des pages web en HTML

- Structuration judicieuse des données sous forme de paragraphes, tableaux, listes...
- Utilisation correcte des balises de base et d'entête
- Justesse de définition de liens dans une page
- Définition correcte des cadres externes et Internes dans un document web ;

B. Créer une feuille de style en CSS

- Définition et création des styles dans le respect de la charte graphique
- Justesse de l'enregistrement de la feuille de style
- Appel correct de la feuille de style par la page HTML

C. Intégrer des données multimédias dans une page web

- Justesse de numérisation des données multimédias
- Optimisation correcte de la taille et du temps de téléchargement des objets multimédias
- Intégration correcte dans la page HTML

D. Créer des formulaires de saisie

- Choix judicieux et insertion correcte des contrôles de formulaire dans la page web
- Paramétrage correct des formulaires de saisie pour déclencher des traitements sur les serveurs

E. Programmer en JavaScript

- Choix approprié du type de l'interaction à ajouter à la page web
- Programmation correcte des fonctions de contrôle de formulaires
- Programmation adéquate des traitements associés aux événements (Handlers) avec JavaScript
- Modification dynamique des styles au travers du DOM et des CSS
- Stocker et Mettre à jour des informations persistantes côté client (cookies)

F. Intégrer des composants côté client

- Intégration et configuration correctes d'Applets et d'Activex
- Programmation correcte des interactions avec les composants

G. Créer et manipuler des documents XML

- Structurer correctement un document XML
- Modifier dynamiquement les contenus des balises au travers du DOM et JavaScript
- Organiser et codifier correctement la présentation des informations au travers d'XSLT (ou CSS)

OBJECTIFS OPERATIONNELS DE SECOND NIVEAU

LE STAGIAIRE DOIT MAITRISER LES SAVOIR, SAVOIR-FAIRE, SAVOIR -PERCEVOIR OU

SAVOIR-ETRE JUGES PREALABLES AUX APPRENTISSAGES DIRECTEMENT REQUIS POUR L'ATTEINTE DE L'OBJECTIF DE PREMIER NIVEAU, TELS QUE :

Avant d'apprendre à Créer des pages web en HTML (A) :

1. Connaître la plate forme du travail
2. Connaître Internet
3. Expliquer le rôle d'un site WEB et son apport par rapport à d'autre supports d'information
4. Installer l'outil de développement à étudier
5. Expliquer l'utilité d'avoir des balises d'entête dans un document WEB

Avant d'apprendre à Créer une feuille de style en CSS (B)

6. Expliquer la structure d'un document html ;
7. Expliquer l'utilité de la mise en forme d'un document html ;
8. Expliquer l'utilité à utiliser les feuilles de style

Avant d'apprendre à coder un document XML

9. Présentation du modèle Objet de document (DOM)

RESUME THEORIQUE ET TRAVAUX PRATIQUES

Cours & Exemples pratiques

I - HTML

Introduction

Le langage Html est un langage qui permet de créer une page web . il est l'abréviation de **Hyper Text Markup Language** .

Le Html n'est pas un langage de programmation comme le c / c++ , vb ...etc , mais , c'est un simple texte qui contient des balises ou tag en anglais. C'est donc le Html n'est pas besoin d'un compilateur pour s'exécuter , mais seulement d'un navigateur (Browser en anglais) qui interprète les balises Html , comme l'Internet Explorer ou Netscape Navigator .

Comment commencer

Pour écrire les balises Html , on a besoin seulement d'un simple éditeur de texte comme le WordPad ou le Bloc-Note .

Pour être interprété par le navigateur , la page html doit porter l'extension .html ou htm .

Ex : test.html

Pour ceux qui ne veulent saisir du code html , il existe plusieurs éditeur de html qui permettent de générer les balise par assistant .

On peut citer :

- WebExpert
- FrontPage
- Dreamweaver
- WebCreator

Leçon 1 : Les Bases

Une page web contient les balises html. Ces dernières sont composées en général, par un marqueur d'ouverture et un marqueur de fermeture .

Les marqueurs Html utilisent les caractères < et > pour limiter le début et la fin . Le marqueur de fin s'écrit avec un "/".

Exemple : <marqueur> texte </marqueur>

I- Structure d'une page Html

Trois balises permettent de créer la structure de base d'une page quelconque .

Ce sont <html>, <head> et <body> :

- La première est celle qui permet de définir le langage utilisé,

Vous devez toujours placer cette balise <html> au début de votre page puis se termine par </html>.

- Puis y placer d'autres balises à l'intérieur :

Pour l'en-tête du document, partie non-visible, mais qui contient les informations permettant au moteur de recherche de trouver votre site, vous devez placer la balise <head> puis se termine par </head>.

Je reviendrais plus tard sur les informations contenu à l'intérieur de cette balise que je nomme balise "en-tête".

- Pour le corps du document, la partie visible pour l'internaute qui visitera votre site, vous devez placer cette balise <body> puis se termine par </body>.

C'est dans cette dernière balise que se trouveront tous les éléments que vous montrerez à vos visiteurs !

Revenons à la balise "en-tête" :

Cette balise contient une autre balise qui est le titre de votre page, ce titre apparaîtra sur la barre en haut de votre navigateur et permet aussi aux moteurs de recherche de vous trouver, donc vous devez y placer un titre explicite !

Cette balise est <title> puis se termine par </title>.

II- Création d'une simple page-web

Voilà, maintenant un exemple :

```
<html>
```

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

```
<!--début de votre page HTML-->
```

```
<head>
```

```
<!--votre en-tête-->
```

```
<title>Titre de votre site</title>
</head>
<!--fermeture de l'en-tête-->
<body>
<!--début du corp de votre page-->!
<!--taper votre texte-->
</body>
<!--fermeture du corp de votre page-->
</html>
<!--fermeture de votre fichier HTML-->
```

<i>Exemple:</i>	<i>Résultat</i>
<pre><html> <head> <title>ceci est un test</title> </head> <body> <i>Bienvenu dans ma première page</i> </body> </html></pre>	Bienvenu dans ma première page

Leçon 2 : Les formes

Comment formater votre page :

Pour cela, vous pouvez ajouter des attributs à la balise <body>

Comment formater un texte en gras :

 définit un texte gras puis se termine par

La règle : Votre Texte en Gras

Exemple : **Votre Texte en Gras**

Comment formater un texte en italique :

<i> définit un texte en Italique puis se termine par </i>

La règle : <i>Votre Texte en Italique</i>

Exemple : *Votre Texte en Italique*

Comment formater un texte souligné :

<u> définit un texte souligné puis se termine par </u>

La règle : <u>Votre Texte Souligné</u>

Exemple : Votre Texte Souligné

Comment formater un texte barré:

<s> définit un texte barré puis se termine par </s>

La règle : <s>Votre Texte barré</s>

Exemple : ~~Votre Texte barré~~

Comment formater un texte en couleur :

 définit un texte en couleur puis se termine par

La règle : Votre Texte en Couleur

Exemple : Votre Texte en Couleur

Comment modifier la taille du texte :

 définit la taille du texte puis se termine par

La règle : Votre Texte en taille 5

Exemple : Votre Texte en taille 5

Comment modifier la police du texte :

 définit la police du texte puis se termine par

La règle : Votre Texte en Arial

Exemple : Votre Texte en Arial

Comment faire un retour à la ligne :

 définit un retour à la ligne

Exemple :

Texte de ma première ligne

Texte de ma seconde ligne

Texte de ma troisième ligne

Ce qui nous donnent :

Texte de ma première ligne

Texte de ma seconde ligne

Texte de ma troisième ligne

Comment formater un paragraphe :

<p> définit un paragraphe puis se termine par </p>

La règle : <p align="right">Votre Paragraphe Aligné à Gauche avec Right</p>

Exemple :

Votre Paragraphe Aligné à Gauche avec Right

Votre Paragraphe Aligné au Centre avec Center

Votre Paragraphe Aligné à Droite avec Left

Tous ces balises de formats peuvent se cumuler :

Exemple :

<i><u><s>Ecrire en gras, en italique, en souligné, rayé et en couleur</s></u></i>

Résultat : Ecrire en gras, en italique, en souligné, barré et en couleur

Comment faire des titres avec l'attribut H :

Il existe six styles prédéfinis pour les titres, allant de H1 à H6

<i>Exemple</i>	<i>Résultat</i>
<pre><h1>Titre H1</h1> <h2>Titre H2</h2> <h3>Titre H3</h3> <h4>Titre H4</h4> <h5>Titre H5</h5> <h6>Titre H6</h6></pre>	<p>Titre H1</p> <p>Titre H2</p> <p>Titre H3</p> <p>Titre H4</p> <p>Titre H5</p> <p>Titre H6</p>

Leçon 3 : Les Listes en Html***Généralité :***

Le Html offre la possibilité de créer des listes dans lesquelles chaque élément se trouve sur une ligne différente.

On distingue trois types de listes :

La liste non ordonnée , (Unordered List): chaque élément de la liste se traduit dans le code par le marqueur , (list input) et à l'affichage par un point .

La liste ordonnée , (ordered List) : les éléments sont précédés d'un numéro d'ordre .

La liste de définition ,<dl>(Definition List) : chaque élément peut-être accompagné d'une définition écrite à la ligne suivante avec un résultat .

Exemple:

<i>Type de liste</i>	<i>Exemple de code</i>	<i>Résultat</i>
Liste non ordonnée	Langage de programmation VB Pascal C/C++ 	Langage de programmation • VB • Pascal • C/C++
Liste ordonnée	Microsoft Office Word Excel Access 	Microsoft Office 1. Word 2. Excel 3. Access
Liste de définition	<dl> définition de termes : <dt>HTML <dd>Hyer Text Markup Language <dt>WWW <dd>World Wide Web </dl>	définition de termes : HTML Hyer Text Markup Language WWW World Wide Web

Remarque & complément

Le paramètre type de la liste permet de choisir la puce affichée pour chaque entrée de la liste .

<i>Type</i>	<i>Résultat</i>
Langage de programmation <ul type = "disc"> VB Pascal	Langage de programmation • VB

 C/C++ 	<ul style="list-style-type: none"> • Pascal • C/C++
Langage de programmation <ul type = "circule"> VB Pascal C/C++ 	Langage de programmation <ul style="list-style-type: none"> ○ VB ○ Pascal ○ C/C++
Langage de programmation <ul type = "disc"> VB Pascal C/C++ 	Langage de programmation <ul style="list-style-type: none"> ▪ VB ▪ Pascal ▪ C/C++

Le paramètre type de la liste permet de choisir le type de numérotation .

<i>Type</i>	<i>Résultat</i>
Développement web <ol type = "1"> Html Javascript Java 	Développement web 1. Html 2. Javascript 3. Java
Développement web <ol type = "A"> Html Javascript Java 	Développement web A. Html B. Javascript C. Java
Développement web <ol type = "a"> Html Javascript Java 	Développement web a Html b Javascript c Java
Développement web <ol type = "I"> Html Javascript Java 	Développement web I. Html II. Javascript III. Java

Leçon 4 : Les images

Les navigateurs Html reconnaissent deux formats d'images : les images gif et jpg.

Comment insérer une image sur votre page :

L'insertion d'image se fait par la balise suivante :

```

```

le "img src " caractérise l'affichage de l'image stockée dans un fichier dont l'adresse est donnée par le chemin entre guillemets .

Exemple : ou

Résultat :



Le paramètre ALIGN de

La balise d'insertion d'image, doit être précisée par l'ajout de paramètres suivantes :

```
<img src= "image.gif" align=TOP|BUTTOM|MIDDLE|LEFT|RIGHT>
```

- Les valeurs TOP|BUTTOM|MIDDLE sont utilisées lorsqu'on veut adapter des images à un court texte (légende de l'image).

Exemple:

Brique



Exemple:

Brique



Exemple:

Brique



- Les valeurs LEFT|RIGHT permettent de border l'image avec un texte explicatif:

Exemple: `` Ceci est un exemple de texte explicatif d'image .

Résultat :

Ceci est un exemple de texte explicatif d'image .



Exemple: `` Ceci est un exemple de texte explicatif d'image .

Résultat:



Ceci est un exemple de texte explicatif d'image .

- Pour mettre le texte en dessous de l'image, il suffit d'utiliser un paramètre spécial `<BR CLEAR = "ALL">`

Exemple: `<BR CLEAR = "ALL">` Ceci est un exemple de texte explicatif d'image .

Résultat:



Ceci est un exemple de texte explicatif d'image .

Réglage des dimensions de l'image

Deux paramètres permettent de modifier ou définir la taille des images en pixel.

- **width** = 100

cet attribut définit la taille en longueur de l'image

Exemple : ``

- **height** = 50

cet attribut définit la taille en hauteur de l'image

Exemple : ``

Autre paramètres de ``

- **hspace** = 10

cet attribut définit l'espacement horizontal avant l'image, c'est à dire que plus ce nombre sera élevé, plus l'image se décalera sur la droite de l'écran;

Exemple : ``

➤ **vspace** = 10

cet attribut définit l'espacement vertical avant l'image, c'est à dire la distance au texte dans le sens vertical, en bas et en haut de l'image.

Exemple : ``

➤ **ALT** = "texte"

Cette balise permet d'afficher du texte à la place de l'image ou avant que celle-ci apparaisse.

Exemple : ``

Compléments : Motif de fond

Pour utiliser une image comme motif de fond d'une page Html, on insère le paramètre Background, dans la balise Body.

Exemple : `<Body Background = "Briques.gif" >`

Leçon 5 : Les liens**Généralités :**

En cliquant sur un texte ou sur une image , on active un lien qui désigne l'adresse d'un document (URL).

L'URL = **U**niform **R**esource **L**ocator du document

Comment insérer un lien sur votre page :

le code est `ici le texte qui apparait à l'écran`

Exemple :

`Ma Page`

Résultat :

Ma Page

les liens sont toujours soulignés, celui-ci vous amène à cette page ce lien sert donc à naviguer dans votre site mais l'on peut aussi faire un lien vers un autre site web :

Exemple :

` Moteur de recherche `

Résultat :

Moteur de recherche

pour ce type de lien, il est préférable de rajouter ceci `target="_new"`, afin d'ouvrir une nouvelle fenêtre ce qui nous donne :

`Moteur de recherche google`

Comment faire un lien image :

Comme nous avons vu précédemment pour le lien "texte", il suffit d'y mettre une image

Exemple :

``

Résultat :

**Comment faire un lien texte-image :**

Exemple :

`
Briques, Cliquer sur l'image`

Résultat :



Comment faire un lien interne sur votre page :

Vous devez placer un marqueur invisible entre les balises <body> et </body> afin de déterminer l'endroit où votre lien vous enverra sur votre page .

**Il vous suffit de placer ce code : **

Puis de placer le lien sur votre page pour renvoyer vos visiteurs sur votre marqueur.

Exemple :

```
<a href="#nom_du_marqueur">Aller sur mon marqueur</a>
```

Résultat :

Cliquer sur le lien ci-dessous pour aller sur l'ancre

Aller sur mon marqueur

Comment définir un lien vers un e-mail :

L'insertion d'un lien vers une adresse électronique se fait par la définition de l'e-mail de destinataire :

Exemple :

```
<a href = mailto = "nom@yahoo.fr">Envoyer un message</a>
```

Résultat :

Envoyer un message

Leçon 6 : Les tableaux

La première balise a utilisé, la balise <table> qui se terminera par </table>

Voici quelques attributs que l'on peut rajouter :

border = pour définir la bordure du cadre

cellspacing = pour définir l'espacement horizontal

cellpadding = pour définir l'espacement vertical

Exemple : <table border=10 cellspacing=5 cellpadding=20>

Comment faire plusieurs lignes :

La balise a utilisé est <tr> puis se termine par </tr>

Exemple :

```
<table><!--Début de Votre Tableau-->
```

```
<tr><!--Votre Première Ligne-->
```

```
</tr><!--Fin de Votre Première Ligne-->
```

```
<tr><!--Votre Deuxième Ligne-->
```

```
</tr><!--Fin de Votre Deuxième Ligne-->
```

etc...

```
</table><!--Fin de Votre Tableau-->
```

Comment faire plusieurs colonnes :

La balise à utiliser est <td> puis se termine par </td>

Exemple :

```
<table><!--Début de Votre Tableau-->
```

```
<tr><!--Votre Première Ligne-->
```

```
  <td><!--Votre Première Colonne-->
```

```
  </td><!--Fin de Votre Première Colonne-->
```

```
  <td><!--Votre Deuxième Colonne-->
```

```
  </td><!--Fin de Votre Deuxième Colonne-->
```

etc...

```
</tr><!--Fin de Votre Première Ligne-->
```

```
</table><!--Fin de Votre Tableau-->
```

Voyons un exemple concret :

```
<table border=10 cellspacing=5 cellpadding=20 bordercolor=#003366 bgcolor=#6699FF>
```

```
  <tr>
```

```
    <td>1</td>
```

```

        <td>2</td>
        <td>3</td>
        <td>4</td>
        <td>5</td>
    </tr>
    <tr>
        <td>6</td>
        <td>7</td>
        <td>8</td>
        <td>9</td>
        <td>10</td>
    </tr>
</table>

```

Ce qui nous donnent :

1	2	3	4	5
6	7	8	9	10

Comment colorer le tableau :

La balise `< BGCOLOR = "red">` définit la couleur de fond de tout le tableau :

Exemple : `< TABLE BGCOLOR = "red">` .

Comment colorer une cellule :

La balise `< BGCOLOR = "gold">` définit la couleur de fond d'une cellule :

Exemple : `< TD BGCOLOR = "red">` .

Comment modifier les dimensions du tableau

Deux balises permettent de définir les dimensions du tableau :

● `< TABLE width = 400>` .

● `< TABLE height = 100>` .

Exemple :

```
<table width=400 height=100 border=10 cellspacing=5 cellpadding=20 >
```

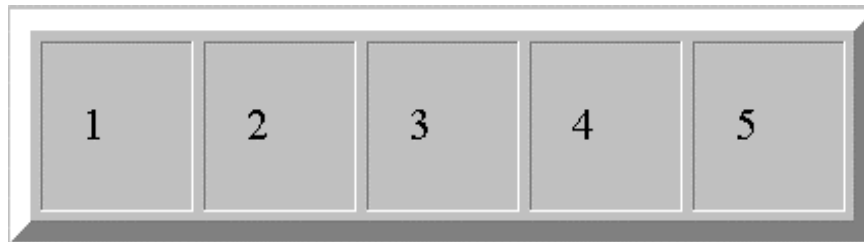
```
<tr>
```

```
<td>1</td>
```

```
<td>2</td>
```

```
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
```

Résultat :



1	2	3	4	5
---	---	---	---	---

Paramètre ALIGN :

Pour aligner tout le tableau , la balise est la suivante :

```
<TABLE = LEFT / CENTER / RIGHT >
```

Pour aligner les données dans les cellules , la balise est la suivante :

```
<TD = LEFT / CENTER / RIGHT >
```

Pour mettre une légende au tableau :

La balise est <CAPTION></CAPTION>

Exemple :

```
<TABLE>
```

```
<CAPTION>ici la légende du tableau</CAPTION>
```

```
</TABLE>
```

Leçon 7 : Document multi-cadres (Frames)

Généralité :

Le Frame est une fenêtre composée de plusieurs cadres . Souvent , on trouve un cadre pour le menu et un autre pour afficher le contenu des liens .

