Support de cours JavaScript

Note:
La majeure partie des textes composant ce document est issue d'un site Internet:

http://www.editeurjavascript.com

Et certains listings proviennent du livre:

Moncur Michael, *Javascript 1.3*, éd. Le Tout en Poche, CampusPress France, Paris, 1999

A) Introduction

Qu'est-ce que le Javascript?

Le Javascript est une extension du langage HTML qui est incluse dans le code. Ce langage est un langage de programmation qui permet d'apporter des améliorations au langage HTML en permettant d'exécuter des commandes.

A quoi ressemble un script?

Un script est une portion de code qui vient s'insérer dans une page HTML. Le code du script n'est toutefois pas visible dans la fenêtre du navigateur car il est compris entre des balises (ou tags) spécifiques qui signalent au navigateur qu'il s'agit d'un script écrit en langage JavaScript.
Les balises annonçant un code Javascript sont les suivantes:

```
<SCRIPT language="Javascript">
Placez ici le code de votre script
</SCRIPT>
```

Masquage du script pour les anciens browsers

Ce code est ainsi invisible du point de vue du navigateur c'est-à-dire que ce dernier n'affiche pas dans sa fenêtre le code Javascript. Toutefois, d'anciens navigateurs, créés avant l'arrivée du Javascript, ne connaissent pas ces balises et donc les ignorent...
Le code de votre Javascript risque donc de s'afficher sur votre belle page web et venir gâcher votre travail. L'astuce consiste donc à ajouter des balises de commentaires à l'intérieur même des balises de script. Ainsi les anciens navigateurs ignoreront tout simplement l'intégralité du script, tandis que les navigateurs récents l'interprèteront (comme il se le doit) comme du Javascript!
Voici ce que donne le script une fois "masqué" pour les anciens navigateurs:

```
<SCRIPT language="Javascript">
<!--
Placez ici le code de votre script
// -->
</SCRIPT>
```

Ajouter des commentaires dans votre code

Comme dans tout langage de programmation, il est bon d'ajouter des commentaires dans un de ses scripts

- d'une part pour s'y retrouver lorsque, plusieurs mois après, l'on voudra revoir son script
- d'autre part par esprit d'échange pour permettre à vos visiteurs curieux de comprendre votre script (cela n'est pas toujours désiré...)

Il ne faut pas confondre les balises de commentaires du langage HTML (destinées à masquer le script pour certains browsers) et les caractères de commentaires Javascript (permettant de documenter son script)!

Pour écrire des commentaires, Javascript utilise les conventions utilisées en lange C/C++

- Pour mettre en commentaires tout une ligne on utilise le double slash:
  // Tous les caractères derrière le // sont ignorés
- Pour mettre en commentaire une partie du texte (éventuellement sur plusieurs lignes) on utilise le **/*** et le ***/**:

```
/* Toutes les lignes comprises entre ces repères
Sont ignorées par l'interpréteur
de code */
```

Il faut veiller à ne pas embriquer des commentaires, au risque de provoquer une erreur lors de l'exécution du code!

Un exemple de Javascript

Comme généralement dans les tutoriels de Javascript on va faire afficher un boite de dialogue suite au chargement d'une page HTML. Dans ce cas le script est totalement inutile voire ennuyeux pour vos visiteurs... Cet exemple montre ce que l'abus de Javascript peut donner... Il faudra apprendre à se servir du Javascript avec modération!
Voici la page HTML correspondante:

```
<HTML>
<HEAD>
<TITLE> Voici une page contenant du Javascript</TITLE>
</HEAD>
<BODY>
<SCRIPT language="Javascript">
<!--
alert("Voici un message d alerte!");
// -->
</SCRIPT>
</BODY>
</HTML>
```

B) Javascript dans les pages HTML

A quel emplacement insérer le Javascript dans votre page HTML

Il y a plusieurs façon d'inclure du JavaScript dans une page HTML:

- Grâce à la balise <SCRIPT>
- En mettant le code dans un fichier
- Grâce aux événements

Dans la balise script

Le code Javascript peut être inséré où vous le désirez dans votre page Web, vous devez toutefois veiller à ce que le navigateur est entièrement chargé votre script avant d'exécuter une instruction. En effet, lorsque le navigateur charge votre page Web, il la traite de haut en bas, de plus vos visiteurs (souvent impatients) peuvent très bien interrompre le chargement d'une page, auquel cas si l'appel d'une fonction se situe avant la fonction dans votre page il est probable que cela génèrera une erreur si cette fonction n'a pas été chargée.