La structure générale d'un document multi-cadres est donc de la forme suivante :

```
<html>
```

```
<head>-----</head>
```

```
<frameset> Définition de la division en cadres et
```

```
des documents à afficher dans chacun d'eux .
```

```
</frameset>
```

```
</html>
```

- ◆ La balise <frameset> ne peut comporter que l'un des attributs Rows ou Cols <lignes ou colonnes>.
- ◆ Pour pouvoir poursuivre la subdivision en formes , il faut alors imbriquer d'autres commandes <frameset> à l'intérieur de la première .

Un exemple simple :

On veut diviser la fenêtre en 2 colonnes principales

<i>Code</i>	<i>Présentation</i>		
<pre><html> <head>-----</head> <frameset cols="50%", "50%"> <Frame src ="fichier1.html"> <Frame src ="fichier2.html"> </frameset> </html></pre>	<div style="border: 1px solid black; padding: 10px; text-align: center;"> <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="border: 1px solid black; width: 50%; padding: 5px;">Fichier1.html</td> <td style="border: 1px solid black; width: 50%; padding: 5px;">Fichier2.html</td> </tr> </table> <p>Bien sûr , dans les eux cadres, ce sont les contenus des 2 documents qui seront affichés</p> </div>	Fichier1.html	Fichier2.html
Fichier1.html	Fichier2.html		

Les principales balises

→<FRAMESET>.....</FRAMESET>

Cette divise la fenêtre en plusieurs fenêtres

→Paramètres ROWS et COLS permettent de partager la fenêtre horizontale ou verticale. Ils sont suivis d'une liste e valeurs séparées par des virgules , qui détermine le fractionnement de la fenêtre .

Par exemple :

- ROWS = "30%,70%" signifie un partage en 2 rangées dont les hauteurs sont les 30% et 70% de la hauteur disponible .

- COLS = "10%, *, 2 *" signifie un partage en 3 colonnes de largeurs 10% , 30% et 60%.

→ Paramètres SRC :

```
<FRAME SRC ="fichier.html"> </FRAME>
```

Définit le contenu d'un cadre . Si le fichier n'est pas local, il faut indiquer l'URL du document à afficher .

→ Autres paramètres :

∞ NAME : précise le nom du cadre

∞ SCROLLING = "yes"/"no"/"auto" : impose ou non la présence de barre de défilement .

∞ NORSIZE : interdit la modification de taille par l'utilisateur.

∞

Les liens dans les cadres

En l'absence d'indication complémentaire , on doit indiquer l'adresse du document : ... qui s'affichera dans le même cadre.

Si on veut afficher le document dans u autre cadre, il faut préciser le nom du cadre de destination : ...

Exemple :

```
<frameset cols="50%","50%">
```

```
    <Frame src ="fichier1.html" name ="cadre1">
```

```
    <Frame src ="fichier2.html" name ="cadre2">
```

```
</frameset>
```

Leçon 8 : Les Formulaires dans une page HTML

Introduction

Les formulaires Html (FORMS en anglais) sont l'ensemble des composants appelés champs qui permettent à l'utilisateur d'entrer des informations , d'exprimer ses choix , de saisir du texte ...

Bien sûr , les réactions de l'utilisateur doivent être prévues à l'avance , càd doivent être programmés , Ces programmes sont stockés sur le serveur pour que les données envoyées soient traitées par la suite .

La balise FORM

Toute la partie formulaire de la page doit se trouver entre les marqueurs `<FORM>.....</form>` .

Les types de champs

⇒ **Champ de texte simple** : Cette zone permet de saisir un texte court comme le nom & prénom ..etc.

`<INPUT TYPE = "texte" name ="nom du champ" value ="texte initial" size = longueur >`.

- Type= "texte" est facultatif, c'est le champ par défaut .
- Name = nom du champ (utile pour la programmation).
- Value : pour donner un texte visible au champ .
- Size : fixe la longueur visible du champ.
- Maxlength : pour limiter le nombre de caractères

Exemple :

Taper votre nom : `<input name = "nom" value="said">`

Résultat :

Taper votre nom :

⇒ **Champ mot de passe** : Pour saisir un mot de passe , on utilise le type password .

`<input type = "password" name = "pass" maxlegth = taille >`

Exemple :

Entrer votre mot de passe : `<input type="password" maxlength=5>`

Résultat :

Entrer votre mot de passe

⇒ **Zone de texte multi-ligne** : Pour permettre à l'utilisateur de saisir un texte de plusieurs lignes , on utilise le champ `<TEXTAREA>` :

`<TEXTAREA ROWS = "n" COLS = "n" >`

texte par défaut ...`</TEXTAREA>`

- ROWS précise le nombre de ligne
- COLS précise le nombre de colonnes
- Le texte par défaut est , l'utilisateur peut l'effacer .

⇒ **Liste de sélection** : Pour permettre un choix dans une liste e plusieurs options présentées sous forme de liste déroulante , on utilise la balise <SELECT>.

```
<select>
<option selected>élément 1 </option>
<option>élément 2 </option>
<option>élément 3 </option>
<option>élément 4 </option>
</select>

<option selected>élément 1 </option>
<option>élément 2 </option>
<option>élément 3 </option>
<option>élément 4 </option>
```

- <option> pour introduire chaque option de la liste
- <option selected> l'option sélectionné par défaut

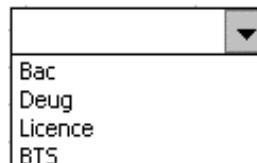
Exemple :

Quel est votre niveau d'étude :

```
<select>
<option selected>Bac </option>
<option>Deug</option>
<option>Licence</option>
<option>BTS</option>
</select>
```

Résultat :

Quel est votre niveau d'étude :



⇒ **Case à cocher** : A fin de répondre à des questions de type Oui/Non ou Vrai/Faux , on utilise des boîtes à cocher avec la balise :

```
<input type = "checkbox" name= "nom" checked> Question.
```

- `<input type = "checkbox">` est indispensable pour la case à cocher.
- Name = "...", chaque case doit avoir un nom spécifique .
- Checked pour cocher par défaut une case .

Exemple :

`<input type = "checkbox" name="nouveau">` Nouveau utilisateur

`<input type = "checkbox" name="nouveau">`Déjà inscrit

Résultat :

Nouveau utilisateur
 Déjà inscrit

⇒ **Boutons de sélection** : Le type radio est utilisé lorsque l'utilisateur doit faire un choix entre plusieurs . L'utilisateur pourra donc sélectionner qu'un seul bouton radio ..

`<input type = "radio" name= "nom" value = ".." checked>` Option 1 .

`<input type = "radio" name= "nom" value = ".." checked>` Option 2 .

- Le type = "radio" est indispensable pour insérer le bouton .
- Le name = "nom" : l'ensemble des boutons doit porter le même nom .
- Value = ".." chaque bouton radio doit posséder une valeur différente .
- Checked : pour désigner le bouton sélectionné par défaut .

Exemple :

Vous êtes :

`<input type = "radio" name = "titre" value = "t1" checked>` Mr

`<input type = "radio" name = "titre" value = "t2" >` Mlle

`<input type = "radio" name = "titre" value = "t3" >` Mde

Résultat :

Vous êtes :

Mr
 Mlle
 Mde

⇒ **Boutons de commande** :

Les boutons de commande permettent à l'utilisateur de déclencher des événements auxquels seront rattachées des fonctions .

- Bouton commun

`<input type = "button" name = " nom" value="Cliquer">`

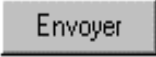
Cliquer

La valeur value est le texte qui apparaît sur le bouton .

- Bouton de validation

Il permet d'envoyer les informations saisies dans les champs , soit sur le serveur pour être traité , soit en message électronique .

```
<input type = "submit" name = " nom" value= "Envoyer">
```



- Bouton de réinitialisation

Ce bouton permet de vider les champs saisis

```
<input type = "reset" name = " nom" value= "Effacer">
```



Complément : transmission de données

La balise Form possède plusieurs attributs permettant de spécifier que doit être fait lors de la validation du formulaire .

- Action : <FORM action ="URL"> permet d'indiquer l'URL qui va recevoir les informations lorsqu'on clique sur envoyer .

L'URL est l'adresse d'un programme qui va récupérer les données et les traiter . Si le champ action est absent , l'URL sera celle du document courant .

Leçon 9 : HTML et multimédia :

1. Qu'appelle-t-on "module externe" ou "plug-in"?

Bien que le son et la vidéo soient le propre du multimédia, les navigateurs ne sont pas capables de les exécuter seuls : ils disposent d'une architecture ouverte qui leur permet de faire appel à des programmes supplémentaires externes. Ce sont les plug-in.

Ces programmes sont souvent installés avec le navigateur ou bien disponibles, généralement gratuitement, sur Internet.

2. Visualiser ses "plug-ins" sous NN

Il est possible de visualiser sous Netscape Navigator, l'ensemble des plug-in disponibles pour ce navigateur. Il suffit d'effectuer le cheminement suivant : Help>A propos des modules externes. Cet enchaînement lance un JavaScript capable d'analyser ce qui est disponible. Lancer cette commande et examiner le résultat.

3. Des dispositions différentes sous NN et IE

L'apparition des plug-ins revient à NetScape avec la version 2 de Navigator : grâce aux plug-ins, il est devenu alors possible de lire n'importe quel type de fichier à même la fenêtre du navigateur. La réaction de MicroSoft fut alors de reprendre le concept de plug-in et de l'approfondir : ce fut la technologie ActiveX (ActiveX est un langage de script utilisant les technologies Visual Basic et OLE). Les détails techniques sur cette différence d'approche et de mise en oeuvre sortent du cadre de ce cours ; nous retiendrons simplement que Netscape Navigator et Internet Explorer ne partagent pas la même approche des modules externes. L'effet principal est de nous compliquer (encore une fois !) la tâche.

4. Visualiser les objets ActiveX sous IE

Il est possible de visualiser sous Internet Explorer, l'ensemble des objets Active disponibles pour ce navigateur. Il suffit d'effectuer le cheminement suivant : Outils>Option Internet>Paramètres (sous l'onglet Général)> Afficher les Objets.

On pourra visualiser les propriétés de ces objets par un clic droit de la souris. On notera l'identifiant de classe (classid) utile pour l'appel de l'objet ActiveX dans la page HTML.

II. Balises d'insertion de contenu multimédia

1. La balise <embed>

Cette balise, initialement introduite par Netscape, ne fait même plus partie de la recommandation HTML Transitionnel du W3C. Elle permettait d'insérer dans la page HTML l'exécution du plug-in correspondant au type du fichier.

Le W3C recommande l'utilisation de la balise <object>.

2. La balise <object>

a. Syntaxe

La balise <object> est utilisée pour inclure images, séquences video, applets Java et interfaces VRML. Cette balise est prévue pour remplacer les balises à usage plus limité et <applet>, ainsi que les éléments propriétaires <embed> et <bgsound>, en dépit de problèmes relatifs à son support par les navigateurs. Dans la mesure du possible, on continuera donc à utiliser les balises et <applet>.

Cette balise accepte plusieurs attributs principaux :

- data : spécifie l'URI (adresse) de l'objet inséré.
- Les attributs width et height définissent les dimensions de l'objet. La valeur peut être exprimée en nombre de pixels ou en pourcentages de la taille de l'élément contenant l'objet. *Attention* : la plupart des navigateurs requièrent la présence de ces attributs.
- classid> peut être utilisé pour définir une implémentation permettant la gestion de l'objet. Cela est le cas pour les applets Java et Python, les contrôles ActiveX, ainsi que le montre l'exemple suivant...

```
<object classid="yahtzee.py" codetype="application/x-python" standby="Prêt pour un petit Yahtzee?"
title="Mon Yahtzee">
  <object classid="java:Yahtzee.class" codetype="application/java" width="400" height="250"
standby="Prêt pour un petit Yahtzee?" title="Mon Yahtzee">
    <object data="yahtzee.gif" type="image/gif" title="Une animation Yahtzee" width="200"
height="100"> Yahtzee est mon <em>jeu favori</em>. </object>
  </object>
</object>
```

Cet exemple montre aussi une solution afin de permettre aux navigateurs qui ne peuvent supporter le type d'objet, d'afficher une alternative. Dans cet exemple, le jeu écrit en Python est utilisé si le navigateur le supporte. Au cas où cela ne serait pas possible, une version Java est disponible. Une troisième possibilité est offerte avec l'affichage d'une image, et enfin une dernière alternative est disponible avec du texte brut. Cette possibilité préserve la compatibilité avec les plus anciens navigateurs, qui ignoreront les éléments <object>, et se contenteront d'afficher le texte.

L'exemple précédent utilise également les attributs type et codetype pour aider les navigateurs à déterminer la nature des fichiers en jeu : cela leur évite ainsi de tenter de charger un type de fichier qu'ils ne savent pas gérer. L'attribut type indique le type de média de la ressource référencée par l'attribut data, tandis que l'attribut codetype indique le type de média de la ressource référencée par l'attribut classid.

L'attribut standby est aussi utilisé dans l'exemple précédent. Il fournit un court texte affiché dans la fenêtre pendant le chargement de l'objet.

b. Élément <param>

L'élément <object> peut contenir des éléments <param>, avant tout autre élément-enfant, permettant de spécifier un certain nombre de propriétés qui sont transmises aux objets multimédia chargés. Par exemple,

```
<object classid="java:Clock.class" codetype="application/java" width="100" height="100" title="Une
horloge en temps réel!" standby="Quelle heure est-il?">
  <param name="type" value="analog"></param>
  <param name="bgcolor" value="white"></param>
  <param name="fgcolor" value="navy"></param>
</object>
```

Les applet dans pages web :

Applet	<code><APPLET></APPLET></code>	
Nom de fichier applet	<code><APPLET CODE="***"></code>	
Paramètres	<code><APPLET PARAM NAME="***"></code>	
Position de l'applet	<code><APPLET CODEBASE="URL"></code>	
Identificateur Applet	<code><APPLET NAME="***"></code>	(pour faire référence ailleurs dans une page)
Alternative en texte	<code><APPLET ALT="***"></code>	(pour les navigateurs non-java)
Alignement	<code><APPLET ALIGN="LEFT RIGHT CENTER"></code>	
Grandeur	<code><APPLET WIDTH=? HEIGHT=?></code>	(en pixels)
Espacement	<code><APPLET HSPACE=? VSPACE=?></code>	(en pixels)

II – CSS

Chapitre 1: Concept et utilité

1.1 Présentation

Le concept des feuilles de style n'est pas à proprement parler une nouveauté dans le domaine de la publication Html. Introduit en 1997 par Microsoft avec son Internet Explorer 3.0 (mais elles existaient déjà avec Arena sous Unix), ces feuilles de style n'ont connu qu'un intérêt limité du fait que celles-ci n'étaient pas reconnues par Netscape Navigator 3.0.

Depuis les choses ont bien changé. D'abord les browsers 4.0 de Microsoft et de Netscape reconnaissent tous deux les feuilles de style et surtout, la norme Html 4.0 en a repris le concept (CSS version 1) et le recommande d'ailleurs vivement aux "Web designers".

1.2 Concept

Dans un document d'une certaine importance, il arrive fréquemment que l'on attribue à certains éléments des caractéristiques de mise en forme identiques. Par exemple, les noms de chapitres seront mis en police Arial, en gras et en couleur bleue.

On peut imaginer que l'on puisse donner à cette définition de mise en forme un nom soit "titre" et qu'à chaque nouveau chapitre, plutôt que d'écrire chaque fois le nom du titre et puis de devoir le mettre en Arial, gras, bleu, l'on puisse dire à l'ordinateur, nom du chapitre mais dans la mise en forme que j'ai défini sous le nom de "titre". Cette définition de mise en forme particulière, on va l'appeler feuille de style.

Le concept de feuilles de style [Style Sheets] est né. Il existait déjà dans les traitements de texte comme dans Word de Microsoft (comme par hasard...). Allez dans le menu Format de Word, vous y trouvez Style ! Il ne restait plus qu'à coupler le concept au langage Html par des propriétés spécifiques.

<H1><FONT	STYLE des titres
COLOR=blue>Titre1</H1>	STYLE des sous-titres
<H2>- A -	STYLE du texte
</H2>	STYLE des titres
<H3><FONT	STYLE des sous-titres
COLOR="red">...a...</H3>	STYLE du texte
<H1><FONT	
COLOR=blue>Titre2</H1>	
<H2>- B-	
</H2>	
<H3><FONT	
COLOR="red">...b...</H3>	

Vous remarquez que l'on parle de feuilles de style [style sheets] car le but du jeu est d'en définir plusieurs. On parle aussi de feuilles de style en cascade [Cascading Style Sheets ou CSS] car en cas de styles identiques, un ordre de priorité sera déterminé par le browser (voir FAQ).

Précisons pour terminer que les feuilles de style ne sont pas une composante directe du langage Html mais un développement à part dans la publication de pages Web.

1.3 Utilité et avantages

- Séparation du contenu et de la mise en forme.
- Cohésion de la présentation tout au long du site avec les feuilles de style externes.
- Modifier l'aspect d'un page ou d'un site sans en modifier le contenu et cela en quelques lignes plutôt que de devoir changer un grand nombre de balises.
- Un "langage" neuf, compréhensible, simple et logique par rapport au Html et à ses différentes versions.
- Une façon d'écriture concise et nette par rapport au Html qui devient vite fouillis.
- Réduire le temps de chargement des pages.
- Palier certaines insuffisances du langage Html (contrôle des polices, contrôle de la distance entre les lignes, contrôle des marges et des indentations (sans devoir utiliser de tableaux ou de balise <DD>...) et ainsi augmenter la créativité des écrivains du Web.
- Permettre le positionnement au pixel près du texte et/ou des images sans passer par les "layers" exclusifs à Netscape 4.0.

1.4 Compatibilité

Les feuilles de style fonctionnent sous :
Explorer 3.0 mais de façon incomplète
Explorer 4.0
Netscape 4.0

Attention !!! Les feuilles de style ne sont pas reprises par Netscape 3.0.

Chapitre 2 : Définition d'un style

La définition de base d'un style est simple :

balise { propriété de style: valeur; propriété de style: valeur }

Exemple :

H3 { font-family: Arial; font-style: italic }

Donc ici, la balise H3 sera en Arial et en italique. Et dans votre document, toutes les balises <H3> auront comme style Arial et italique.

Simple! Mais de nombreux commentaires s'imposent :

- Les feuilles de style portent sur des balises principalement et quelques autres éléments comme par exemple A:link pour un lien non-visité et A:visited pour un lien visité. Comme balises souvent utilisées avec des feuilles de style, on peut citer les en-têtes Hn, P, BODY...
- Les propriétés de style sont entourées par des "{" et pas des [ou des parenthèses.