Ainsi, on place généralement le maximum d'éléments du script dans la balise d'en-tête (ce sont les éléments située entre les balises <HEAD> et </HEAD>). Les évènements Javascript seront quant à eux placés dans le corps de la page (entre les balises <BODY> et </BODY>) comme attribut d'une commande HTML...

```
<SCRIPT language="Javascript">
<!--
Placez ici le code de votre script
// -->
</SCRIPT>
```

L'argument de la balise <SCRIPT> décrit le langage utilisé. Il peut être "JavaScript" "JavaScript1.1" "JavaScript1.2".

On peut ainsi (en passant un argument différent de "JavaScript") utiliser d'autres langages de programmation que celui-ci (par exemple le VbScript).

Pour utiliser différentes versions de JavaScript tout en conservant une certaine compatibilité, il suffit de déclarer plusieurs balises **SCRIPT** ayant chacune comme paramètre la version du JavaScript correspondante.

Dans un fichier externe

Il est possible de mettre les codes de JavaScript en annexe dans un fichier (à partir de Netscape 3.0 uniquement). Le code à insérer est le suivant:

```
<SCRIPT LANGUAGE=Javascript SRC="url/fichier.js"> </SCRIPT>
```

Où url/fichier.js correspond au chemin d'accès au fichier contenant le code en JavaScript, sachant que si celui-ci n'existe pas le navigateur exécutera le code inséré entre les 2 balises.

Grâce aux évènements

On appelle évènement une action de l'utilisateur, comme le clic d'un des boutons de la souris. Le code dans le cas du résultat d'un événement s'écrit:

<balise **eventHandler**="code Javascript à insérer">

eventHandler représente le nom de l'événement.

C) Les Variables

Le concept de variable

Une variable est un objet repéré par son nom, pouvant contenir des données, qui pourront être modifiées lors de l'exécution du programme.

En Javascript, les noms de variables peuvent être aussi long que l'on désire, mais doivent répondre à certains critères:

- un nom de variable doit commencer par une lettre (majuscule ou minuscule) ou un "_"
- un nom de variables peut comporter des lettres, des chiffres et les caractères _ et & (les espaces ne sont pas autorisés!)
- Les noms de variables ne peuvent pas être les noms suivants (qui sont des noms réservés):
- abstract
- boolean break byte
- case catch char class
- default do double
- else extends
- false final finally float
- goto
- if, implements, import, in, instanceof, int, interface
- long
- native, new, null
- package, private, protected, public
- return
- short, static, super, switch, synchronized
- this, throw, throws, transient, true, try
- var, void
- while, with

Les noms de variables sont sensibles à la casse (le Javascript fait la différence entre un nom en majuscule et un nom en minuscules), il faut donc veiller à utiliser des noms comportant la même casse!

La déclaration de variables

Le Javascript étant très souple au niveau de l'écriture (à double-tranchant car il laisse passer des erreurs...), la déclaration des variables peut se faire de deux façons:

- soit de façon explicite, en faisant précéder la variable du mot clé *var* qui permet d'indiquer de façon rigoureuse qu'il s'agit d'une variable:
  var chaine= "bonjour"
- soit de façon implicite, en écrivant directement le nom de la variable suivie du caractère = et de la valeur à affecter:
  chaine= "bonjour"

Le navigateur détermine seul qu'il s'agit d'une déclaration de variable.

Même si une déclaration implicite est tout à fair reconnue par le navigateur, il est tout de même plus rigoureux de déclarer les variables de façon explicite avec le mot *var*...

Portée (visibilité) des variables

Selon l'endroit où on déclare une variable, celle-ci pourra être accessible (visible) de partout dans le script ou bien que dans une portion confinée du code, on parle de "portée" d'une variable.

Lorsqu'une variable est déclarée sans le mot clé *var*, c'est-à-dire **de façon implicite**, elle est accessible de partout dans le script (n'importe quelle fonction du script peut faire appel à cette variable). On parle alors de **variable globale**

Lorsque l'on déclare **de façon explicite** une variable en javascript (en précédant sa déclaration avec le mot *var*), sa portée dépend de l'endroit où elle est déclarée.

- Une variable déclarée au début du script, c'est-à-dire avant toute fonction sera globale, on pourra alors les utiliser à partir de n'importe quel endroit dans le script
- Une variable déclarée par le mot clé *var* dans une fonction aura une portée limitée à cette seule fonction, c'est-à-dire qu'elle est inutilisable ailleurs, on parle alors de variable locale

Les types de données dans les variables

En Javascript il n'y a pas besoin de déclarer le type de variables que l'on utilise, contrairement à des langages évolués tels que le langage C ou le Java pour lesquels il faut préciser s'il s'agit d'entier (*int*), de nombre à virgule flottante (*float*), de caractères (*char*), ...