- Le couple "propriété de style/valeur" est séparé par un double-point (:).
- Chaque couple "propriété de style/valeur" est séparé par un point-virgule (;).
- Il n'y a pas de limite pour le nombre de couples "propriétés de style/valeur".
- L'espace entre propriétés de style et valeur n'est pas obligatoire mais aide fortement à la lisibilité du code source.
- Pour la lisibilité toujours, vous pouvez écrire vos styles sur plusieurs lignes :


```
H3 {font-family: Arial;
font-style: italic;
font-color: green}
```
- On peut attribuer plusieurs valeurs à une même propriété. Dans ce cas, on séparera les différentes valeurs par des virgules.


```
H3 {font-family: Arial, Helvetica, sans-serif}
```
- On peut attribuer un même style à plusieurs balises (séparées par des virgules).


```
H1, H2, H3 {font-family: Arial; font-style: italic}
```

Chapitre 3 : Styles internes

Il faut maintenant incorporer les styles dans le document Html. Commençons par le plus simple, soit l'incorporation à l'intérieur d'une page. D'où le titre "Styles internes".

3.1 A l'intérieur des balises <HEAD></HEAD>

Cette façon de procéder est de loin la plus commune et la plus logique. D'abord parce que les balises HEAD contiennent des informations pour le browser et les feuilles de style appartiennent à celles-ci. Ensuite parce que l'on rejoint ainsi l'essence même des feuilles de style qui est de séparer les éléments de mise en forme du contenu.

```
<HTML>
<HEAD>
<STYLE type="text/css">
<!--
La ou les feuille(s) de style
-->
</STYLE>
</HEAD>
<BODY>
```

- La balise <STYLE> avertit le navigateur que l'on va utiliser des feuilles de style.
- L'attribut type="text/css" informe que ce qui suit est du texte et qu'il s'agit de cascading style sheets (css). Pour l'instant, il s'agit de la seule possibilité mais on peut prévoir à l'avenir d'autres versions de ce "langage".
- La balise Html de commentaires <!-- ... --> empêche que les browsers qui ne connaissent pas les feuilles de style, tentent d'interpréter ces instructions. Les informations à l'intérieur des tags de commentaires seront ignorées par ces browsers.
- Pour vos propres commentaires à propos des feuilles de style, on utilisera la convention désormais classique (C, C++, Javascript...) de /* commentaires */.

3.2 A l'intérieur des balises <BODY></BODY>

On peut aussi utiliser, au coup par coup, les feuilles de style dans le corps (BODY) du document. Cette façon de faire nous paraît illogique et peu conforme à l'esprit des feuilles de style qui est de définir un style déterminé valable pour la globalité du document. Mais elle existe pour quelques utilisations spécifiques...

```
<HTML>
<BODY>
<H1 style="font-family: Arial; font-style: italic"> blabla </H1>
</BODY>
</HTML>
```

Signalons :

- que le style Arial, italique n'affectera que cette seule balise H1.
- que la syntaxe est légèrement différente de la précédente.
- que l'écriture : `<STYLE type="text/css">H1 { "font-family: Arial; font-style: italic" }</STYLE>` fonctionne aussi.

Chapitre 4 : Styles externes

C'est déjà bien de pouvoir définir une présentation de style valable pour une page (styles internes). Mais CSS nous propose mieux. Définir une présentation de style valable pour plusieurs pages et même pour toutes les pages d'un site. Ce qui est possible, en créant une page externe qui regroupera toutes les feuilles de style, et en reliant chaque page à cette page de style.

On crée d'abord, dans le répertoire du site, un fichier avec l'extension .css soit styles.css qui contiendra toutes les feuilles de style.

```
<HTML>
<HEAD>
--- Les différentes feuilles de style ---
</HEAD>
<BODY>
</BODY>
</HTML>
```

Ensuite, on crée une page normale soit page1.htm (bien entendu dans le même répertoire que le fichier styles.css).

```
<HTML>
<HEAD>
<LINK rel=stylesheet type="text/css" href="styles.css">
</HEAD>
```

- La balise `<LINK>` avertit le browser qu'il faudra réaliser un lien.
- L'attribut `rel=stylesheet` (sans s et sans guillemets) précise qu'il y trouvera une feuille de style externe.
- L'attribut `type="text/css"` précise que l'information est du texte et du genre cascading style sheets (css).
- L'attribut classique de lien `href=" ... "` donne le chemin d'accès et le nom du fichier à lier.

Chapitre 5 : Les classes et les ID

5.1 Notion de classes

Mais on désire parfois affecter des styles différents à une même balise. Pas de problèmes, les feuilles de style vous proposent la solution des classes [class].

La syntaxe est ici aussi des plus simple.

La définition d'un style était :

```
balise { propriété de style: valeur }
```

Elle devient :

```
balise.nom_de_classe { propriété de style: valeur }  
remarquez le point entre balise et nom_de_classe
```

Ou, comme la mention de la balise est facultative,

```
.nom_de_classe { propriété de style: valeur }
```

Attention! L'emploi du point (.) devant le nom de classe est indispensable.

Pour appeler l'effet de style dans le document, on ajoute le nom de la classe à la balise.

```
<balise class="nom_de-classe"> .... </balise>
```

Un exemple ?

Je souhaite mettre ce qui est important dans le texte en gras et en bleu. Je crée la classe .essentiel :

```
.essentiel { font-weight: bold; font-color: #000080 }
```

Et dans le document Html, il suffit d'appeler la classe .important quand cela se révèle nécessaire :

```
<P class=".essentiel"> ... blabla ... </P>  
<H1 class=".essentiel">Titre 1</H1>  
<TABLE><TR><TD class=".essentiel">cellule</TD></TD>...
```

5.2 Notion des ID

Comme la convention nom/point/nom est utilisée aussi en Javascript (voir Apprendre le Javascript du même auteur : www.ccim.be/ccim328/js/index.htm) , il a fallu trouver une autre convention d'écriture lorsqu'on désire utiliser les feuilles de style avec du Javascript. Ce sont les ID, aussi appelés les identifiants.

Les ID fonctionnent exactement comme les classes. Pas mieux, pas plus. C'est la même chose!

La syntaxe est :

```
#nom_de_ID { propriété de style: valeur }
```

Et pour l'appeler :

```
<balise id="nom_de_ID"> .... </balise>
```

Notons qu'on ne pourra effectuer qu'un seul appel à #nom_de_ID par page. Ainsi,

```
Pour #essentiel{ ... }
```

```
<P id=essentiel> est correct.
```

Mais si on rencontre dans la même page

```
<H1 id=essentiel> ce n'est plus correct !
```

Si vous pensez utiliser des feuilles de style, mais sans vous compliquer la vie avec des scripts, oubliez au plus vite ID et utilisez exclusivement les classes.

Si par contre vous souhaitez utiliser des scripts avec les feuilles de style pour faire du DHTML par exemple (voir plus loin dans le site), la notion de ID vous sera alors indispensable.

Chapitre 6 : et <DIV>

6.1 Utilité

Dernier point, il fallait penser à un système pour "déconnecter" certains morceaux de paragraphe ou plusieurs paragraphes de cette logique d'écriture avec des feuilles de style. Ce sont respectivement les balises SPAN et DIV qui créaient ainsi des petits blocs particuliers dans le document sans devoir repasser par les éléments structurels du Html classique. Notons que SPAN et DIV s'utilisent toujours avec les classes et les ID.

6.2 SPAN

La balise ... permet d'appliquer des styles à des éléments de texte d'un paragraphe ou si vous préférez à un morceau de paragraphe. Ainsi je voudrais écrire :

Un monde de **géants**.

```
<HTML>
<HEAD>
<STYLE type="text/css">
.element{font-size: x-large; color: navy}
</STYLE>
</HEAD>
<BODY>
<P>Un monde de <SPAN class=element>géants</SPAN>.</P>
</BODY>
</HTML>
```

6.3 DIV

La balise <DIV> ... </DIV> permet de regrouper plusieurs paragraphes ou si vous préférez, de délimiter une zone comportant plusieurs paragraphes.

```
<HTML>
<HEAD>
```

```
<STYLE type="text/css">
.zone{font-size: x-small}
</STYLE>
</HEAD>
<BODY>
La balise <DIV>
<DIV class=zone>
<P>Commentaire :</P>
<P>N'oubliez pas l'attribut class!</P>
</DIV>
</BODY>
</HTML>
```

Donne comme résultat :

La balise <DIV>

Commentaire :

N'oubliez pas l'attribut class!

Chapitre 7 : Positionner avec CSS

Quel concepteur de pages Web n'a pas laissé échapper quelques jurons bien sentis en essayant, à grand renfort de tableaux, de placer précisément du texte ou une image là où il le désirait ?

Le miracle est arrivé ! Outre le balise <LAYER> (mais qui n'est comprise que par Netscape 4.0), il est désormais possible de positionner, au pixel près, du texte ou une image avec les feuilles de style.

Notons que ce positionnement n'est possible que sous les versions 4 de Netscape et d'Explorer. Et que cette technique est encore un peu hasardeuse à ce jour, surtout sur le plan de la compatibilité avec les deux browsers susnommés.

Le positionnement des éléments par les feuilles de style est repris sous la spécification CSS-P.

7.1 Position absolue ou relative

La position absolue {position: absolute} se détermine par rapport au coin supérieur gauche de la fenêtre du browser. Les coordonnées de ce point sont top = 0 et left = 0. Les coordonnées d'un point s'expriment en pixels, de haut en bas pour top et de gauche à droite pour left.

La position relative {position: relative} se détermine par rapport à d'autres éléments de la page, par exemple un élément du code Html.

Précisons que l'on utilisera presque toujours le positionnement absolu car plus facile et plus sûr.

7.2 Positionner du texte

Plaçons en position absolue le texte "Publication Html" à 50 pixels du haut de la fenêtre (top) et à 150 pixels à gauche (left).

```
<HTML>
<HEAD>
<STYLE type="text/css">
.pub{position: absolute; top: 100px; left: 25px;
color: yellow; font-size: x-large; font-weight: bold;}
</STYLE>
</HEAD>
<BODY>
<DIV class=pub> Publication Html </DIV>
</BODY>
</HTML>
```

Ajoutons que plusieurs encodages sont possibles.

7.3 Positionner une image

Plaçons l'image htmlplus.gif en position absolue à 50 pixels de haut de la fenêtre (top) et à 100 pixels à gauche (left). Les dimensions de l'image sont width=242 et height=84.

```
<HTML>
<BODY>
<span style="position: absolute; top: 50px; left: 100px; width: 242px;
height: 84px;">
<IMG src="htmlplus.gif">
</span>
</BODY>
</HTML>
```

Spécifiez toujours les propriétés width et height avec les feuilles de style car par défaut, Netscape 4.0 et Explorer 4.0 ne réagissent pas de la même façon.

Ajoutons que plusieurs encodages sont possibles.

7.4. Superposer du texte sur une image

Reprenons l'image htmlplus.gif et on y superposera le nom de l'auteur de ce tutorial, au pixel près dans la barre qui souligne le terme Html.

```
<HTML>
<BODY>
<span style="position: absolute; top: 50px; left: 100px; width: 242px; height: 84px;">
<IMG src="htmlplus.gif">
</span>
<span style="position: absolute; top: 96px; left: 145px;
color: yellow; font-size: x-small; font-weight: bold;">
Van Lancker Luc
</span>
</BODY>
</HTML>
```



Ajoutons que plusieurs encodages sont possibles.

Chapitre 8 : Mini FAQ sur les styles

8.1 Les feuilles de style sont-elles "case sensitive" ?

Non, les feuilles de style ne sont pas sensibles à la case [case insensitive]. Ecrire CLASS ou class ou Class est donc équivalent. Cependant les éléments qui ne sont pas sous le contrôle des feuilles de style comme les noms de police ou les URLs peuvent être case sensitive. Pour le système d'exploitation (et je pense à Unix), Arial n'est peut-être pas égal à arial, de même IMAGE.gif n'est pas forcément égal à image.gif.

8.2 Quels caractères peut-on utiliser en CSS ?

Les noms des feuilles de style, des sélecteurs, des classes et ID peuvent contenir les lettres de a-z ou de A-Z, les chiffres de 0-9, le trait d'union et le caractère _. Les noms ne peuvent commencer par un chiffre ou un tiret. La documentation officielle affirme que les caractères spéciaux ASCII 160-255 peuvent être utilisés. Mais cela ne fonctionne pas chez moi. On prendra vite l'habitude (voir les langages de programmation) de les éviter.

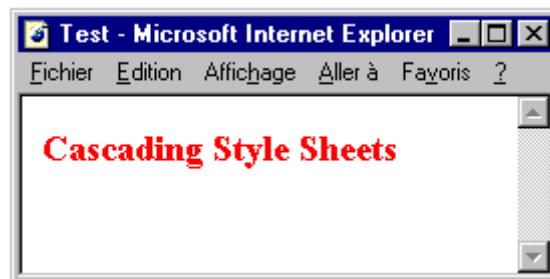
8.3 Peut-on utiliser dans la même page, à la fois des éléments de style CSS et Html ?

Oui. Les feuilles de style seront ignorées par les browsers qui ne supportent pas les feuilles de style. Et c'est tant mieux pour la compatibilité de votre site sous les différents navigateurs.

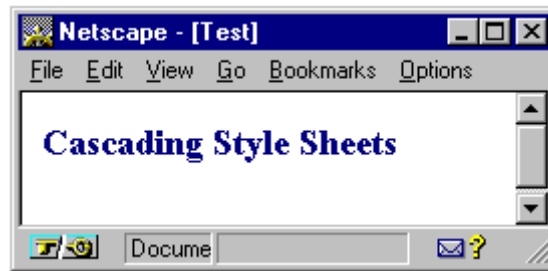
8.4 Qui l'emportent : les attributs Html ou les propriétés CSS ?

Les propriétés des feuilles de style l'emportent sur les attributs Html. Si les deux sont spécifiés, les attributs Html seront affichés avec les browsers qui ne supportent pas les CSS et n'auront aucun effet avec les browsers qui supportent les feuilles de style.

Ainsi, `<H3 style="color: red">Cascading Style Sheets</H3>` apparaîtra en rouge sous un browser compatible CSS (par exemple Microsoft Explorer).



et en bleu sous un browser qui ne reconnaît pas les feuilles de style (par exemple Netscape 3.0).



Ce qui est très bien pour la compatibilité mais qui ne simplifie pas la clarté de l'écriture.

8.5 Et s'il y a concurrence entre plusieurs éléments de style ?

En cas de concurrence entre plusieurs éléments de style, intervient alors la notion de "cascade" ou d'ordre de priorité.

La concurrence entre plusieurs éléments de style peut provenir des différentes possibilités de localisation de feuilles de style :

- dans un fichier externe avec l'extension css.
- dans la balise HEAD du document.
- dans le BODY du document.

La règle de priorité appliquée par le browser sera d'appliquer pour la présentation du document la feuille de style la plus proche de l'élément. Ainsi, un style spécifié dans le BODY sera retenu par rapport à un style déclaré dans un fichier externe.

Il y a cependant moyen de contourner ces règles de priorité par la déclaration ! important;. Du genre BODY {background: white ! important; color: black}. Nous en resterons là dans le cadre de ce tutorial.

Chapitre 9 Liste des propriétés

La liste complète (et officielle) des propriétés et recommandations concernant les feuilles de style version 1, peut être trouvée à l'adresse <http://www.w3.org/pub/WWW/TR/REC-CSS1>. En voici une sélection :

9.1. Les styles de police

font-family

- définit un nom de police ou une famille de police
- <nom> ou <famille>
- police précise (Arial, Times, Helvetica...) ou
- famille (serif, sans-serif, cursive, fantasy, monospace)
- H3 {font-family: Arial}

font-style

- définit le style de l'écriture
- normal ou italique ou oblique

H3 {font-style: italic}

font-weight

définit l'épaisseur de la police
normal ou bold ou bolder ou lighter ou valeur numérique soit
(100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900)
P {font-weight: bold}

font-size

définit la taille de la police
xx-small ou x-small ou small ou médium ou large ou x-large ou xx-large
ou larger ou smaller
ou taille précise en points (pt), inches (in), centimètres (cm), pixels (px)
ou pourcentage (%)
P {font-size: 12pt}

font-variant

définit une variante par rapport à la normale
normal ou small-caps
P {font-variant: small-caps}

font

raccourci pour les différentes propriétés de police
P {font: bold italic}

9.2. Les styles du texte

text-align

définit l'alignement du texte
left ou center ou right
H1 {text-align: center}

text-indent

définit un retrait dans la première ligne d'un bloc de texte
souvent utilisé avec <P>, n'oubliez pas dans ce cas </P>.
spécifié en inches (in) ou en centimètres (cm) ou en pixels (px)
P {text-indent: 1cm}

text-decoration

définit une décoration (?) du texte, soit barré, clignotant, etc.
blink ou underline ou line-through ou overline ou none
A:visited {text-decoration: blink}

text-transform

définit la casse du texte (majuscule, minuscule)
uppercase (met les caractères en majuscules) ou
lowercase (met les caractères en minuscules) ou
capitalize (met le premier caractère en majuscule)
P {text-transform: uppercase}

color

définit la couleur du texte
par exemple en hexadécimal
H3 {color: #000080}

word-spacing

définit l'espace entre les mots
en points (pt), inches (in), centimètres (cm), pixels (px)
ou pourcentage (%)
P {word-spacing: 5pt}

letter-spacing

définit l'espace entre les lettres
spécifié en points (pt), inches (in), centimètres (cm), pixels (px)
ou pourcentage (%)
P {letter-spacing: 2pt}

line-height

définit l'interligne soit l'espace entre les lignes du texte
en points (pt), inches (in), centimètres (cm), pixels (px)
ou pourcentage (%)
P {line-height: 10pt}

width

détermine la longueur d'un élément de texte ou d'une image
en points (pt), inches (in), centimètres (cm), pixels (px)
ou pourcentage (%)
H1 {width: 200px}

height

détermine la hauteur d'un élément de texte ou d'une image
en points (pt), inches (in), centimètres (cm), pixels (px)
ou pourcentage (%)
H1 {height: 100px}

white-space

espace ou blanc
normal ou pre ou nowrap
PRE {white-space: pre}

9.3. Les arrière-plans

background-color

définit la couleur de l'arrière-plan
couleur (par exemple en hexadécimal) ou transparent
H1 {background-color: #000000}

background-image

définit l'image de l'arrière-plan
URL de l'image
BODY {background-image: image.gif}

background-repeat

définit la façon de répéter l'image d'arrière-plan
repeat ou no-repeat ou repeat-x (x = nombre de répétitions horizontales) ou
repeat-y (y = nombre de répétitions verticales)
P {background-image: image.gif; background-repeat: repeat-4}

background-attachment

spécifie si l'image d'arrière-plan reste fixe avec les déplacements de l'écran
scroll ou fixed
BODY {background-image: image.gif; background-attachement: fixed}

background-position

spécifie la position de l'image d'arrière-plan par rapport au coin
supérieur gauche de la fenêtre
{ 1, 2 }
{ top ou center ou bottom , left ou center ou right }
ou en points (pt), inches (in), centimètres (cm), pixels (px)
ou pourcentage (%)
BODY {background-image: img.gif; background-position: right top}

background

raccourci pour les différentes propriétés d'arrière-plan
P {background: image.gif fixed repeat}

9.4. Les marges

margin-top

détermine la valeur de la marge supérieure
en unité de longueur ou auto
{ margin-top: 5px }

margin-right

détermine la valeur de la marge droite
en unité de longueur ou auto
{ margin-right: 5px }

margin-bottom

détermine la valeur de la marge inférieure
en unité de longueur ou auto
{ margin-bottom: 5px }

margin-left

détermine la valeur de la marge gauche
en unité de longueur ou auto
{ margin-left: 5px }

margin

regroupe les différentes propriétés de la marge

9.5. Les bords et les "enrobages"

border-top-width

donne l'épaisseur du bord supérieur
thin ou medium ou thick ou spécifié par l'auteur
H3 {border-top-width: thin }

border-right-width

donne l'épaisseur du bord droit
thin ou medium ou thick ou spécifié par l'auteur
H3 {border-right-width: medium }

border-bottom-width

donne l'épaisseur du bord inférieur
thin ou medium ou thick ou spécifié par l'auteur
H3 {border-bottom-width: thick }

border-left-width

donne l'épaisseur du bord gauche

thin ou medium ou thick ou spécifié par l'auteur
H3 {border-left-width: 0.5cm}

border-width

regroupe les différentes propriétés de border-width

border-color

détermine la couleur de la bordure
H3 {border-color: yellow}

border-style

détermine le style du trait de la bordure
none ou solid ou dotted ou dashed ou double
ou groove ou ridge ou inset ou outset
{border-style: solid dashed}

border

regroupe toutes les propriétés des bords

padding-top

valeur de remplissage haut entre l'élément et le bord
en points (pt), inches (in), centimètres (cm), pixels (px)
ou pourcentage (%)
H3 {padding-top: 3px}

padding-right

valeur de remplissage droite entre l'élément et le bord
en points (pt), inches (in), centimètres (cm), pixels (px)
ou pourcentage (%)
H3 {padding-right: 3px}

padding-bottom

valeur de remplissage bas entre l'élément et le bord
en points (pt), inches (in), centimètres (cm), pixels (px)
ou pourcentage (%)
H3 {padding-bottom: 3px}

padding-left

valeur de remplissage gauche entre l'élément et le bord
en points (pt), inches (in), centimètres (cm), pixels (px)
ou pourcentage (%)
H3 {padding-left: 3px}

padding

regroupe les différentes propriétés de remplissage

9.6. Les listes

list-style-type

détermine le type de puces ou de numérotation

disc ou circle ou square

decimal ou lower-roman ou upper-roman ou lower-alpha ou upper-alpha

OL {list-style-type: square}

list-style-image

permet de remplacer les puces par une image

url ou none

OL {list-style-image: image.gif}

list-style-position

spécifie si les puces sont à l'intérieur ou à l'extérieur du texte

inside ou outside

UL {list-style-position: inside}

list-style

regroupe toutes les propriétés de liste

III – JAVASCRIPT

A - Programmation JavaScript : généralités :

Javascript est un langage de programmation très récent, créé par les sociétés Netscape et Sun Microsystems vers la fin de l'année 1995.

Son objectif principal : introduire de l'interactivité avec les pages HTML, et effectuer des traitements simples sur le poste de travail de l'utilisateur.

Le moyen : introduire de petits programmes, appelés **SCRIPTS**, dans les pages HTML.

Jusqu'ici la programmation ne pouvait être exécutée que sur le serveur. La possibilité d'inclure des programmes dans les pages HTML et de les exécuter directement sur le poste client est intéressante, car elle permet de décharger le serveur de ce travail et ... d'éviter les attentes des réponses aux requêtes adressées via le Réseau.

Le code du programme est interprété par le navigateur client (qui reçoit la page). Il ne peut pas être exécuté indépendamment, ce qui le limite comme langage de programmation, contrairement à JAVA (à ne pas confondre !).

C'est un langage basé sur des objets, très simple et conçu, en principe, pour des non spécialistes. En résumé, voici ses principales caractéristiques :

- JS est un langage de programmation **structurée** qui concourt à **enrichir le HTML**, à le rendre plus "intelligent" et interactif.
- Le code JS est **intégré** complètement dans le **code HTML**, et interprété par le navigateur client
- JS contient des **gestionnaires d'événement** : il reconnaît et réagit aux demandes de l'utilisateur, comme un clic de la souris, une validation de formulaire, etc...

Mais c'est un langage limité :

- ce n'est pas un langage de programmation à part entière, indépendant.
- c'est un langage de script, dépendant de HTML, c'est une **extension de HTML**. Sa syntaxe ressemble à Java, car elle reprend celle du langage C, mais il est en fait très différent. Java est un langage complet , compilé et complètement autonome du langage HTML
- ce n'est pas véritablement un langage orienté objet (pas d'héritage de classe , ni de polymorphisme ..)

Ecriture et exécution du code JS

On peut placer du code JS dans une page HTML à 3 endroits et sous des formes bien différentes.

1. **Entre les balises <SCRIPT>....</SCRIPT>** dans la section d'en-tête, ou dans le corps de la page.

```
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
... code
//code : instructions, déclarations de fonctions, etc..
</SCRIPT>
</HEAD>
```

- Le code inclus dans la séquence **<SCRIPT>** est évalué au début du chargement de la page.
- S'il est inclus dans la section **<HEAD>**, il n'est pas exécuté tout de suite.

- Par contre, s'il fait partie du corps du document, il est immédiatement exécuté en même temps que le code HTML est interprété.
- Il est nécessaire d'inclure les déclarations de fonctions dans la section `<HEAD>..</HEAD>`. En effet, les fonctions doivent être connues dès le chargement du document, mais ne doivent être exécutées que lors d'un appel explicite de l'utilisateur, le plus souvent en réponse à un événement (voir ci-dessous).

2. Associé à une balise HTML qui gère un événement.

Le code JS est généralement inséré sous forme d'un appel de fonction, affectée à un *gestionnaire d'événement* qui apparaît comme un nouvel attribut d'un composant de formulaire. L'exécution du code est alors provoquée par appel d'une fonction JS (préalablement déclarée) dont l'exécution constitue une réponse à l'événement.

Un événement survient à l'initiative de l'utilisateur, par exemple en cliquant sur un bouton, ou après la saisie du texte dans un champ de formulaire.

Ecriture générale

```
<balise ... onEvenement="code JS" | "fonction JS">
```

où

- *balise* est le nom de certaines balises, souvent des composants de formulaire
- *onEvenement* est un nouvel attribut de la balise
- Bien entendu il faut connaître les associations entre événements et balises "sensibles" à ces événements.

Ici on peut voir les correspondances entre [les balises et les événements](#)

Exemple

Le code HTML suivant crée un bouton de nom "bouton1", sur lequel est écrit "Calculer". Quand l'utilisateur appuie sur ce bouton, l'événement *onClick* est déclenché et la fonction *calculer()* est appelée. Son code, déclaré dans l'en-tête est alors exécuté.

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
  function calculer() {
    ....// code.....
  }
</SCRIPT>
</HEAD>

<BODY>
<FORM>
<INPUT TYPE="button" VALUE="Calculer" onClick="calculer()">
</FORM>
</BODY>
</HTML>
```

3. Associé au pseudo-protocole *javascript*: dans une URL.

Cette pseudo-URL permet de lancer l'exécution d'un script écrit en JS, au lieu d'être une requête pour obtenir un nouveau document (comme avec les protocoles habituels http: , ftp:)

Ecriture générale

```
<A HREF="JavaScript:code JS" | "appel fonction JS">texte|image activable</A>
```

L'opérateur**void**

Pour empêcher que le code ou la fonction appelée dans l'URL JavaScript, ne remplace le document courant, on applique l'opérateur void, qui neutralise toute valeur ou tout effet de retour.

```
<A HREF="JavaScript:void( appel-fct(..) )">.....</A>
```

Exemple

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
  function calculer() {
    ....// code.....
  }
</SCRIPT>
</HEAD>
<BODY>
.....
<A HREF="JavaScript:calculer()">Pour calculer</A>
.....
</BODY>
```

Quelques remarques

- JS fait la distinction entre majuscules et minuscules, contrairement aux balises HTML. C'est une source fréquente d'erreur.
- Pour comprendre le code, inclure des commentaires abondants :
// pour une simple ligne de commentaires
/**/ pour les encadrer sur plusieurs lignes.
- Quand on ne définit pas de fonctions, on peut inclure le code directement dans la section <BODY>.
- On peut (depuis Netscape 3) placer le code dans un fichier spécifique d'extension **.JS** :
<SCRIPT LANGUAGE=JavaScript SRC=source.js>
</SCRIPT>
où *source.js* doit être un fichier accessible au moment de l'exécution, dans le répertoire courant ou à une adresse URL précisée.
Un tel fichier externe permet de réutiliser le code dans de multiples pages, sans avoir à l'inclure dans le source.

Exemple

```
<html>
<head>
<script>
  function saluer() {
    alert("Bonjour tout le monde !");
  }
</script>
</head>
```

```

<body>
<H4>Exécution immédiate</H4>
<script>
  alert("Bonjour tout le monde !");
</script>
<H4>Exécution sur événement onClick</H4>
<form>
  <input type="button" name="Bouton1" value="Salut" onClick="saluer()">
</form>
<H4>Exécution sur protocole javascript:</H4>
<A HREF = "javascript:saluer()">pour saluer</a>
</body>
</html>

```

Voici quelques exemples introductifs et progressifs

- accompagnés de commentaires et
- de corrigés écrits eux-mêmes en JavaScript

I. Afficher du texte :

```

EXEMPLE 1
<HTML>
<HEAD> <TITLE> Exemple 1 </TITLE> </HEAD>
<BODY>
  <SCRIPT LANGUAGE="JavaScript">
    var bonjour = "Bonjour !";
    var question = "Comment allez vous ";
    var phrase = bonjour + "<BR>" + question;
    document.write(phrase, "aujourd'hui ?");
  </SCRIPT>
</BODY>
</HTML>

```

A la lecture de ce code JS, comprendre :

- la *déclaration* et l'*initialisation* (c'est-à-dire : donner une valeur initiale) des variables bonjour, question, et phrase.
- l'*affectation* d'une valeur à une variable, avec l'**opérateur =**, sur le modèle :
variable = valeur (ou expression)
- l'instruction **document.write("texte")** est l'instruction générale d'affichage de "texte" dans le même document (le document "courant").
- les constantes "texte" sont placées entre " " ; à défaut JS croit qu'il s'agit d'une variable.
- la *concaténation* (mise "bout à bout") de variables de type texte et de constantes "texte" est réalisée avec l'**opérateur +**
- document.write(...) affiche aussi bien les valeurs des variables que les constantes texte.
On peut séparer les éléments à afficher par l'**opérateur +** ou plus simplement par la **virgule ,**.

- o toutes les balises HTML peuvent figurer comme constantes texte, et sont exécutées comme du code HTML.
Ainsi, "
" inséré dans la valeur de la variable **phrase**, fera passer à la ligne suivante la suite du texte à afficher.

Comprendre le script et prévoir le résultat de son exécution.
Pour vérifier, appuyer sur ce bouton :

II. Afficher une image

```

EXEMPLE 2
<HTML>
<BODY> <CENTER>
<SCRIPT language = "JavaScript" >
document.write("<IMG SRC='../images/lycee2.jpg'> <BR> <H1>
                Une photo de mon lycée</H1><P>");

document.write("<IMG SRC='../images/monchat.gif'> <BR> <H1>
                et de mon chat .... </H1>");

</SCRIPT>
</CENTER>
</BODY>
</HTML>

```

Comprendre le script, en particulier le rôle du code HTML inséré et prévoir le résultat de son exécution.

Pour vérifier, appuyer sur ce bouton :

III. Afficher la date et l'heure

```

EXEMPLE 3
<HTML>
<BODY>
<SCRIPT language = "JavaScript" >
var date = new Date();
    // déclaration d'une variable de type Date
    // la variable date contient alors la date courante
var mois = date.getMonth() + 1;
    // la variable mois contient le N° du mois à partir de 0 (à 11)
var jour = date.getDate() ;
var annee = date.getYear();
if (navigator.appName == 'Netscape')
    annee = annee + 1900 ;
    // getYear() donne le numéro de l'année
    // pour netscape et Mozilla à partir de 1900
document.write("<Date> " , jour, " / " , mois, " / "+annee, "<BR>");
document.write("<Heure> " , date.getHours(), " : " ,date.getMinutes(),
                " : ",date.getSeconds() );

</SCRIPT>
</BODY>
</HTML>

```

A la lecture de ce code JS, comprendre :

- o La déclaration `var date = new Date();` déclare un objet appelé `date`, de **type Date**, initialisé à la date du système.

- Les fonctions `getDate()`, `getMonth()` etc... s'appliquent à la variable `date`. Ces fonctions ne peuvent s'adresser à ce type de variable `Date`.
On dit que ce sont des **méthodes** de l'**objet** `date`.
La notation générale est : **objet.méthode()**

Comprendre le script et prévoir le résultat de son exécution.

Pour vérifier :

IV. **Calculer la somme des 100 premiers naturels non nuls :**

```

EXEMPLE 4-1
<HTML>
<HEAD> <TITLE> Exemple 4</TITLE> </HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
var somme=0;
var nombre= 100;
for (i=1; i <=nombre ; i++)
  somme=somme + i ;
document.write("La somme des premiers naturels jusqu'à " + nombre +
  " est égale à " + somme+ "<BR>");
document.write( "Pour vérifier : (" +nombre+ " * "+(nombre+1)+ " )/2 = "+somme);
</SCRIPT>
</BODY>
</HTML>

```

A la lecture de ce code JS, comprendre :

- l'écriture générale d'une structure répétitive de type FOR
- `i++` signifie `i = i + 1`

Comprendre le script et prévoir le résultat de son exécution.

Pour vérifier :

```

EXEMPLE 4-2 (modification de l'exercice 4-1)
<SCRIPT LANGUAGE="JavaScript">
var somme=0;
var nombre= 100;
for (i=1; i <=nombre ; i++) {
  somme=somme + i ;
  document.write("Pour i = ", i, " ---> somme = ", somme , "<BR>");
}
</SCRIPT>

```

Ici remarquer les accolades `{ ... }` indispensables pour englober plusieurs instructions (séparées par des `;`) dans l'instruction répétitive FOR. Qu'obtient-on pour l'exercice 4-2, où (contrairement à l'ex 4-1) l'instruction d'affichage est incluse dans l'instruction FOR ?

Pour vérifier :

V. **L'événement OnClick sur un bouton de formulaire**

Considérons un composant de type **bouton**

Un clic sur le bouton provoque un événement **OnClick** auquel le programmeur peut rattacher du code JS (le plus souvent une fonction).

Dans l'exemple qui suit, la fonction **ALERT()** est appelée par le clic. Son rôle est d'afficher une boîte avec un message.

```

EXEMPLE 5
<HTML>
<HEAD> <TITLE> Exemple 5 du cours JS1</TITLE>
</HEAD>
<body>
<form>
<input type="button" value="Cliquez" onClick="alert('Coucou, c'est moi !')">
</form>
</body>
</HTML>

```

Testez :

VI. Boîte de dialogue et instruction conditionnelle

alert() affiche une boîte de message simple et attend la validation ou Echap.

reponse = prompt("texte", "chaîne par défaut") affiche "texte" et une ligne de saisie dans une boîte.

La poursuite du programme est bloquée tant que l'utilisateur n'a pas saisi une chaîne de caractères validée.

Prompt(..) a pour résultat la chaîne saisie ou sinon, la "chaîne par défaut" (si elle doit être vide, mettre ""). Cette chaîne est donc affectée ici à la variable nommée *reponse*

Le script utilise aussi la structure conditionnelle *SI .. ALORS .. SINON*

Exemple :

```

EXEMPLE 6
<HTML>
<HEAD> <TITLE> Exemple 6</TITLE> </HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
var prenom="" , bahut="" ;
prenom=prompt( "Votre prénom", "");
bahut=prompt( "Le nom de votre établissement", "");
if ( prenom !="" && bahut !="" )
    document.write("Bonjour ", prenom, " ! <BR> Vous plaisez-vous à ", bahut, " ?")
else
    alert("saisie incomplète !");
</SCRIPT>
</BODY>
</HTML>

```

Lire attentivement ce code JS. Voici quelques informations :

- Dans l'instruction conditionnelle, la condition qui suit **if** doit obligatoirement être mise entre parenthèses.
- principaux opérateurs de comparaison :
 - == (égalité, à ne pas confondre avec l'affectation =)
 - != (différent)
 - Autres : > , < , >= , <=
- principaux opérateurs logiques

- **&&** : et (utilisé ici pour composer 2 conditions simples).
- **||** : ou (le caractère | est obtenu au clavier avec *Alt Gr -*)
- **!** : non.

S'efforcer de décrire pas à pas la signification et le résultat des lignes de code.
Pour expérimenter son exécution :

VII. Fonctions et procédures

Les fonctions doivent être déclarées dans la section `<HEAD>... </HEAD>`, entre les balises `<SCRIPT> </SCRIPT>`, pour être comprises avant toute utilisation.

Elles sont appelées habituellement dans le corps de la page entre `<BODY>... </BODY>`

- soit à partir d'une section de code `<SCRIPT>... </SCRIPT>` soit affectées à une procédure d'événement liée à un composant d'un formulaire

Leur nom est introduit par le mot **function**. L'identificateur de la "function" est obligatoirement suivi de parenthèses. Entre ces parenthèses on écrit éventuellement la liste des paramètres (cf exemple VIII).

JS ne fait apparemment pas de distinction entre fonctions et procédures.

- Les **procédures** sans paramètre s'écrivent :

```
function nom-fonction () {
  séquence d'instructions
}
```

- Les **fonctions** fournissent un résultat :

```
function nom-fonction () {
  séquence d'instructions
  return paramètre-résultat
}
```

Pour approfondir leurs différences, voir le chapitre [Fonctions JS](#)
Voici un exemple simple de procédure sans paramètre :

```
EXEMPLE 7
<HTML>
<HEAD>
<TITLE> Exemple de fonction</TITLE>
<SCRIPT language="JavaScript">
  function bonjour() {
    document.write("Bonjour",<BR>);
  }
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT
  bonjour();
</SCRIPT>
<FORM>
<input type="button" value="Salut" onClick="alert('Et bon courage à tous ')">
```

```
</FORM>
</BODY>
</HTML>
```

A la lecture de ce code JS, noter et retenir :

- la place de l'appel de la fonction **bonjour()**
- elle effectue une **action**, afficher *Bonjour !*, **sans calculer une valeur résultat**. (l'absence d'instruction *return* caractérise une *procédure*).

Comprendre le script et prévoir le résultat de son exécution.

Pour tester :

VIII. Procédures et fonctions avec paramètre(s)

Les *procédures* et *fonctions* avec paramètre(s) sont déclarées conformément au paragraphe précédent.

Ils en diffèrent par la présence d'une liste de paramètres placés entre parenthèses.