En fait le Javascript n'autorise la manipulation que de 4 types de données:

- **des nombres**: entiers ou à virgules
- **des chaînes de caractères** (string): une suite de caractères
- **des booléens**: des variables à deux états permettant de vérifier une condition
- true: si le résultat est vrai
- false: lors d'un résultat faux
- **des variables de type *null***: un mot caractèristique pour indiquer qu'il n'y a pas de données

Nombre entier

Un nombre entier est un nombre sans virgule qui peut être exprimé dans différentes bases:

- Base décimale: L'entier est représenté par une suite de chiffre unitaires (de 0 à 9) ne devant pas commencer par le chiffre 0
- Base hexadécimale: L'entier est représenté par une suite d'unités (de 0 à 9 ou de A à F (ou a à f)) devant commencer par *0x* ou *0X*
- Base octale: L'entier est représenté par une suite d'unités (incluant uniquement des chiffres de 0 à 7) devant commencer par *0*

Nombre à virgule (*float*)

Un nombre à virgule flottante est un nombre à virgule, il peut toutefois être représenté de différentes façons:

- un entier décimal: 895
- un nombre comportant un point (et non une virgule): 845.32
- une fraction: 27/11
- un nombre exponentiel, c'est-à-dire un nombre (éventuellement à virgule) suivi de la lettre *e* (ou *E*), puis d'un entier correspondant à la puissance de 10 (signé ou non, c'est-à-dire précédé d'un + ou d'un -)

Chaîne de caractères (*string*)

Une chaîne de caractère est, comme son nom l'indique, une suite de caractères. On la représente par la suite de caractères encadrée par des guillemets simples (') ou doubles ("), sachant que les deux types de guillemets ne peuvent être mélangés pour une même chaîne de caractères, ce qui signifie que les guillemets dans une chaîne de caractères existent par paire.

Il existe des caractères spéciaux à utiliser dans les chaînes pour simuler d'une part des caractères non visuels ou pour éviter au navigateur de confondre les caractères d'une chaîne avec ceux du script, ces caractères sont précédés d'un antislash (\):

- \b : touche de suppression
- \f : formulaire plein
- \n : retour à la ligne
- \r : appui sur la touche *ENTREE*
- \t : tabulation
- \" : guillemets doubles
- \' : guillemets simples
- \\ : caractère antislash

Ainsi, si on veut stocker la chaine suivante dans la variable *Titre*:
Qu'y a-t'il dans "c:\windows\"

Il faudra écrire dans le code Javascript:
```
Titre = "Qu'y a-t'il dans \"c:\\windows\\\"";
ou
Titre = 'Qu\'y a-t\'il dans \"c:\\windows\\\"';
```

Booléens (*booleans*)

Un booléen est une variable spécial servant à évaluer une condition, il peut donc avoir deux valeurs:

- vrai: représenté par 1
- faux: représenté par 0

D) Les chaînes de caractères

Les particularités de l'objet *String*

string est un mot anglais qui signifie "*chaîne*", il s'agit en fait de chaîne de caractères. C'est donc une suite de caractères, on la représente généralement par un ensemble de caractères encadré par des guillemets. La longueur maximale d'une telle chaîne est à priori comprise entre 50 et 80 caractères.

L'objet *String* est un objet qui contient un certain nombre de propriétés et de méthodes permettant la manipulation de chaînes de caractères.

Les propriétés de l'objet *String*

L'objet *string* a une seule propriété : la propriété *length* qui permet de retourner la longueur d'une chaîne de caractères. Cette propriété est très utile car lorsque l'on traite une chaîne de caractères on aime généralement savoir à quel moment s'arrêter.
La syntaxe de la propriété *length* est la suivante:

```
x = nom_de_la_chaine.length;
x = ('chaine de caracteres').length;
```

On peut donc directement passer la chaîne de caractères comme objet, en délimitant la chaîne par des apostrophes et en plaçant le tout entre parenthèses.
La méthode traditionnelle consistant à appliquer une propriété à une variable de type *string* fonctionne bien évidemment aussi.

Les méthodes de l'objet *String*

Les méthodes de l'objet *string* permettent de récupérer une portion d'une chaîne de caractère, ou bien de la modifier.
Pour comprendre les méthodes suivantes, il est tout d'abord nécessaire de comprendre comment est stockée une chaîne de caractères:
Il s'agit en fait d'une sorte de tableau constitué de *n* caractères (*n* est donc le nombre de caractères), on note 0 la position du premier caractère (celui à l'extrême gauche), puis on les compte de gauche à droite en incrémentant ce nombre:

Méthode	description
charAt(chaîne, position**)** ou **chaîne.charAt(**position**)**	Retourne le caractère situé à la position donnée en paramètre
indexOf(sous-chaîne, position**)**	Retourne la position d'une sous-chaîne (lettre ou groupe de lettres) dans une chaîne de caractère, en effectuant la recherche de gauche à droite, à partir de la position spécifiée en paramètre.
lastIndexOf(sous-chaîne, position**)**	La méthode est similaire à *indexOf()*, à la différence que la recherche se fait de droite à gauche: Retourne la position d'une sous-chaîne (lettre ou groupe de lettres) dans une chaîne de caractère, en effectuant la recherche de droite à gauche, à partir de la position spécifiée en paramètre.
substring(position1, position2**)**	La méthode retourne la sous-chaîne (lettre ou groupe de lettres) comprise entre les positions 1 et 2 données en paramètre.
toLowerCase()	Convertit tous les caractères d'une chaîne en minuscule
toUpperCase()	Convertit tous les caractères d'une chaîne en majuscule
toUpperCase()	Convertit tous les caractères d'une chaîne en majuscule

D'autres methodes

Méthode	description
anchor(nom_a_donner)	Transforme le texte en ancrage HTML
big()	Augmente la taille de la police
blink()	Transforme la chaîne en texte clignotant
bold()	Met le texte en **gras** (balise <B>)
fixed	Transforme le texte en police fixe (balise <TT>)
fontcolor(couleur**)**	Modifie la couleur du texte (admet comme argument la couleur en hexadécimal ou en valeur littérale)
fontsize(Size**)**	Modifie la taille de la police, en afectant la valeur passée en paramètre
italics()	Transforme le texte en *italique* (balise <I>)
link(URL**)**	Transforme le texte en hypertexte (balise <A href>)
small()	Diminue la taille de la police
strike()	Transforme le texte en ~~texte barré~~ (balise <strike>)
sub()	Transforme le texte en $_{\text{indice}}$ (balise <sub>)
sup()	Transforme le texte en $^{\text{exposant}}$ (balise <sup>)

E) Les événements

Qu'appelle-t'on un événement?

Les évènements sont des actions de l'utilisateur, qui vont pouvoir donner lieu à une interactivité. L'événement par excellence est le clic de souris, car c'est le seul que le HTML gère. Grâce au Javascript il est possible d'associer des fonctions, des méthodes à des événements tels que le passage de la souris au-dessus d'une zone, le changement d'une valeur, ...
Ce sont les gestionnaires d'événements qui permettent d'associer une action à un événement. La syntaxe d'un gestionnaire d'événement est la suivante: on*Evenement*="Action_Javascript_ou_Fonction();"

Les gestionnaires d'événements sont associés à des objets, et leur code s'insèrent dans la balise de ceux-ci...

Liste des événements

Evénement (gestionnaire d'événement)	**Description**
Click (onClick)	Se produit lorsque l'utilisateur clique sur l'élément associé à l'événement
Load (onLoad)	Se produit lorsque le navigateur de l'utilisateur charge la page en cours
Unload (onUnload)	Se produit lorsque le navigateur de l'utilisateur quitte la page en cours
MouseOver (onMouseOver)	Se produit lorsque l'utilisateur positionne le curseur de la souris au-dessus d'un élément
MouseOut (onMouseOut)	Se produit lorsque le curseur de la souris quitte un élément Cet événement fait partie du Javascript 1.1
Focus (onFocus)	Se produit lorsque l'utilisateur donne le focus à un élément, c'est-à-dire que cet élément est sélectionné comme étant l'élément actif
Blur (onBlur)	Se produit lorsque l'élément perd le focus, c'est-à-dire que l'utilisateur clique hors de cet élément, celui-ci n'est alors plus sélectionné comme étant l'élément actif
Change (onChange)	Se produit lorsque l'utilisateur modifie le contenu d'un champ de données
Select (onSelect)	Se produit lorsque l'utilisateur sélectionne un texte (ou une partie d'un texte) dans un champ de type "text" ou "textarea"
Submit (onSubmit)	Se produit lorsque l'utilisateur clique sur le bouton de soumission d'un formulaire (le bouton qui permet d'envoyer le formulaire)

Objets auxquels on peut associer des événements

Chaque événement ne peut pas être associé à n'importe quel objet... il est évident qu'un événement *OnChange* ne peut pas s'appliquer à un lien hypertexte...

Objet	Evénements associables
Lien hypertexte	onClick, onMouseOver, onMouseOut
Page du navigateur	onLoad, onUnload
Bouton, Case à cocher, Bouton radio, Bouton Reset	onClick
Liste de sélection d'un formulaire	onBlur, onChange, onFocus
Bouton Submit	onSubmit
Champ de texte et zone de texte	onBlur, onChange, onFocus, onSelect

F) Les Opérateurs

Qu'est-ce qu'un opérateur?