Bien sûr, ces paramètres doivent recevoir des valeurs correctes qui leur sont affectées lors de l'appel de la fonction.

```
Function nom-fonction (liste de paramètres formels) {
  séquence d'instructions
  [return paramètre-résultat ]
}
```

Voici l'exemple simple d'une fonction à un paramètre :

```
EXEMPLE 8
<HTML>
<HEAD> <TITLE> Exemple 8</TITLE>
  <SCRIPT LANGUAGE="JavaScript">
    function somme_N_entiers (N) {
      var somme=0;
      for (i=1; i <=N ; i++) {
        somme=somme + i ;
        document.write("Pour i = ", i, " ---> somme = " , somme , "<BR>");
      }
      return somme
    }
  </SCRIPT>
</HEAD>

<BODY>
<SCRIPT>
var nombre= prompt("Somme jusqu'à N = ", 10);
document.write("Somme des 100 premiers entiers non nuls = " , somme_N_entiers(nombre));
</SCRIPT>
</BODY>
</HTML>
```

A la lecture de ce code JS, comprendre :

- l'appel de la fonction n'est pas ici provoqué par un événement du genre **onClick** (cf exemple V), mais par *l'affichage de son résultat demandé* par l'instruction **document.write**

- ce résultat est l'*image de la valeur* de **nombre** par la fonction **somme_N_entiers**, c'est-à-dire la valeur du paramètre **somme** qui est calculée par le code de la fonction.
- la fonction est définie avec un paramètre **formel** dont la valeur n'est connue qu'au moment de l'appel. Il y a alors transmission de la valeur du paramètre d'appel et affectation au paramètre formel.
- bien comprendre que dans ces conditions, la définition d'une fonction informatique est la description formelle d'un calcul ou d'une suite d'instructions.
- l'appel, c'est-à-dire l'exécution de la fonction avec la valeur du paramètre transmis, s'apparente au calcul de $f(x)$, l'*image de la valeur* x attribuée à la variable x , par une fonction mathématique f .

Comprendre le script et prévoir le résultat de son exécution.

Pour tester :

IX. Fonction appelée par un événement Utilisation de formulaire

```

EXEMPLE 9
<HTML>
<HEAD> <TITLE> Exemple 9</TITLE>
  <SCRIPT LANGUAGE="JavaScript">
    fonction somme_N_entiers (nb) {
      var somme=0;
      for (i=1; i <=nb ; i++) {
        somme=somme + i ;
        document.write("Pour i = ", i, " ---> somme = " , somme , "<BR>");
      }
      return somme;
    }
  </SCRIPT>
</HEAD>

<BODY>
<H3><CENTER>Somme des N premiers entiers non nuls</CENTER></H3>
<FORM name=formulaire>
<!-- attention, ce qui suit n'est pas une affectation ! -->
  Nombre = <INPUT type=text name=nombre value=10>
  <INPUT type=button name=bouton value="Calculer"
  onClick="formulaire.resultat.value=somme_N_entiers(formulaire.nombre.value)">
  Résultat = <INPUT type=text name=resultat value="">
</FORM>
</BODY>
</HTML>

```

- Dans ce dernier exemple, proche du précédent VIII, on reprend la fonction **somme_N_entiers(N)**, en enlevant l'affichage contenu dans l'itération.
- Cette fonction est simplement chargée de donner le résultat du calcul contenu dans la variable **somme**, grâce à l'instruction **return somme**
- Les nouveautés résident dans la façon de dialoguer avec le programme; pour cela on utilise des composants de formulaire.

- entrée de la valeur du paramètre **N** dans un composant de type champ de texte, nommé **nombre**, avec une valeur par défaut égale à 10. appel de la fonction provoqué par l'événement **onClick** sur le bouton nommé **bouton**.
- **somme_N_entiers()** utilise un paramètre dont la valeur est contenue dans le champ **nombre**
- cette fonction calcule un résultat appelé **somme** qui est affectée à la valeur d'un autre composant de type texte, nommé **resultat**, et qui est chargé de l'affichage.
- remarquer que tout cela est inclus dans du code HTML, et ne nécessite pas de section SCRIPT
- en revanche, comme on ne dispose pas de variables JS, on fait référence aux valeurs des contenus de ces composants par l'intermédiaire de leur propriété **value**. Il ne faut pas confondre la valeur et le nom du composant, donné par la propriété **name**.
- bien comprendre la façon de noter la référence à ces valeurs :
par exemple, **dialogue.nombre.value** signifie :
"la **valeur** du composant nommé **nombre** situé dans le formulaire nommé **dialogue**".

B - Les structures de données en JavaScript

Les données constantes

JS fournit les constantes suivantes, directement disponibles :

- les constantes numériques : entiers, "réels" écrits en notation décimale, ou flottante (c-à-d en notation scientifique, par ex : 2718E-3)
- les 2 constantes booléennes : *true* ou *false*
- les chaînes de caractères, entourées indifféremment de guillemets " " ou d'apostrophes ' ' (à privilégier).
- la constante spéciale **null** signifie "rien", "absence de valeur". Cette "valeur" est attribuée à toute variable utilisée sans être définie (par exemple `prompt()` retourne `null` si on sélectionne le bouton *Annuler*)

Manipulation des chaînes de caractères

- JavaScript facilite beaucoup l'affichage des résultats en convertissant automatiquement les entiers en chaînes de caractères, ce qui permet de concaténer des nombres avec des chaînes de caractères (transtypage automatique).
- Dans l'instruction d'écriture dans le document courant, `document.write()`, les données à afficher peuvent être séparées par des virgules ou des `+`. (la concaténation par l'opérateur `+` est recommandée)
- Des caractères spéciaux peuvent aussi être insérés dans les chaînes : `\b` (retour arrière), `\f` (saut de page), `\n` (nouvelle ligne), `\r` (Entrée), `&` (tabulation); `\'` pour une apostrophe
- On peut insérer des codes HTML sous forme de chaînes, qui seront bien interprétées à l'exécution comme de véritables balise, et non par affichées telles quelles.

Exemple

```
<script>
document.write("Voici la valeur approchée à ", 1E-3,
    " près de la constante e : ", 2718E-3);
document.write("<P>");
document.write("Mon <i>lycée</i> est situé en
<u>Seine-Saint-Denis</u> dans la ville d'Epina-sur-Seine");
</script>
```

Voici la valeur approchée à 0.001 près de la constante e : 2.718
 Mon *lycée* est situé en Seine-Saint-Denis dans la ville d'Epina-sur-Seine

(Attention ! le symbole "exposant" E doit être collé au dernier chiffre.)

Déclaration et affectation de variables• Types de variables

On distingue 5 types de variables en JS.

Les nombres : number, les chaînes : string, les booléens : boolean, les objets : object et les fonctions : function

La fonction **typeof()** appliquée à une variable retourne son type.

```
var chaine = 'bonjour';
document.write(typeof(chaine));
```

En voici l'exécution :

string

• Notation : **var nom = valeur**

Déclaration de la variable **nom** initialisée avec la valeur **valeur**; son type est alors celui de la valeur.

- Le mot **var** est facultatif, mais recommandé.
- Les variables doivent commencer par une lettre ou par _
- JS distingue minuscules et majuscules.
Les variables suivantes *mavariab*le, *maVariable*, *Mavariab*le, *MaVariable* sont toutes distinctes.
- Le symbole = est réservé à l'affectation; le symbole de comparaison (égalité) se note == (2 symboles = collés).
- JS attribue *automatiquement* à la variable le type de la valeur affectée.
Sinon JS lui attribue une valeur et un type indéterminés "**undefined**"

```
var x ;
document.write("Voici la valeur de x : x = ' + x+ "<BR>");
document.write("et son type : typeof(x) = " + typeof(x) );
```

En voici l'exécution :

Voici la valeur de x : x = undefined

et son type : `typeof(x) = undefined`

- Attention

L'attribution implicite de type, à première vue pratique, risque d'engendrer confusion et absence de rigueur.

Il faut s'habituer à **déclarer** les variables et à les **initialiser** (c'est-à-dire leur donner une valeur initiale au moment de leur déclaration, ce qui fixe leur type).

Par exemple

```
nomLycee = "J. Feyder" ; TauxTva = 20.6 ;
```

Eviter absolument ce qui suit (c'est compris par JS, mais peu recommandable !)

```
var x = 12.5;
document.write('Ici x est une variable "réelle" : x = '+ x+ "<BR>");
x = "Bonjour !" ;
document.write("Maintenant x est une chaîne de caractères : x = " + x );
```

En voici l'exécution :

```
Ici x est une variable "réelle" : x = 12.5
Maintenant x est une chaîne de caractères : x = Bonjour !
```

Affectation de variables

- Mécanisme de l'affectation

variable = valeur (de l'expression, de même type)

Exemple à étudier :

```
var a=10 ; b= 15;
document.write( " a= "+a + " ; b= "+b+"<BR>");
a = 2 * b - 5 ; // valeur de a = 2 fois la valeur de b - 5
document.write( " a= "+a + " ; b= "+b+"<BR>");
b = a + b ; // nouvelle valeur de b = précédente valeur de b + valeur de a
document.write( " a= "+a + " ; b= "+b);
```

En voici l'exécution :

```
a= 10 ; b= 15
a= 25 ; b= 15
a= 25 ; b= 40
```

- Portée des variables

- Portée d'une variable

La portée d'une variable est le domaine où cette variable est connue et utilisable.

De façon générale les variables définies directement dans une séquence de script (entre `<script> ...</script>`) ont une **portée globale** sur toutes les parties de script.

Exemple :

```
<HEAD>
<script>
var lycee="J-Feyder";
var ville="Epinay-sur-Seine";
document.write("Mon lycée est situé dans la ville d\'"+ville+"<BR>");
var date = new Date();
</script>
</HEAD>
</BODY>
```

```
<script>
document.write("Il porte le nom du cinéaste "+lycee+"<BR>");
document.write("Nous sommes au mois N°",date.getMonth() + 1,"<BR>");
</script>
</BODY>
```

En voici l'exécution :

```
Mon lycée est situé dans la ville d'Epinay-sur-Seine
Il porte le nom du cinéaste J-Feyder
Nous sommes au mois N°11
```

Construction des expressions

On peut distinguer plusieurs types d'expressions :

- expressions **arithmétiques** : construites par opérations sur les entiers et les réels.
Principales opérations

- les 4 opérations usuelles : + , - , * , /)
- ++ (incrément) , -- (décrément)
- % (modulo ou reste par une division entière)

- expressions **d'affectation** (ou attribution)

- l'opérateur d'affectation : *variable = expression*
- autres opérateurs d'attribution : += , -= , *= , /= , %= , ++ , --
- x += 4 équivaut à x = x + 4
x ++ équivaut à x = x + 1

- expressions **chaînes de caractères**

- + opérateur de concaténation (mise bout à bout) de deux chaînes
- var aujourd'hui = " Lundi " + 3 + " novembre" + 1997;
- document.write(aujourd'hui, "
");
- += ajoute la chaîne de droite à la chaîne de gauche

```
var message = "Bonjour ";
message += "tout le monde !" ;
document.write(message);
```

- En voici l'exécution :

```
Bonjour tout le monde !
```

- expressions **booléennes** ou logiques

- opérateurs de comparaison : == (égal, même valeur) , != (différent) , > , >= , < , <=
- opérateurs logiques : && (ET) , || (OU) , ! (NON) , utilisés surtout dans les instructions conditionnelles.

- Les expressions conditionnelles
(condition) ? val1 : val2 : évalue la condition et exécute val1 si vrai, ou val2 si faux.
Exemple : message = (fin = true) ? "bonjour" : "au revoir"

Tableaux JS

Un tableau dans un langage de programmation n'a rien à voir avec un tableau HTML ou Word !

C'est un ensemble de données, en général de même type (chaîne de caractères, nombres ..), rangées sous un même nom et distingués par un numéro.

Ce numéro, l'indice, est placé entre [], et caractérise un élément du tableau.

Déclaration

var nom_tableau = **new Array**(dimension) ;

Le mot **new** commande la construction d'un objet de type **Array**, c'est-à-dire, tableau.

Le paramètre *dimension*, s'il est présent, est le nombre d'éléments.

Exemples

- `var MonTableau = new Array(8)`
construit un tableau nommé **MonTableau**, de taille 8 éléments.
- `var Les4saisons = new Array("printemps", "été", "automne", "hiver");`
construit un tableau en initialisant 4 éléments, c'est-à-dire en leur affectant une valeur initiale (qui pourra ensuite être modifiée).
- `var mois= new Array(12);`

Utilisation

Les éléments d'un tableau de taille *dim*, sont indicés à partir de 0 jusqu'à *dim* - 1.

Exemples à étudier :

```
var MonTableau = new Array(8);
Ces 8 éléments sont nommés MonTableau[0], ..., MonTableau[7]
var mois= new Array(12); mois[0]="Janvier"; ...mois[11]="Décembre";
```

```
var Les4saisons = new Array("printemps", "été", "automne", "hiver");
document.write("Voici les 4 saisons : <UL>")
for (i=0 ; i<4 ; i++) {
  document.write("<LI>", Les4saisons[i] )
}
document.write("</UL>");
```

En voici l'exécution :

```
Voici les 4 saisons :
• printemps
• été
• automne
• hiver
```


Tableaux à plusieurs dimensions

Il faut les gérer comme des tableaux de tableaux (à une dimension).

Exemple et exécution :

```
tab=new Array(3);
tab[0]= new Array(1,2,3);
tab[1]= new Array(4,5,6);
tab[2]= new Array(7,8,9);
for (i=0;i<3;i++) {
  for (j=0;j<3;j++)
    document.write(tab[i][j]," ");
  document.write("<br> ");
}
```

```
1 2 3
4 5 6
7 8 9
```

Propriétés

- **length** donne le nombre d'éléments.

```
var mois= new Array(12);
document.write("Il y a ", mois.length, "mois dans l'année");
var NbMois = mois.length ; document.write(" partagés en ", NbMois / 3, " trimestres");
```

Méthodes

- **reverse()** change l'ordre des éléments
- **sort()** trie suivant l'ordre croissant, ou suivant le modèle indiqué en paramètre

Particularités et bizarreries ..

- **var tab = new Array()** crée un objet vide, de type tableau, sans fixer sa dimension
- **var tab= new Array("bonjour", charge_e, 1.6E-19, "C")** est accepté et crée un tableau hétéroclite de 4 éléments

```
var unLepton="un électron";
var tab= new Array(unLepton, " porte une charge négative égale à ", 1.6E-19," C");
for( i=0; i<tab.length; i++) document.write(tab[i] );
```

- En voici l'exécution :

```
un électron porte une charge négative égale à 1.6e-19 C
```

- On peut aussi créer un tableau directement "en extension", sans faire appel au constructeur Array(). On liste les valeurs des éléments dans [...].

Exemple : les 2 définitions suivantes sont équivalentes

```
var Les4saisons = new Array("printemps", "été", "automne", "hiver");
var Les4saisonsbis = ["printemps", "été", "automne", "hiver"];
document.write("Voici les 4 saisons : <br>")
for (i=0 ; i<Les4saisonsbis.length ; i++)
  document.write(Les4saisonsbis[i], " ")
```

- En voici l'exécution :

```
Voici les 4 saisons :
```

printemps été automne hiver

C - Les structures de contrôle en JavaScript

Structure algorithmique d'un programme

Un programme informatique est assemblé à partir de 3 catégories principales d'instructions, quel que soit le langage utilisé pour le coder :

- instructions séquentielles
- instructions conditionnelles (ou alternatives)
- instructions itératives (ou répétitives)

Nous allons voir en parallèle les notations algorithmiques et leur traduction en JS.

La séquence d'instructions

Il s'agit d'une suite d'actions qui sont exécutées dans l'ordre, les unes après les autres sans choix possibles, ni répétitions.

Séquence ou bloc d'instructions	
algorithme	Code JavaScript
début Instruction 1 Instruction 2 fin	<pre>{ Instruction 1; Instruction 2; }</pre>

L'instruction conditionnelle if ... [then] .. else

Syntaxe

Structure conditionnelle	
algorithme	Code JavaScript
SI condition ALORS Séquence 1 SINON Séquence 2 FINSI	<pre>if (condition) { séquence 1 } else { séquence 2 }</pre>

Remarques

1. La condition doit être **toujours entourée de ()**
2. Le mot *alors* (**then**) est toujours sous-entendue.
3. La séquence *alors* est exécutée si la condition est vraie.
4. La séquence *else* (optionnelle) est exécutée si la condition est fausse.
5. les { } ne sont pas obligatoires qu'en cas d'instructions multiples.

Exemple 1-1

```
<SCRIPT language = "JavaScript" >
var prixHT =150 ; prixTTC=0;
if ( prixHT == 0 )
  alert("donnez un prix HT !");
else
  prixTTC = prixHT * 1.206;
```

```
document.write("Prix HT = " + prixHT + "<BR>");
document.write("Prix TTC = ", prixTTC, "<BR>");
</SCRIPT>
```

Qu'obtient-on exactement à l'exécution ?

Conditionnelles imbriquées; exemple 1-2

Les instructions conditionnelles peuvent être imbriquées : cela signifie que dans la structure conditionnelle, **séquence 1** et/ou **séquence 2**, peuvent elles-mêmes contenir une structure conditionnelle.

```
<SCRIPT language = "JavaScript" >
var age=0;
age=prompt("Donnez votre âge : ","");
if ( age <= 0 )
alert("Cela n'a pas de sens !");
else
if (age <=13)
alert("Vous êtes encore bien trop jeune ...")
else
if (age <18)
alert("Désolé, vous êtes encore mineur(e)")
else
if (age <25)
alert("Vous êtes déjà majeur(e) !")
else alert("Ne vous vieillissez donc pas !");
</SCRIPT>
```

Qu'obtient-on exactement à l'exécution ?

L'itération contrôlée FOR

L'instruction for permet de répéter une séquence d'instructions tant qu'une condition est vraie; syntaxe

Structure itérative FOR	
algorithme	Code JavaScript
POUR I de <i>valeur initiale</i> à <i>valeur finale</i> Répéter Séquence d'instructions FIN POUR	for (<i>valeur initiale</i> ; <i>condition</i> ; <i>poursuite</i>) { séquence d'instructions }

La condition qui suit **FOR** est composée de 3 éléments :

1. une valeur ou expression initiale portant sur une variable entière appelée compteur.
2. une condition : tant qu'elle est vraie, la répétition est poursuivie.
3. une expression de poursuite qui consiste en la mise à jour du compteur.

Exemple 2

```
<SCRIPT language = "JavaScript" >
document.write("Table des carrés<BR>");
for (var i = 0; i <15; i++) {
```

```
document.write("i = "+i+"      i² = "+i*i+"<BR>");
}
</SCRIPT>
```

Qu'obtient-on exactement à l'exécution ?

Exemple 3

```
<SCRIPT language = "JavaScript" >
function hasard(N) {
// renvoie une valeur entière au hasard entre 1 et N inclus
return Math.floor(Math.random()*N)+1;
}
document.write("Tableau de 100 nombres au hasard<BR>");
max= prompt("Nombres au hasard de 1 à ", "100");
tab = new Array(100);
for (var i = 0; i <100; i++) {
    tab[i]= hasard(max);
}
for (var i = 0; i <100; i++) {
    document.write("tab [, i,] = ", tab[i], " ");
}
</SCRIPT>
```

Qu'obtient-on exactement à l'exécution ?

- Les accolades sont obligatoires s'il y a plusieurs instructions.
- Chaque instruction se termine par un point-virgule ;
- La séquence sera répétée tant que la condition est vraie, compte-tenu de la mise à jour du compteur.
- Normalement la mise à jour du compteur doit "rapprocher" de l'arrêt de l'itération.
- Sinon, les conditions mal conçues peuvent entraîner des "boucles infinies", comme par exemple :

```
for (i= 11 ; i>10 ; i ++ ) { .... }
```

Variante

```
for (variable in objet ) {
    séquence instructions
}
```

Exemple 4

```
<SCRIPT language = "JavaScript" >
function hasard(N) {
return Math.floor(Math.random()*N)+1;
}
document.write("Tableau de 100 nombres au hasard<BR>");
max= prompt("Nombres au hasard de 1 à ", "100");
tab = new Array(1000);
for (var i = 0; i <100; i++) {
    tab[i]= hasard(max);
}
for (var i in tab ) {
    document.write("tab [, i,] = ", tab[i], " ");
}
}
```

<SCRIPT>

Qu'obtient-on exactement à l'exécution ?