Les opérateurs sont des symboles qui permettent de manipuler des variables, c'est-à-dire effectuer des opérations, les évaluer, ...
On distingue plusieurs types d'opérateurs:

- les opérateurs de calcul
- les opérateurs d'assignation
- les opérateurs d'incrémentation
- les opérateurs de comparaison
- les opérateurs logiques
- (les opérateurs bit-à-dit)
- (les opérateurs de rotation de bit)

Les opérateurs de calcul

Les opérateurs de calcul permettent de modifier mathématiquement la valeur d'une variable

Opérateur	Dénomination	Effet	Exemple	Résultat (avec x valant 7)
+	opérateur d'addition	Ajoute deux valeurs	x+3	10
-	opérateur de soustraction	Soustrait deux valeurs	x-3	4
*	opérateur de multiplication	Multiplie deux valeurs	x*3	21
/	plus: opérateur de division	Divise deux valeurs	x/3	2.3333333
=	opérateur d'affectation	Affecte une valeur à une variable	x=3	Met la valeur 3 dans la variable x

Les opérateurs d'assignation

Ces opérateurs permettent de simplifier des opérations telles que *ajouter une valeur dans une variable et stocker le résultat dans la variable*. Une telle opérations s'écrirait habituellement de la façon suivante par exemple: *x=x+2*
Avec les opérateurs d'assignation il est possible d'écrire cette opération sous la forme suivante: *x+=2*
Ainsi, si la valeur de x était 7 avant opération, elle sera de 9 après...

Les autres opérateurs du même type sont les suivants:

Opérateur	Effet
+=	addition deux valeurs et stocke le résultat dans la variable (à gauche)
-=	soustrait deux valeurs et stocke le résultat dans la variable
*=	multiplie deux valeurs et stocke le résultat dans la variable
/=	divise deux valeurs et stocke le résultat dans la variable

Les opérateurs d'incrémentation

Ce type d'opérateur permet de facilement augmenter ou diminuer d'une unité une variable. Ces opérateurs sont très utiles pour des structures telles que des boucles, qui ont besoin d'un compteur (variable qui augmente de un en un).

Un opérateur de type *x++* permet de remplacer des notations lourdes telles que *x=x+1* ou bien *x+=1*

Opérateur	Dénomination	Effet	Syntaxe	Résultat (avec x valant 7)
++	Incrémentation	Augmente d'une unité la variable	x++	8
--	Décrémentation	Diminue d'une unité la variable	x--	6

Les opérateurs de comparaison

Opérateur	Dénomination	Effet	Exemple	Résultat (avec x valant 7)
== A ne pas confondre avec le signe d'affectation (=)!!	opérateur d'égalité	Compare deux valeurs et vérifie leur égalité	x==3	Retourne 1 si X est égal à 3, sinon 0
<	opérateur d'infériorité stricte	Vérifie qu'une variable est strictement inférieure à une valeur	x<3	Retourne 1 si X est inférieur à 3, sinon 0
<=	opérateur d'infériorité	Vérifie qu'une variable est inférieure ou égale à une valeur	x<=3	Retourne 1 si X est inférieur à 3, sinon 0
>	opérateur de supériorité stricte	Vérifie qu'une variable est strictement supérieure à une valeur	x>3	Retourne 1 si X est supérieur à 3, sinon 0
>=	opérateur de supériorité	Vérifie qu'une variable est supérieure ou égale à une valeur	x>=3	Retourne 1 si X est supérieur ou égal à 3, sinon 0
!=	opérateur de différence	Vérifie qu'une variable est différente d'une valeur	x!=3	Retourne 1 si X est différent de 3, sinon 0

Les opérateurs logiques (booléens)

Ce type d'opérateur permet de vérifier si plusieurs conditions sont vraies:

Opérateur	Dénomination	Effet	Syntaxe
||	OU logique	Vérifie qu'une des conditions est réalisée	((condition1)||(condition2))
&&	ET logique	Vérifie que toutes les conditions sont réalisées	((condition1)&&(condition2))
!	NON logique	Inverse l'état d'une variable booléenne (retourne la valeur 1 si la variable vaut 0, 0 si elle vaut 1)	(!condition)

G) Les structures conditionnelles

On appelle *structure conditonnelle* les instructions qui permettent de tester si une condition est vraie ou non, ce qui permet de donner de l'interactivité à vos scripts par exemple.

L'instruction if

L'instruction if est la structure de test la plus basique, on la retrouve dans tous les langages (avec une syntaxe différente...). Elle permet d'exécuter une série d'instruction si jamais une condition est réalisée.

La syntaxe de cette expression est la suivante:
```
if (condition réalisé) {
liste d'instructions
}
```

Remarques:

- la condition doit être entre des parenthèses
- il est possible de définir plusieurs conditions à remplir avec les opérateurs ET et OU (&& et ||)

par exemple:
`if ((condition1)&&(condition2))` teste si les deux conditions sont vraies
`if ((condition1)||(condition2))` exécutera les instructions si l'une ou l'autre des deux conditions est vraie

- s'il n'y a qu'une instruction, les accolades ne sont pas nécessaires...

L'instruction if ... else

L'instruction *if* dans sa forme basique ne permet de tester qu'une condition, or la plupart du temps on aimerait pouvoir choisir les instructions à exécuter **en cas de non réalisation de la condition**...
L'expression *if ... else* permet d'exécuter une autre série d'instruction en cas de non-réalisation de la condition.

La syntaxe de cette expression est la suivante:
```
if (condition réalisé) {
liste d'instructions
}
else {
autre série d'instructions
}
```

une façon plus courte de faire un test

Il est possible de faire un test avec une structure beaucoup moins lourde grâce à la structure suivante:
***(**condition**) ?** instruction si vrai **:** instruction si faux*

Remarques:

- la condition doit être entre des parenthèses
- Lorsque la condition est vraie, l'instruction de gauche est exécutée
- Lorsque la condition est fausse, l'instruction de droite est exécutée

Les boucles

Les boucles sont des structures qui permettent d'exécuter plusieurs fois la même série d'instructions jusqu'à ce qu'une condition ne soit plus réalisée...

La façon la plus commune de faire une boucle, est de créer un compteur (une variable qui s'incrémente, c'est-à-dire qui augmente de 1 à chaque tour de boucle) et de faire arrêter la boucle lorsque le compteur dépasse une certaine valeur.

La boucle for

L'instruction *for* permet d'exécuter plusieurs fois la même série d'instructions: c'est une boucle!

Dans sa syntaxe, il suffit de préciser le nom de la variable qui sert de compteur (et éventuellement sa valeur de départ, la condition sur la variable pour laquelle la boucle s'arrête (basiquement une condition qui teste si la valeur du compteur dépasse une limite) et enfin une instruction qui incrémente (ou décrémente) le compteur.

La syntaxe de cette expression est la suivante:
```
for (compteur; condition; modification du compteur) {
liste d'instructions
}
```

Par exemple:
```
for (i=1; i<6; i++) {
Alert(i)
}
```
Cette boucle affiche 5 fois la valeur de *i*, c'est-à-dire 1,2,3,4,5
Elle commence à i=1, vérifie que i est bien inférieur à 6, etc... jusqu'à atteindre la valeur i=6, pour laquelle la condition ne sera plus réalisée, la boucle s'interrompra et le programme continuera son cours.

* il faudra toujours vérifier que la boucle a bien une condition de sortie (i.e le compteur s'incrémente correctement)
* une instruction *Alert(i);* dans votre boucle est un bon moyen pour vérifier la valeur du compteur pas à pas!
* il faut bien compter le nombre de fois que l'on veut faire exécuter la boucle:
* *for(i=0;i<10;i++)* exécute 10 fois la boucle (i de 0 à 9)
* *for(i=0;i<=10;i++)* exécute 11 fois la boucle (i de 0 à 10)
* *for(i=1;i<10;i++)* exécute 9 fois la boucle (i de 1 à 9)
* *for(i=1;i<=10;i++)* exécute 10 fois la boucle (i de 1 à 10)

L'instruction while

L'instruction *while* représente un autre moyen d'exécuter plusieurs fois la même série d'instructions.

La syntaxe de cette expression est la suivante:
```
while (condition réalisée) {
liste d'instructions
}
```

Cette instruction exécute la liste d'instructions **tant que** (*while* est un mot anglais qui signifie *tant que*) la condition est réalisée.

La condition de sortie pouvant être n'importe quelle structure conditionnelle, les risques de boucle infinie (boucle dont la condition est toujours vraie) sont grands, c'est-à-dire qu'elle risque de provoquer un plantage du navigateur!

Saut inconditionnel

Il peut être nécessaire de faire sauter à la boucle une ou plusieurs valeurs sans pour autant mettre fin à celle-ci.

La syntaxe de cette expression est "*continue;*" (cette instruction se place dans une boucle!), on l'associe généralement à une structure conditionnelle, sinon les lignes situées entre cette instruction et la fin de la boucle seraient obsolètes.

Exemple: Imaginons que l'on veuille imprimer pour x allant de 1 à 10 la valeur de 1/(x-7) ... il est évident que pour x=7 il y aura une erreur. Heureusement, grâce à l'instruction *continue* il est possible de traiter cette valeur à part puis de continuer la boucle!

```
x=1
while (x<=10) {
if (x == 7) {
Alert('division par 0');
continue;
}
a = 1/(x-7);
Alert(x);
x++
}
```

Il y avait une erreur dans ce script... peut-être ne l'avez-vous pas vue:
Lorsque x est égal à 7, le compteur ne s'incrémente plus, il reste constamment à la valeur 7, il aurait fallu écrire:

```
x=1
while (x<=10) {
if (x == 7) {
Alert('division par 0');
x++;
continue;
}
a = 1/(x-7);
Alert(x);
x++
}
```

Arrêt inconditionnel

A l'inverse, il peut être voulu d'arrêter prématurément la boucle, pour une autre conditon que celle précisé dans l'en-tète de la boucle. L'instruction *break* permet d'arrêter une boucle (*for* ou bien *while*). Il s'agit, tout comme *continue*, de l'associer à une structure conditionnelle, sans laquelle la boucle ne ferait jamais plus d'un tour!