Les instruction break et continue

L'instruction break permet de sortir de l'itération ou de la séquence en cours.

L'instruction continue sort de la séquence en cours puis reprend à l'itération suivante.

L'itération WHILE (TANT QUE)

L'instruction répétitive while permet de répéter une séquence d'instructions tant qu'une expression est vraie.

syntaxe

Structure itérative WHILE	
algorithme	Code JavaScript
TANT QUE (<i>condition est vraie</i>) REPETER Séquence d'instructions FIN TANT QUE	WHILE (<i>condition</i>) { séquence d'instructions }

Exemple 5

```
<SCRIPT language = "JavaScript" >
function hasard(N) {
return Math.floor(Math.random()*N)+1;
}
max= prompt("Nombres au hasard de 1 à ", "6");
document.write("<H2>Tableau de nombres entre 1 et ", max,
" tirés au hasard, jusqu'à obtenir ", max, "</H2>");
tab = new Array(10*max);
a = hasard(max);
tab[0] = a;
i = 0 ;
while ( a != max ) {
a = hasard(max);
tab[i]= a;
i ++
}
document.write(max, ' a été obtenu au ', i, 'ème tirage <P>');
i=0;
while ( tab[i] != null ) {
document.write("tab [", i, "] = ", tab[i], "---");
if ( i %5 ==0 && i!=0) document.write("<br>");
i ++;
};
</SCRIPT>
```

Qu'obtient-on exactement à l'exécution ?

D - Les classes prédéfinies de JavaScript

Généralités sur les objets en JS

- JS n'est pas un vrai langage orienté-objet. Des concepts primordiaux ne sont pas implémentés, notamment l'héritage. Mais il peut constituer une introduction à la syntaxe objet. Dans le présent chapitre, nous prendrons connaissance de quelques classes de données prédéfinies.

Ensuite, nous étudions les objets du navigateur, puis enfin nous apprendrons comment définir nos propres classes d'objets en JavaScript.

- **Une classe** d'objets est un ensemble d'informations regroupés sous un même nom, qui décrivent la structure et le comportement commun de ces objets.
Pour définir une classe, il faut préciser :
 1. les propriétés (ou attributs) qui décrivent ses caractéristiques
Une **propriété** est une variable qui est attachée à un type d'objets, et qui est contenue de ses caractéristiques.
 2. les méthodes (procédures ou fonctions) qui décrivent ses comportements et ses actions
Une **méthode** est une fonction qui ne s'applique qu'à une classe d'objet.
- Construction des objets
Pour obtenir un objet, appelé aussi instance de la classe, on utilise une fonction spéciale appelée **constructeur** et qui porte le nom de la classe.
Un objet est alors créé par l'instruction :
var objet = new Classe();
- Utilisation des objets
Propriétés et méthodes constitutives d'une classe ne sont alors applicables qu'à un objet de cette classe.
objet.propriété
objet.méthode()

Classes d'objets prédéfinis

Elles sont définies dans JS, accompagnées de données (*propriétés*) et de fonctions (*méthodes*) utilisables directement par le programmeur.

Nous allons parcourir ces 4 classes d'objets : **Math, String, Date et Image**, afin de savoir construire des objets de ces classes et utiliser leurs propriétés et leurs méthodes.

Math

Les fonctions mathématiques usuelles doivent être préfixées par le nom de l'objet **Math**, desquelles elles dépendent. Ce sont les "**méthodes**" de calcul de l'objet **Math**.

Par exemple :

- **Math.PI** désigne une propriété de l'objet Math : le nombre PI
- **Math.sin(1.50)** appelle une méthode de l'objet Math et calcule $\sin(1.50)$, l'angle étant exprimé en radians.

Pour alléger les notations, on peut "*factoriser*" le préfixe **Math** dans une séquence de calcul mathématique, comme dans l'exemple suivant :

```
<script>
r=10;
with (Math ) {
s = PI * pow(r , 2);
theta = PI / 3;
x = r * cos( theta );
y = r * sin( theta);
document.write("PI = ",PI, "<br>");
document.write("s = ", s, "<br> Coordonnées de M : x = ",x, "      y = ",y)
}
</script>
```

En voici l'exécution :

```
PI = 3.141592653589793
s = 314.1592653589793
Coordonnées de M : x = 5.000000000000001 y = 8.660254037844385
```

Liste des principales méthodes

- Math.sqrt() , racine carrée.
- Math.log() , Math.exp() , Math.abs() ,Math.cos () , Math.sin() , Math.tan()
- Math.floor(), Math.ceil() entier immédiatement inférieur / supérieur.
- Math.pow(base, exposant), fct puissance, où base et exposant sont des expressions numériques quelconques évaluables.
- Math.max() , Math.min()
- Math.random(), nombre "réel" choisi au hasard dans [0 , 1[
- Math.round()arrondi à l'entier le plus proche.

Conversion chaîne <--> nombre (entier ou flottant)

- Fonctions parseInt(), parseFloat() et eval() récupère les chaînes de caractères passée en paramètre et s'efforce de les évaluer numériquement : voir le paragraphe suivant **String** spéciale **isNaN()** évalue l'argument pour déterminer s'il s'agit d'un nombre ("NaN" = **Not** a Number). Elle retourne alors TRUE ou FALSE.
Exemple d'utilisation pour contrôler la saisie d'un entier :
- <SCRIPT LANGUAGE="JavaScript1.1">
- function testnum(chnum) {
- num=parseInt(chnum);
- if (isNaN(num))
- alert(chnum+ '\n'est pas pas un entier !');
- }
- </SCRIPT>
- Exercices : Faire afficher un [tableau de valeurs](#) pour la fonction ln(x) dans [-10, 10]

String

Déclaration

```
var chaine = "<B>Bonjour !</B>";
document.write ("La longueur de la chaine ",chaine, " est : ",
chaine.length,". Pourquoi ?<br>");
```

La longueur de la chaîne **Bonjour !** est : 16. Pourquoi ?

var chaine = "Bonjour !" crée une variable nommée chaine et lui attribue :

19. le type String.
20. la valeur "Bonjour !" avec l'attribut **gras**

Propriétés

- La valeur peut donc être composée d'une suite de caractères quelconques, y compris des balises HTML.
- Des caractères spéciaux peuvent aussi être insérés dans les chaînes : \b (retour arrière), \f (saut de page) , \n (nouvelle ligne) , \r (Entrée), \t (tabulation); \' pour une apostrophe

- L'unique propriété **length** permet d'obtenir la longueur de la chaîne.
- L'opération + concatène (mise à la suite) 2 chaînes pour en former une nouvelle

```
var info = "\"informatique\" ";
var monOption = "l'option " + info ;
document.write("J'enseigne encore " + monOption) ;
```

J'enseigne encore l'option "informatique"

Quelques fonctions (ou méthodes)

- **eval()** évalue numériquement une expression arithmétique fournie sous forme de chaîne de caractères.
- **parseInt()**, donne un nombre entier résultant de la conversion (si possible) d'une chaîne de caractères.
Si la conversion n'est pas possible, la valeur renvoyée est 0.
- **parseFloat()**, donne un nombre décimal de la même façon.
- La fonction **toString(base)** convertit l'objet (nombre généralement) en une chaîne représentant le nombre écrit dans la base indiquée.
- Exemple à étudier :

```
var a = 5+ "007";
var b = 5+ parseInt("007");
var c = 2+ parseFloat("1.1416");
var d=255;
document.write(" a = ", a,"<br> b = ", b,"<br> c = ", c);
for (var i=0; i<255 ; i++)
  document.write("En décimal i = "+i+ " s'écrit "+ i.toString(16)+")
```

a = 5007
b = 12
c = 3.1416
En décimal d= 255 s'écrit ff

- Exercice d'entraînement au [calcul mental](#)
- **chaine.toUpperCase()**, pour mettre chaîne en majuscule.
Exemple :

```
var chaine = "Bonjour !";
chaine = chaine.toUpperCase();
```

- **chaine.substring(d, l)** extrait une partie de chaîne, à partir du caractère de position d+1, jusqu'à l.
Exemple :

```
var chaine = "aujourd'hui";
document.write(" chaine.substring( 2,6) = ", chaine.substring( 2,6));
```

chaine.substring(2,6) = jour

- **chaine.charAt(n)** donne le caractère placé en nième position (n de 0 à chaine.length-1).

Exemple :

```
var chaine = "informatique";
document.write("J'épelle :<br>");
for (i=0 ; i< chaine.length ; i++)
document.write (chaine.charAt(i), " - ");
```

```
J'épelle :
i - n - f - o - r - m - a - t - i - q - u - e -
```

- **chaine.indexOf(s_ch)** donne la 1ère position du caractère de chaine égal au 1er caractère de s_ch.

Exemple :

```
var chaine = "informatique";
var s_ch = "ma";
var car = "i";
var position = 2;
document.write ("1ère position de ", s_ch, " dans ", chaine," est : ",
chaine.indexOf( s_ch),"<br>");
document.write ("position de ", car ," dans ", chaine," à partir de la
position ", position," est : ", chaine.indexOf(car, position)"<br>");
```

```
1ère position de ma dans informatique est : 5
position de i dans informatique à partir de la position 2 est : 8
```

- La méthode **chaine.split(séparateur)**
Appliquée à un texte, elle fournit un tableau de chaînes dont les éléments sont les sous-chaînes extraites suivant le séparateur.

Exemples

Chaîne initiale : jules.toto@mail.ac-creteil.fr
découpage suivant "@" :
tab_ch1[0] = jules.toto
tab_ch1[1] = mail.ac-creteil.fr
découpage suivant "." :
tab_ch1[0] = jules.toto
tab_ch1[1] = mail.ac-creteil.fr

Date

L'objet Date permet de définir et gérer les dates et les heures.

L'origine des dates a été choisie le 1er janvier 1900 et est exprimée en millisecondes.

Construction d'un objet de type Date

Pour construire un objet de type Date, il faut utiliser un **constructeur Date()** avec le mot-clé **new**

```
variable = new Date(liste de paramètres)
```

Attention, les secondes et les minutes sont notées de 0 à 59, les jours de la semaine de 0 (dimanche) à 6, les jours du mois de 1 à 31, les mois de 0 (janvier) à 11, et les années sont décomptées depuis 1900 .

On peut passer différents paramètres pour construire divers objets date

- Date() , pour obtenir la date et l'heure courante (connue du système)
- Date(month day, year hour:min:sec) pour obtenir par exemple : *December 25, 1995 13:30:00*
- Date(année, mois, jour), une suite convenable de 3 entiers, par exemple (2000, 0, 1)
- une suite de 6 entiers (année, mois, jour, heures, minutes , secondes), (même exemple : 95, 11, 25 , 13, 30 , 00

Méthodes

Elles permettent d'extraire diverses informations d'un objet date

- **set....()** : pour transformer des entiers en Date
- **get....()** : pour transformer en date et heure des objets Date
- **to...()** : pour retourner une chaîne de caractères correspondant à l'objet Date
- après les préfixes **set** et **get** , on peut mettre Year, Month, Date , Hours, Minutes, Seconds pour obtenir respectivement : nombre d'années depuis 1900, le numéro du mois, le N° du jour dans le mois, et les heures, minutes et secondes.
- **getDay()** donne le N° du jour de la semaine (le 0 tombe le dimanche)
- **getTime()** donne le nombre de millisecondes écoulées depuis le 1/1/1970, très pratique pour calculer des intervalles entre 2 dates.

Exemple

```
var aujourd'hui= new Date();
var maDate = new Date ("November 24, 1981");
var jour = maDate.getDate () // jour vaut 24.
document.write("Nous étions le ", jour, "/",
    maDate.getMonth()+1, "/", maDate.getYear()+1900 ,<br>);
document.write("Nous sommes le ", aujourd'hui.getDate(), "/",
    aujourd'hui.getMonth()+1, "/", aujourd'hui.getYear()+1900 );
```

```
Nous étions le 24/11/1981
Nous sommes le 8/11/3906
```

E - Procédures et fonctions

Déclaration et appel des fonctions en JS

- On distingue traditionnellement les procédures et les fonctions. JavaScript ne différencie pas leur syntaxe. Il est recommandé de les inclure dans la section d'en-tête du document à l'intérieur du couple de balises
....
- Une **procédure** est une suite d'instructions qui forment un tout et qui sont regroupées sous un même nom.
Une **fonction** est une suite d'instructions qui calcule un résultat; celui-ci est transmis à l'expression qui a appelé la fonction, après le mot **return**. A noter que l'instruction **return** peut être utilisée plusieurs fois en cas de valeur retournée conditionnellement.
- De plus, procédures et fonctions peuvent admettre des **paramètres**.
Ce sont des variables dont les valeurs sont fixées par le programme appelant au moment de

l'exécution et qui apparaissent "formellement", sans valeur affectée au niveau de la déclaration.

S'il n'y a pas besoin de paramètres, le nom de la fonction est suivi d'un couple de parenthèses vides.

Déclaration générale d'une procédure et d'une fonction

```
<HEAD>
<SCRIPT LANGUAGE=JavaScript >
Function nomProcédure(param1, param2, ...) {
  séquence d'instructions;
}
Function nomFonction(param1, param2, ...) {
  séquence d'instructions;
  return nom_variable
}
</SCRIPT>
</HEAD>
```

Appel d'une procédure et d'une fonction

- JS lit les fonctions présentes dans une page, lors de son ouverture, **mais ne les exécute pas**.
- **Une fonction n'est exécutée qu'au moment de son appel.**
- Dans l'écriture de l'appel de la fonction, il faut fournir une liste de valeurs correspondant exactement à la liste des paramètres présents dans la déclaration.
- Les procédures forment des instructions à part entière, tandis que les fonctions doivent être affectées à une variable du type de retour ou incluses dans des expressions comme document.write(...).

```
nomProcédure(valeur1, valeur2, ...);
variable = nomFonction(valeur1, valeur2, ...);
```

Exemple

Etudier l'exemple suivant et prévoir exactement son exécution.

```
<HEAD>
<SCRIPT>
// déclaration de la procédure
function bonjour(prenom) {
  document.write("Bonjour, comment vas-tu ", prenom, " ?<br>");
}
// déclaration de fonctions
function volumeSphere(rayon) {
  return 4/3*Math.PI*Math.pow(rayon,3);
}
function calculerPrix(PrixUnitaire, NbArticles) {
  return PrixUnitaire* NbArticles;
}
</SCRIPT>
</HEAD>
<BODY>
// appel de la procédure
bonjour("Toto");
//appels des fonctions
```

```
var montant=CalculPrix( 150 , 4 );

document.write( "Tu dois payer ", calculerPrix( 150, 4), "F.<BR>");
document.write( "Sais-tu que le volume de la sphère de rayon unité est ",
    volumeSphere(1)," ?<BR>" );
</BODY>
```

Visibilité des paramètres

- De façon générale, les paramètres formels d'une fonction ne sont connus qu'à l'intérieur de la fonction.
De même, les **variables locales**, variables qui sont explicitement déclarées à l'intérieur de la fonction.
- Conséquences :
Si la valeur d'un paramètre ou d'une variable locale est modifiée **dans** la fonction, cette modification ne sera pas connue à l'extérieur de la fonction. On dit que cette variable n'est pas visible au niveau du programme général
- Examiner l'exemple suivant : qu'obtient-on exactement à l'exécution ?

```
<HEAD>
<SCRIPT LANGUAGE=JavaScript >
function bonjour(nom) { // nom est un paramètre local
    var ch ="Salut !"; // ch est une variable locale
    document.write("au début de la fonction : Bonjour "+nom +"<BR>");
    nom = "Alain"; // on modifie le paramètre local
    document.write("à la fin de la fonction : Bonjour "+nom +"<BR>");
}

var prenom = "Jacques";
bonjour(prenom) ;
document.write("après appel de la fonction : Bonjour "+prenom +"<BR>");
</SCRIPT>
</HEAD>
```

- On ajoute à la fin l'instruction suivante :

```
document.write("après appel de la fonction : "+ ch + " bonjour "+nom +"<BR>");
```

- Que va t-il se passer exactement ?
- Et si on supprime + ch , qu'obtient-on ?

Fonctions récursives

JavaScript est un langage récursif !

Exemple classique : la **fonction factorielle**

```
<HTML>
<BODY>
<script>
function fact(n) {
if (n==0) return 1
else return (n*fact(n-1))
}
```

```

nb=prompt("N= ", "0");
document.write("Liste des premières factorielles jusqu'à ", nb);
for (var i=0;i<nb;i++)
document.write(i+ " ! = "+fact(i)+"<br>")
</script>
</BODY>
</HTML>

```

Qu'obtient-on exactement à l'exécution ?

L'objet Function

- A chaque fonction rencontrée, JavaScript construit un tableau de ces arguments, appelés **arguments**.

Si la fonction s'appelle calculer, ce tableau a :

- pour éléments **calculer.arguments[i]**
- pour taille **calculer.arguments.length**

- Exemple

- fonction calculerPrix(PrixUnitaire, NbArticles) {
- if (calculerPrix.arguments.length !=2)
- alert("impossible de calculer le prix !")
- else
- return PrixUnitaire* NbArticles;
- }

Dans le cas où l'on ne passe pas 2 paramètres effectifs lors de son appel, cette fonction affiche un message d'alerte et renvoie la valeur undefined

On peut définir aussi une fonction dynamiquement, en cours d'exécution, et avec un nombre d'arguments non précisés au départ !

- Ce nombre d'arguments variables (et non défini à la déclaration), trouve son utilité dans le cas où ces arguments doivent être traités comme les éléments d'un tableau

```

function somme() {
var resultat=0 ;
for (var i=0; i < somme.arguments.length; i++)
resultat += somme.arguments[i] ;
return resultat ;
}
// quelques appels
document.write("1+2+3 = somme(1,2,3) = " + somme(1,2,3)+"<p>");
document.write("1+2+3+ .. +10 = somme(1,2,3,4,5,6,7,8,9,10) = "
+ somme(1,2,3,4,5,6,7,8,9,10)+"<p>");
document.write(" somme(15.4) = " + somme(15.4)+"<p>");
document.write(" somme() = " + somme()+"<p>");

```

Qu'obtient-on exactement à l'exécution ?

- Création d'instance de la classe Function

Il existe un constructeur Function(), semblable aux constructeurs Date(), Image(), permettant de définir une nouvelle fonction, lors de l'exécution.

Syntaxe :

- **nom_fct=Function("arg1", .. "argN", "code_fct")**, où :
- arg1, .. argN est la liste des paramètres de la fonction
- code_fct est le code, donc la suite des instructions, séparées par des ;

Exemple :

Les 2 fonctions f1 et f2 sont équivalentes :

```
<script>
function f1(x) {
return x*x
}
var f2 = new Function("x","return x*x")
document.write("f1(12) =" + f1(12) + "<BR>")
document.write("f2(12) =" + f2(12) + "<BR>")
</script>
```

F - Programmation événementielle en JS

L'utilisateur déclenche un "événement" (clic, déplacement souris, clic sur un bouton, choix d'un option de liste déroulante etc ...) relativement à un objet (lien, composant de formulaire ..).