Dans l'exemple de tout à l'heure, par exemple si l'on ne savait pas à quel moment le dénominateur (x-7) s'annule (bon...OK...pour des équations plus compliquées par exemple) il serait possible de faire arrêter la boucle en cas d'annulation du dénominateur, pour éviter une division par zéro!

```
for (x=1; x<=10; x++) {
a = x-7;
if (a == 0) {
Alert('division par 0');
break;
}
Alert(1/a);
}
```

H) Les Fonctions

On appelle *fonction* un sous-programme qui permet d'effectuer un ensemble d'instruction par simple appel de la fonction dans le **corps** du programme principal. Cette notion de sous-programme est généralement appelée fonction dans les langages autres que le Javascript (touefois leur syntaxe est généralement différente...). Les fonctions et les permettent d'exécuter dans plusieurs parties du programme une série d'instruction, cela permet une simplicité du code et donc une taille de programme minimale. D'autre part, une fonction peut faire appel à elle-même, on parle alors de fonction récursive (il ne faut pas oublier de mettre une condition de sortie au risque sinon de ne pas pouvoir arrêter le programme...).

Javascript contient des fonctions prédéfinies qui peuvent s'appliquer pour un ou plusieurs types d'objets spécifiques, on appelle ces fonctions des *méthodes*.

La déclaration d'une fonction

Avant d'être utilisée, une fonction doit être définie car pour l'appeler dans le corps du programme il faut que le navigateur la connaisse, c'est-à-dire qu'il connaisse son nom, ses arguments et les instructions qu'elle contient. La définition d'une fonction s'appelle "*déclaration*". La déclaration d'une fonction se fait grâce au mot clé function selon la syntaxe suivante:

```
function Nom_De_La_Fonction(argument1, argument2, ...) {
liste d'instructions
}
```

Remarques:

- le mot clé function est suivi du nom que l'on donne à la fonction
- le nom de la fonction suit les mêmes règles que les noms de variables :
- le nom doit commencer par une lettre
- un nom de fonction peut comporter des lettres, des chiffres et les caractères _ et & (les espaces ne sont pas autorisés!)
- le nom de la fonction, comme celui des variables est sensible à la casse (différenciation entre les minuscules et majuscules)
- Les arguments sont facultatifs, mais s'il n'y a pas d'arguments, les parenthèses doivent rester présentes
- Il ne faut pas oublier de refermer les accolades
- Le nombre d'accolades ouvertes (fonction, boucles et autres structures) doit être égal au nombre de parenthèses fermées!
- La même chose s'applique pour les parenthèses, les crochets ou les guillemets!

Une fois cette étape franchie, votre fonction ne s'exécutera pas tant que l'on ne fait pas appel à elle quelque part dans la page!

Appel de fonction

Pour exécuter une fonction, il suffit de faire appel à elle en écrivant son nom (une fois de plus en respectant la casse) suivie d'une parenthèse ouverte (éventuellement des arguments) puis d'une parenthèse fermée:

```
Nom_De_La_Fonction();
```

Remarques:

- le point virgule signifie la fin d'une instruction et permet au navigateur de distinguer les différents blocs d'instructions
- si jamais vous avez défini des arguments dans la déclaration de la fonction, il faudra veiller à les inclure lors de l'appel de la fonction (le même nombre d'arguments séparés par des virgules!)

Veillez toujours à ce qu'une fonction soit déclarée avant d'être appelée, sachant que le navigateur traite la page de haut en bas (Pour éviter des erreurs de ce type on déclare

généralement les fonctions dans des balises SCRIPT situées dans l'en-tête de la page, c'est-à-dire entre les balises <SCRIPT> et </SCRIPT>)

Grâce au gestionnaire d'évènement *onLoad* (à placer dans la balise BODY) il est possible d'exécuter une fonction au chargement de la page, comme par exemple l'initialisation des variables pour votre script, et/ou le test du navigateur pour savoir si celui-ci est apte à faire fonctionner le script.
Il s'utilise de la manière suivante:

```
<HTML>
<HEAD>
<SCRIPT language="Javascript">
<!--

function Chargement() {
alert('Bienvenue sur le site');
}

//-->
</SCRIPT>
</HEAD>
<BODY onLoad="Chargement();" >

Javascript qui ne sert absolument à rien si ce n'est déranger vos visiteurs...

</BODY>
</HTML>
```

Les paramètres d'une fonction

Il est possible de passer des paramètres à une fonction, c'est-à-dire lui fournir une valeur ou le nom d'une variable afin que la fonction puisse effectuer des opérations sur ces paramètres ou bien grâce à ces paramètres.
Lorsque vous passez plusieurs paramètres à une fonction il faut les séparer par des virgules, aussi bien dans la déclaration que dans l'appel et il faudra veiller à bien passer le bon nombre de paramètres lors de l'appel au risque sinon de créer une erreur dans votre Javascript...