L'événement est décelé (capté) par l'objet cible si celui-ci possède une "sensibilité" à l'événement. Il faut donc connaître la correspondance objet-événement.

S'il prévoit un intérêt à "répondre" à cet événement, le programmeur doit à l'avance, associer du code JS ou une fonction JS à un tel couple objet-événement. L'appel et l'exécution de ce code ou de cette fonction seront automatiquement déclenchés par l'événement, et constituent ainsi la "réponse" à celui-ci.

Les fonctions sont déclarées dans la partie <HEAD> et les appels en général associés à la balise de l'objet HTML qui va capter l'événement. Il faut veiller à bien gérer le passage de paramètres, souvent un formulaire entier.

Evénements JS

Nous avons déjà vu des exemples d'appels de fonctions JavaScript provoquées par des événements qui surviennent **au moment de l'exécution du programme.**

Ces événements sont des actions qui sont déclenchées le plus souvent par l'utilisateur.

Par exemple, un clic sur un bouton (composant de formulaire) est un événement, comme l'est la validation d'un texte saisi dans une ligne de texte, ou le choix d'une option dans un composant case à cocher

Le navigateur reconnaît un ensemble d'événements associés à des balises, des liens et des composants de formulaires. Par programmation, on peut leur associer des fonctions particulières appelées **gestionnaires d'événements** appelée systématiquement lorsque ces événements sont provoqués.

Un gestionnaire d'événement est une procédure particulière attachée à une balise HTML,

1. prédéfinie par le langage JS (par exemple onClick)
2. déclenchée par l'événement correspondant (click sur un composant de type button)
3. qui apparaît dans la balise du composant qui reçoit l'événement

4. et à laquelle on affecte une fonction écrite en JS, déclarée préalablement

définition générale

1. **onEvent** est le nom du gestionnaire d'événements associé à l'événement *Event*, comme **onClick**
2. **Balise** est un nom de balise qui sait gérer un tel événement.
3. "**code JS**" est généralement une fonction déclarée auparavant dans une section `<HEAD><SCRIPT>...</SCRIPT> </HEAD>`
4. Mais ce peut être aussi une suite d'instructions JS, séparées par des virgules.

Supposons déjà déclarée une fonction nommée *calculer*. On peut appeler le navigateur à l'exécuter au moment où l'utilisateur clique sur un bouton.

Pour cela il suffit d'affecter cette fonction *calculer* avec ses paramètres, au gestionnaire **onClick** qui réagit à l'événement *click*

Le code à écrire est : Le paramètre **this.form** fait référence au formulaire qui contient le bouton lui-même.

Remarques

On peut utiliser plusieurs gestionnaires d'événements sur un même composant.

Par exemple : `voir ...`

Liste des événements

- **onBlur** : se produit quand un champ Textarea, Text ou Select n'est plus activé.
- **onChange** : se produit quand un champ Textarea, Text ou Select est modifié par l'utilisateur.
- **onClick** : se produit quand un composant Button, Checkbox, Radio Link, Reset ou Submit reçoit un click souris
- **onFocus** : se produit quand un composant Textarea, Text ou Select est activé.
- **onLoad** : se produit quand le navigateur a fini de charger une fenêtre ou toutes les frames d'un FRAMESET. L'événement *onLoad* se positionne dans la balise BODY ou dans la balise FRAMESET
- **onMouseOver** : se produit quand la souris passe sur un Hyperlien ou une zone activable d'une image.
- **onSelect** se produit quand un composant Textarea ou Text est sélectionné.
- **onSubmit** : se produit quand un formulaire est globalement validé appui du bouton *Submit*.
- **onUnload** : se produit quand on quitte un document. L'événement *onUnload* se positionne dans la balise BODY ou dans la balise FRAMESET.
- **onAbort** : se produit quand l'utilisateur interrompt le chargement d'une image
- **onError** : se produit quand le chargement d'une page ou d'une image produit une erreur.
- **onMouseout** : se produit quand la souris quitte une zone Area d'une image ou un hyperlien.
- **onReset** : se produit quand on clique sur le bouton *reset* d'un formulaire

Tableau récapitulatif

Gest. événement	provoqué par l'utilisateur qui ...	sur les objets ...
onBlur	enlève le focus du composant	text, textarea, select

onChange	change la valeur d'un texte ou d'un composant à options	text, textarea, select
onClick	clique sur un composant ou un hyperlien	button, checkbox, radio, reset, submit
onFocus	donne le focus au composant	text, textarea, select
onLoad	charge la page dans le navigateur	balises BODY, FRAMESET
onMouseOut	la souris quitte un lien ou une ancre	balises <A HREF..>, <AREA HREF..>
onMouseOver	bouge la souris sur un lien ou une ancre	balises <A HREF..>, <AREA HREF..>
onReset	efface les saisies d'un formulaire	bouton reset
onSelect	sélectionne une zone d'édition d'un formulaire	text, textarea
onSubmit	soumet un formulaire	bouton submit
onUnload	quitte la page	balises BODY, FRAMESET

Programmer un formulaire

- **Prérequis:** Il est nécessaire de connaître déjà les champs (ou composants) de formulaires du langage HTML.
- Un formulaire *form* se présente comme un objet inclus dans un objet *document* (page HTML).
On sait que lui-même se conduit comme un *conteneur* d'autres objets que sont les composants usuels d'une interface graphique.
Ces objets possèdent des propriétés et des méthodes (fonctions) qui correspondent pratiquement aux attributs des balises HTML qui les construisent .
Pour l'essentiel, la tâche du programmeur est d'analyser les traitements à effectuer sur les informations extraites du formulaire puis d'écrire les fonctions correspondantes, appelées par le déclenchement d'événements par l'utilisateur.

Champ de TEXTE

```
<FORM NAME="Questions">
<INPUT TYPE="text" NAME="classe" VALUE="" SIZE=10> ;
</FORM>
```

- Le texte entré par l'utilisateur est affecté à la propriété **value** du composant **classe** situé dans le formulaire **Questions** placé dans le document courant appelé **document** (par défaut).
- De l'intérieur de ce document, le contenu du champ de texte est donc "visible" et accessible par son nom complet :
document.Questions.classe.value
- En particulier, il peut être récupéré et affecté à une variable **MaClasse** par l'expression:
MaClasse = document.Questions.classe.value
La référence **F.classe.value** est suffisante car la portée du formulaire F est le document où il est inséré.
- Les formulaires dans un même document sont stockés dans un tableau appelé **forms**; le premier formulaire est alors noté *forms[0]*, la deuxième est *forms[1]* etc..
Si **Questions** est situé en position de premier formulaire, on peut remplacer l'affectation précédente par :
MaClasse = document.forms[0].classe.value

Liste de sélection SELECT

```
<FORM NAME="F">
```



```
<SELECT NAME="liste"
  onChange='F.resultat.value+=F.liste.options[F.liste.selectedIndex].value' > ;
<OPTION VALUE="matin ... " >Matin
<OPTION VALUE="midi ... ">Midi
<OPTION VALUE="soir ... ">Soir
</SELECT>
<INPUT TYPE=TEXTE SIZE=40 NAME="resultat" VALUE="Vous avez sélectionné : " >
</FORM>
```

Matin

Propriétés de l'objet SELECT

- **selectedIndex** est une propriété dont la valeur est le numéro de l'élément sélectionné dans la liste
var num = F.liste.**selectedIndex** ---> num est le numéro de l'élément sélectionné dans le composant liste situé dans le formulaire F.
- **options[]** est le tableau prédéfini contenant les *objets* de la liste
- **F.liste.options[num]** est l'objet champ situé au N° num (rappel : le 1er a le numéro 0)
- **F.liste.options[num].value** est la valeur de l'option N° num de la liste.
- **F.liste.options[num].text** est le texte suivant le champ <OPTION>.

Boutons de sélection RADIO

```
<FORM NAME="F">
<INPUT TYPE="RADIO" NAME="choix" value="sup. à 10" CHECKED
  onClick='F.resultat.value=this.value'>plus de 10
<INPUT TYPE="RADIO" NAME="choix" value="entre 8 et 10"
  onClick='F.resultat.value=this.value'>entre 8 et 10
<INPUT TYPE="RADIO" NAME="choix" value="inf. à 8"
  onClick='F.resultat.value=this.value'>moins de 8

<INPUT TYPE=TEXTE SIZE=15 NAME="resultat" VALUE="" >
</FORM>
```

plus de 10
 entre 8 et 10
 moins de 8

Propriétés de l'objet RADIO

- **F.choix[num]** est le bouton radio N° num (le 1er a le numéro 0) de l'ensemble de boutons nommé **choix**
- **checked** est le booléen décrivant l'état d'un bouton : if (F.choix[num].**checked**)
- **F.choix[num].value** est la valeur associée au bouton N° num de la série de bouton.

Cases à cocher CHECKBOX

Le traitement est complètement semblable au cas des boutons radio, à la différence que chaque case possède un nom propre distinct des autres. L'état **checked** de chaque case doit donc être testé individuellement.

Voir l'exemple complet qui suit.

Bouton de validation SUBMIT

Lorsque l'utilisateur "soumet" ses réponses en cliquant sur le bouton **submit**, une fonction programmée en réaction au gestionnaire de l'événement **onSubmit** peut analyser ses choix et y répondre.

On peut plus simplement préférer utiliser un bouton normal qui lance une fonction en réaction à l'événement **onClick()**

Voir l'exemple suivant.

Exemple récapitulatif

Voici la reprise du questionnaire présenté en exemple dans le chapitre formulaire du cours HTML.

Examiner en détail cette annexe. Exécuter plusieurs fois et parallèlement étudier le code JS. Comprendre comment on peut effectuer des traitements à partir des choix de l'utilisateur.

Envoyer un formulaire

Pour envoyer la saisie d'un formulaire sans faire appel à des programmes à placer sur le serveur WEB, on peut envoyer le formulaire par courrier électronique de façon simple pour l'utilisateur.

Il suffit d'ajouter un attribut particulier **ACTION=** dans la balise **<FORM >** qui précise le mode de transmission

Voici comment envoyer le contenu du formulaire par e-mail,

```
<FORM ACTION="mailto:nom destinataire@nom serveur" METHOD="POST">
.....
<INPUT TYPE="SUBMIT" VALUE="Envoyer"
<INPUT TYPE="RESET" VALUE="Effacer"
</FORM>
```

IV – XML

Le XML pour *eXtensible Markup Language* est donc un langage de balises comme le Html mais il est extensible, évolutif. En XML, les balises ne sont pas prédéfinies. C'est vous qui devez ou pouvez définir vos propres balises.

Et c'est là le problème ! Si les braves navigateurs n'avaient plus de difficultés pour afficher les balises prédéfinies du Html comme les <H1>,
 ou autres <TABLE>, que doivent-ils faire avec vos balises <ok> ou <new> ? Le XML a comme vocation de décrire de l'information et pas d'afficher celle-ci. Ainsi le XML pourtant créé en 1999, est resté durant près de deux ans, un concept plutôt abstrait et théorique faute de moyens fiables pour en afficher le résultat. Avec le développement de nouvelles techniques comme le XSL, il est devenu possible de percevoir concrètement les énormes potentialités de ce nouveau langage

La syntaxe du XML

➤ Le XML impose des règles de syntaxe très spécifiques par rapport au Html. En outre, on retrouvera ces mêmes règles de syntaxe dans tous les langages dérivés du XML comme le XHTML ou le WML par exemple.

▶ Le XML est un langage de balises [Markup Language].

Mais au contraire du Html où les balises sont définies, vous devez inventer vos balises. Rappelez-vous, le XML est eXtensible. Il faut donc écrire soi-même le nom des balises utilisées.

Il y a quelques règles pour la composition des noms (mais elle ne déroutent pas les habitués du Javascript) :

- Les noms peuvent contenir des lettres, des chiffres ou d'autres caractères.
- Les noms ne peuvent débuter par un nombre ou un signe de ponctuation.
- Les noms ne peuvent commencer par les lettres xml (ou XML ou Xml...).
- Les noms ne peuvent contenir des espaces.
- La longueur des noms est libre mais on conseille de rester raisonnable.
- On évitera certains signes qui pourraient selon les logiciels, prêter à confusion comme "-", ";", ".", "<", ">", etc.
- Les caractères spéciaux pour nous francophones comme é, à, ê, ï, ù sont à priori permis mais pourraient être mal interprétés par certains programmes.

On profitera de cette liberté dans les noms pour les rendre le plus descriptif possible comme par exemple <gras_et_italique>.

▶ Les balises sont sensibles au majuscules et minuscules [case sensitive].

Ainsi, la balise <Message> est différente de la balise <message>. La balise d'ouverture et la balise de fermeture doivent donc être identiques. Ainsi par exemple ; <Message> ... </message> est incorrect et <message> ... </message> est correct.

Une tendance se dégage pour n'écrire les balises qu'en minuscules, limitant ainsi les erreurs possibles.

▶ Toute balise ouverte doit impérativement être fermée.

Fini les écritures bâclées du Html où l'on pouvait dans certains cas omettre la balise de fin comme pour le paragraphe <p> ou l'élément de liste .

Ainsi en Html, ce qui suit est affiché correctement :

```
<p>
<ul>
<li>Point 1
<li>Point 2
```

Le XML est beaucoup plus strict. On devrait avoir :

```
<p>
<ul>
<li>Point 1</li>
<li>Point 2</li>
<p>
```

Les éventuelles balises uniques ou appelées aussi balises vides, comme
, <meta> ou en Html, doivent également comporter un signe de fermeture soit balise/. Ainsi une balise <meta/> est correcte en XML.

► Les balises doivent être correctement imbriquées.

Le XML étant très préoccupé par la structure des données, des balises mal imbriquées sont des fautes graves de sens.

Ainsi l'écriture suivante est incorrecte car les balises ne sont pas bien imbriquées :
<parent><enfant>Loïc</parent></enfant>

L'écriture correcte avec une bonne imbrication des éléments est :
<parent><enfant>Marine</enfant></parent>

► Tout document XML doit comporter une racine.

En fait, la première paire de balises d'un document XML sera considéré comme la balise de racine [root].

Par exemple : <racine>
... suite du document XML ...
</racine>

Si on ose faire un lien avec le Html, votre élément racine était <body> ... </body>.

Tous les autres éléments seront imbriqués entre ces balises de racine.

Par exemple : <parents>
 <enfants>
 <petits_enfants> ... </petits_enfants>
 </enfants>
</parents>

► Les valeurs des attributs doivent toujours être mises entre des guillemets.

Le XML peut avoir (comme le Html) des attributs avec des valeurs. En XML, les valeurs des attributs doivent obligatoirement être entre des guillemets, au contraire du Html où leur absence n'a plus beaucoup d'importance.

Ainsi, l'écriture suivante est incorrecte car il manque les guillemets.
<date anniversaire=071185>

La bonne écriture est :

```
<date anniversaire="071185">
```

Un premier document XML

➤ Voici un premier document XML.

Rien de bien compliqué mais ce document sera étoffé en cours d'étude.

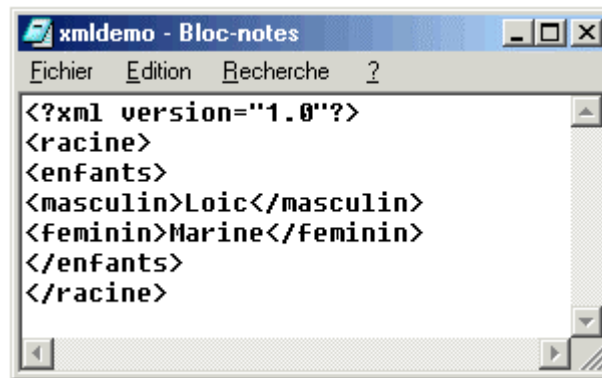
<?xml version="1.0" encoding="ISO-8859-1"?>	
	La déclaration <?xml version="1.0"?> indique au navigateur que ce qui suit est un document XML selon sa version 1.0. Vous remarquerez que cette balise ne comporte pas de signe de fermeture car cette balise n'est pas encore du XML. On en profite généralement pour notifier le "character set" qui indique à l'interpréteur XML [Parser] le jeu de caractères à utiliser. Le jeu de caractères "ISO-8859-1" a, pour nous francophones, l'avantage d'accepter la plupart des lettres avec des accents. Mais il existe d'autres jeux de caractères comme UTF-8 ou UTF-16 plutôt destinés aux anglo-saxons car ils ne reprennent pas les accents.
<racine>	
	L'élément racine indispensable au XML. Vous pouvez utiliser, à votre convenance, n'importe quel nom à l'intérieur de cette balise de racine.
... suite du document XML ...	
	Votre document XML proprement dit, qui respectera bien entendu scrupuleusement la syntaxe du XML ("bien formé").
</racine>	
	Le document XML se termine obligatoirement à la fermeture de la balise de racine.

➤ Elaboration du fichier

Voici un petit fichier XML.

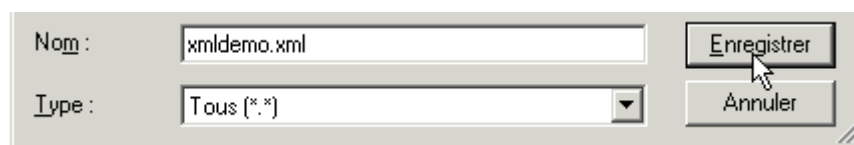
```
<?xml version="1.0"?>
<racine>
<enfants>
<masculin>Loic</masculin>
<feminin>Marine</feminin>
</enfants>
</racine>
```

On le reproduit dans le programme Bloc-notes [notepad] pour les gens de Windows.



```
<?xml version="1.0"?>
<racine>
  <enfants>
    <masculin>Loic</masculin>
    <feminin>Marine</feminin>
  </enfants>
</racine>
```

Et on l'enregistre (*non pas en type de document Texte*) en " Type : Tous (*.*)" sous un nom avec une extension .xml.

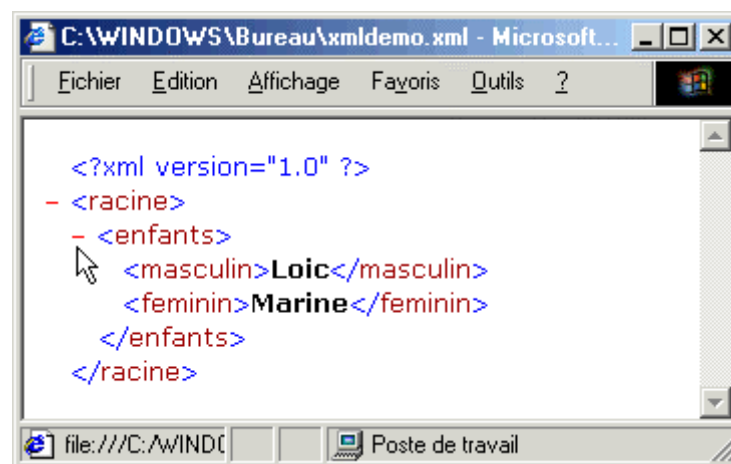


Nom : Enregistrer

Type : Annuler

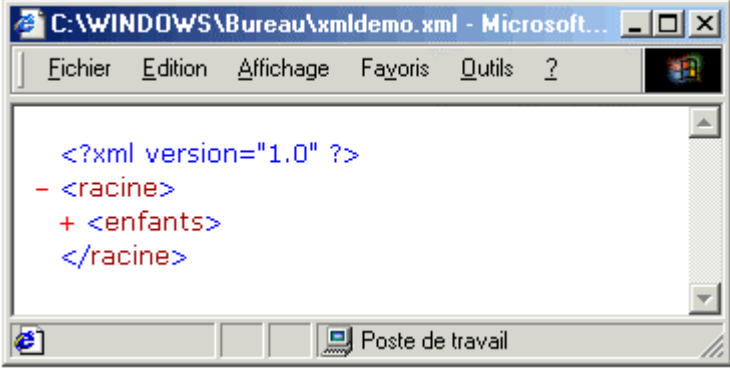
➤ Résultat dans Microsoft Explorer 5 et +

Depuis la version 5 de Microsoft Internet Explorer, les fichiers XML s'affichent sans problèmes.



```
<?xml version="1.0" ?>
- <racine>
- <enfants>
  <masculin>Loic</masculin>
  <feminin>Marine</feminin>
</enfants>
</racine>
```

Vous remarquerez qu'il y a un petit signe - affiché devant des balises englobantes (voir le pointeur sur la capture d'écran). Il suffit de cliquer sur le signe pour masquer celles-ci. Et bien entendu de cliquer sur le signe + pour les faire réapparaître.



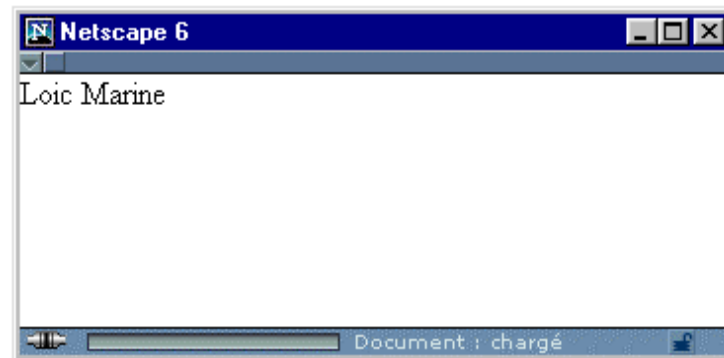
A screenshot of a Microsoft Word window titled "C:\WINDOWS\Bureau\xmldemo.xml - Microsoft...". The window displays XML code in a monospaced font. The code is as follows:

```
<?xml version="1.0" ?>
- <racine>
+ <enfants>
</racine>
```

The status bar at the bottom of the window shows "Poste de travail".

➤ Résultat sous Netscape 6 et +

Le même fichier ne sera visible sur Netscape qu'à partir de la version 6. L'interprétation de ce fichier XML est pour le moins différente.



Le DTD

► Le Document Type Definition

Le DTD ou Document Type Declaration ou encore Document Type Definition est l'ensemble des règles et des propriétés que doit suivre le document XML. Ces règles définissent généralement le nom et le contenu de chaque balise et le contexte dans lequel elles doivent exister. Cette formalisation des éléments est particulièrement utile lorsqu'on utilise de façon récurrente des balises dans un document XML.

L'étude détaillée des DTDs dépassent de loin le cadre de cet ouvrage mais un bref aperçu est cependant utile surtout pour comprendre le fonctionnement des langages dérivés du XML qui ne manquent pas d'utiliser ces fameux DTDs.

En effet, par les DTDs externes, plusieurs concepteurs peuvent se mettre d'accord pour utiliser un DTD commun pour échanger leurs données. Avec le XHTML ou le WML, vous signalez dans l'entête du document que vous utilisez (et suivez) les normes du W3C concernant les langages précités.

► Le DTD interne

On peut inclure son propre DTD au code source du fichier XML. On parlera alors d'un DTD interne.

Le DTD interne suit la syntaxe suivante :

```
<!DOCTYPE                                élément-racine                [
déclaration                               des                            éléments
]>
```

Prenons un fichier comme exemple :

<pre><?xml version="1.0" standalone="yes"?> <!DOCTYPE parent [<!ELEMENT parent (garcon,fille)> <!ELEMENT garcon (#PCDATA)> <!ELEMENT fille (#PCDATA)>]> <parent> <garcon>Loic</garcon> <fille>Marine</fille> </parent></pre>	<p>Comme vous définissez un DTD interne, votre fichier est indépendant (standalone). Début du DTD interne avec parent comme élément de racine.</p> <p>Cet élément racine soit parent contiendra les sous-éléments garcon et fille.</p> <p>#PCDATA indique au Parser XML que l'élément garcon contient des données exprimées en chiffres ou en lettres.</p> <p>Idem pour l'élément fille.</p> <p>Fin du DTD</p> <p>Racine du document XML.</p> <p>Fin du document XML.</p>
--	---

Je ne peux résister à la tentation de livrer un extrait du DTD Strict pour la balise du XHTML.

<pre><!ELEMENT img EMPTY> <!ATTLIST img %attrs; src %URI; #REQUIRED</pre>	<p>La balise img est une balise vide [empty] dont les attributs sont</p> <p>src pour le lien (obligatoire) [required]</p>
--	---

alt	%Text;	#REQUIRED	alt pour le texte alternatif (obligatoire)
longdesc	%URI;	#IMPLIED	longdesc
height	%Length;	#IMPLIED	height
width	%Length;	#IMPLIED	weight
usemap	%URI;	#IMPLIED	usemap
ismap	(ismap)	#IMPLIED	et ismap
>			

► Le DTD externe

Le DTD externe suivra la syntaxe suivante :

```
<!DOCTYPE élément-racine SYSTEM "nom_du_fichier.dtd">
```

Le même fichier que ci-dessus serait alors :

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE parent SYSTEM "parent.dtd">
<parent>
<garcon>Loic</garcon>
<fille>Marine</fille>
</parent>
```

Le fichier de DTD externe (ici dans le même répertoire) "parent.dtd" contiendrait :

```
<!ELEMENT parent (garcon,fille)>
<!ELEMENT garcon (#PCDATA)>
<!ELEMENT fille (#PCDATA)>
```

Mais il est aussi possible de faire référence à un DTD externe situé sur un autre site comme pour

par	exemple	le	XHTML	:
-----	---------	----	-------	---

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//FR"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

► Valide

Dans la littérature relative au XML, on distingue un document "bien formé" d'un document valide.

Un document valide est dit d'un document qui respecte les règles spécifiques de son DTD.

Un document "bien formé" est, pour rappel, un document qui respecte les règles générales de syntaxe du XML.

Afficher le XML avec CSS

➤ Le CSS

Pour afficher les balises XML, on peut faire appel aux bonnes vieilles feuilles de style (CSS), maintenant classiques dans le paysage Html. A chaque balise "inventée" dans le fichier XML, on va définir un élément de style que le navigateur pourra alors afficher.

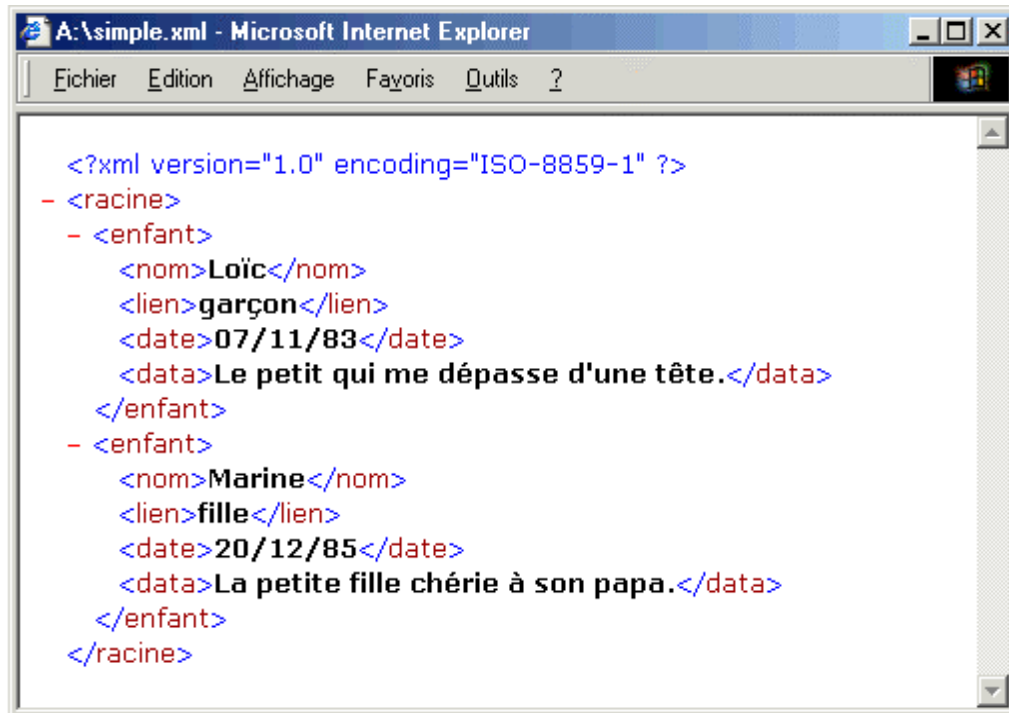
➤ Un exemple de XML + CSS

A seule fin de démonstration, voici un exemple des possibilités d'une feuille de style CSS associée à un document XML.

Voici notre document XML de départ :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<racine>
  <enfant>
    <nom>Loïc</nom>
    <lien>garçon</lien>
    <date>07/11/83</date>
    <data>Le petit qui me dépasse d'une tête.</data>
  </enfant>
  <enfant>
    <nom>Marine</nom>
    <lien>fille</lien>
    <date>20/12/85</date>
    <data>La petite fille chérie à son papa.</data>
  </enfant>
</racine>
```

Affiché dans le navigateur, cela nous donne :

A screenshot of a Microsoft Internet Explorer browser window. The title bar reads 'A:\simple.xml - Microsoft Internet Explorer'. The menu bar includes 'Fichier', 'Edition', 'Affichage', 'Favoris', 'Outils', and '?'. The main content area displays an XML document with the following code:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <racine>
- <enfant>
  <nom>Loïc</nom>
  <lien>garçon</lien>
  <date>07/11/83</date>
  <data>Le petit qui me dépasse d'une tête.</data>
</enfant>
- <enfant>
  <nom>Marine</nom>
  <lien>fille</lien>
  <date>20/12/85</date>
  <data>La petite fille chérie à son papa.</data>
</enfant>
</racine>
```

Tristounet !

On ajoute un fichier .css dont voici le contenu :

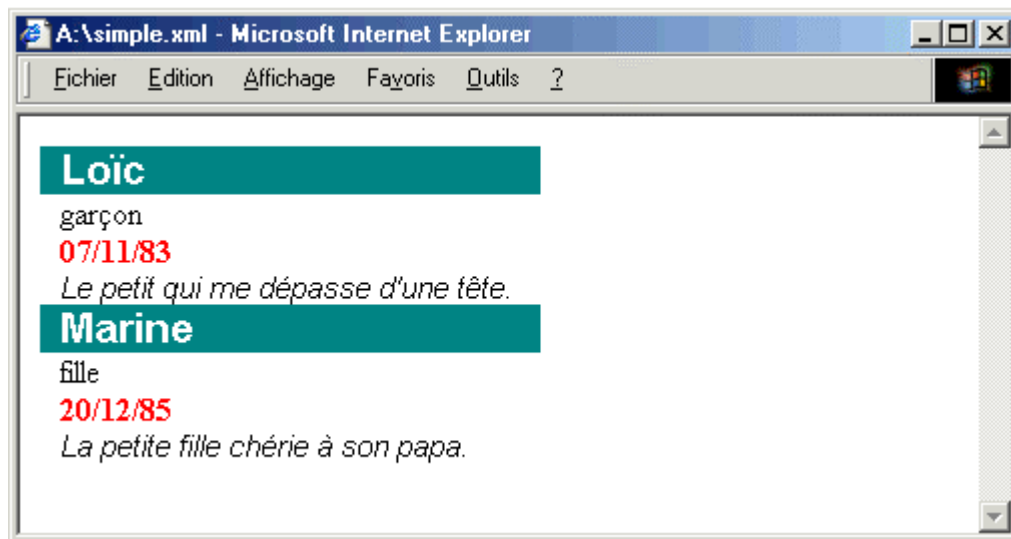
```
<style type="text/css">
racine , enfant {
nom {
  display: block;
  width: 250px;
  font-size: 16pt ;
  font-family: arial ;
```

```
font-size: 11pt ;
font-style: italic;
font-family: arial ;
padding-left: 10px;
}
</style>
```

Après avoir ajouté un lien vers le fichier css dans le fichier xml :

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

On obtient finalement :



Afficher le XML avec XSL

➤ Le XSL - Les feuilles de style du XML

Comme le XML n'utilise pas des balises prédéfinies (car on peut inventer ses propres balises), le navigateur ne "comprend" pas les balises du XML et ne sait pas trop comment afficher un document XML.

Pour néanmoins afficher des documents XML, il est nécessaire d'avoir un mécanisme pour décrire comment le document pourrait être affiché. Un de ces mécanismes est les feuilles de style classiques du Html (CSS), mais le **XSL** pour **eXtensible Stylesheet Language** est de loin un langage de feuille de style plus adapté au XML et donc plus performant..



De façon résumée, le XSL est un langage qui transforme le XML en Html. Mais il fait bien plus ! Ainsi nous avons cru utile de lui consacrer un plus ample développement plus loin dans ce tutorial.

➤ Un exemple de XML + XSL

A seule fin de démonstration, voici un exemple des possibilités du XSL associé à un document XML. Les explications seront données au chapitre consacré au [XSL](#).

Voici notre document XML de départ :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<racine>
  <enfant>
    <nom>Loïc</nom>
    <lien>garçon</lien>
    <date>07/11/83</date>
    <data>Le petit qui me dépasse d'une tête.</data>
  </enfant>
  <enfant>
    <nom>Marine</nom>
    <lien>fille</lien>
    <date>20/12/85</date>
    <data>La petite fille chérie à son papa.</data>
  </enfant>
</racine>
```

Affiché dans le navigateur, cela nous donne :

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <racine>
  - <enfant>
    <nom>Loïc</nom>
    <lien>garçon</lien>
    <date>07/11/83</date>
    <data>Le petit qui me dépasse d'une tête.</data>
  </enfant>
  - <enfant>
    <nom>Marine</nom>
    <lien>fille</lien>
    <date>20/12/85</date>
    <data>La petite fille chérie à son papa.</data>
  </enfant>
</racine>
```

Pas très folichon !

On ajoute un fichier .xsl dont voici le contenu :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<html xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<body style="font-family:Arial; font-size:12pt;">
  <xsl:for-each select="racine/enfant">
    <div style="background-color:teal; color:white;">
      <span style="font-weight:bold; color:white; padding:4px"> <xsl:value-of
```

```

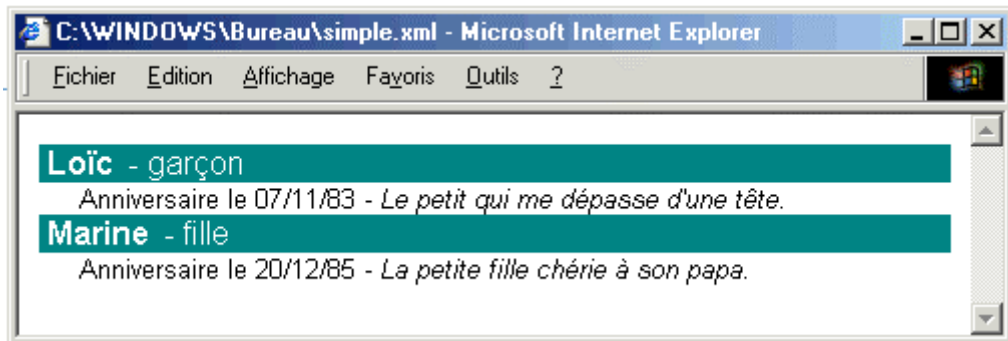
</span >
<span style="font-style:italic"> - <xsl:value-of select="data"/> </span >
</div >
</xsl:for-each>
</body>
</html>

```

Après avoir ajouté un lien vers le fichier xsl dans le fichier xml :

```
<?xml:stylesheet type="text/xsl" href="simple.xsl"?>
```

On obtient finalement :



Afficher du XML dans Html

➤ Du XML dans un fichier Html

On peut toujours incorporer du XML dans un fichier Html avec la balise `<xml> ... </xml>`. Mais en toute logique, quand les navigateurs rencontrent des balises incorrectes ou inconnues, rien n'est affiché. Ce sera le cas avec vos balises XML incorporées dans un fichier Html. Heureusement, on peut passer par une astuce qui répond au doux nom romantique de "îlots de données" [data islands].

➤ Les Data Islands [les îles de données]

Derrière ce nom pour le moins bizarre, se cache une possibilité assez intéressante. Dans un fichier Html, vous pouvez créer un " îlot " de données se trouvant dans un fichier XML distinct et en extraite des données que vous pouvez alors afficher dans le document Html.

Ici, dans le fichier Html, on va désigner le fichier xml extérieur avec un identifiant id :

```
<xml id="fichierxml" src="simple.xml"></xml>
```

Dans un tableau Html, que l'on relie par un attribut à la source des données au moyen de l'identifiant désigné plus haut [datasrc="#id"], on peut finalement aller reprendre des données du fichier XML avec l'attribut de champ de donnée qui reprend comme valeur le nom de la balise XML [datafld="balise_xml"].

Vite, vite, un exemple !

Voilà toujours notre fichier XML (extérieur) :

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<racine>
  <enfant>
    <nom>Loïc</nom>

```

```

<lien>garçon</lien>
<date>07/11/83</date>
<data>Le petit qui me dépasse d'une tête.</data>
</enfant>
<enfant>
  <nom>Marine</nom>
  <lien>fille</lien>
  <date>20/12/85</date>
  <data>La petite fille chérie à son papa.</data>
</enfant>
</racine>

```

Soit :

```

<?xml version="1.0" encoding="ISO-8859-1" ?>
- <racine>
  - <enfant>
    <nom>Loïc</nom>
    <lien>garçon</lien>
    <date>07/11/83</date>
    <data>Le petit qui me dépasse d'une tête.</data>
  </enfant>
  - <enfant>
    <nom>Marine</nom>
    <lien>fille</lien>
    <date>20/12/85</date>
    <data>La petite fille chérie à son papa.</data>
  </enfant>
</racine>

```

Je vais créer un fichier Html classique dans lequel je voudrais reprendre des données du fichier XML et plus précisément le contenu des balises <nom>, <lien> et <date>.

```

<html>
<body>
Voici du Html...
<xml id="fichierxml" src="simple.xml"></xml>
<table border="1" datasrc="#fichierxml">
<tr>
<td><span datafld="nom"></span></td>
<td><span datafld="lien"></span></td>
<td>Anniversaire le <span datafld="date"></span></td>
</tr>
</table>
Et voici encore du Html !
</body>
</html>

```

Ce qui une fois affiché (avec quelques attributs supplémentaires pour le look de la page), offre le résultat suivant :