Imaginons que l'on veuille créer une page Web qui affiche une boîte de dialogue (les boîtes de dialogues sont à éviter dans vos pages car elles sont énervantes, mais c'est toutefois une manière simple d'apprendre le Javascript) qui affiche un texte différent selon le lien sur lequel on appuie.
La méthode de base consiste à faire une fonction pour chaque texte à afficher:

```
<HTML>
<HEAD>
<SCRIPT language="Javascript">
<!--

function Affiche1() {
alert('Texte 1');
}
function Affiche2() {
alert('Texte2');
}

//-->
</SCRIPT>
</HEAD>
<BODY onLoad="Chargement();" >

<A href="javascript:;" onClick="Affiche1();">Texte1</A>
<A href="javascript:;" onClick="Affiche2();">Texte2</A>

</BODY>
</HTML>
```

Il existe toutefois une methode plus "classe" qui consiste à créer une fonction qui a comme paramètre le texte à afficher:

```
<HTML>
<HEAD>
<SCRIPT language="Javascript">
<!--

function Affiche(Texte) {
alert(Texte);
}

//-->
</SCRIPT>
</HEAD>
<BODY onLoad="Chargement();" >

<A href="javascript:;" onClick="Affiche('Texte1');">Texte1</A>
<A href="javascript:;" onClick="Affiche('Texte2');">Texte2</A>

</BODY>
</HTML>
```

Résultat: Aucune différence visuellement mais vous n'avez plus qu'une seule fonction qui peut vous afficher n'importe quel texte...

Travailler sur des variables dans les fonctions

Lorsque vous manipulerez des variables dans des fonctions, il vous arrivera de constater que vous avez beau modifier la variable dans la fonction celle-ci retrouve sa valeur d'origine dès que l'on sort de la fonction...

Cela est du à la portée des variables, c'est-à-dire si elles ont été définies comme variables globales ou locales.

- Une variable déclarée implicitement (non préc&eracute;dée du mot var) sera globale , c'est-à-dire accessible après exécution de la fonction
- Une variable déclarée explicitement (préc&eracute;dée du mot var) sera locale , c'est-à-dire accessible uniquement dans la fonction, toute référence à cette variable hors de la fonction provoquera une erreur (variable inconnue)...

Le mot-clé *this*

Lorsque vous faîtes appel à une fonction à partir d'un objet, par exemple un formulaire, le mot clé *this* fait référence à l'objet en cours et vous évite d'avoir à définir l'objet en tapant *window.objet1.objet2...* ainsi lorsque l'on passe l'objet en cours en paramètre d'une fonction, il suffit de taper *nom_de_la_fonction(this)* pour pouvoir manipuler cet objet à partir de la fonction.

Pour manipuler les propriétés de l'objet il suffira de taper *this.propriete* (où *propriete* représente bien sûr le nom de la propriété).

I) Les Objets du Navigateur

Lorsque vous ouvrez une page Web, le navigateur crée des objets prédéfinis correspondant à la page Web, à l'état du navigateur, et peuvent donner beaucoup d'informations qui vous seront utiles.

Les objets de base du navigateur sont les suivants:

- navigator: qui contient des informations sur le navigateur de celui qui visite la page
- window: c'est l'objet où s'affiche la page, il contient donc des propriétés concernant la fenêtre elle-même mais aussi tous les objets-enfants contenus dans celle-ci
- location: contient des informations relatives à l'adresse de la page à l'écran
- history: c'est l'historique, c'est-à-dire la liste de liens qui ont été visités précédemment
- document: il contient les propriétés sur le contenu du document (couleur d'arrière plan, titre, ...)

Ces objets sont largement dépendant du contenu de la page. En effet, mis à part des objets tels que *navigator* qui sont figés pour un utilisateur donné, le contenu des autres objets variera suivant le contenu de la page, car suivant la page les objets présents dans celles-ci (sous-objets des objets décrits précédemment) sont différents. voyons voir comment ceux-ci sont organisés

Les objets du navigateur sont classés hiérarchiquement

Les objets du navigateur sont classés dans une hiérarchie qui décrit la page affichée à l'écran, et qui permet d'accéder à n'importe quel objet grâce à une désignation dépendant de la hiérarchie (on part du sommet puis on descend l'arborescence).
Dans cette hiérarchie, les descendants d'un objet sont des propriétés de ces objets mais peuvent aussi être des objets qui contiennent eux même des propriétés...

Voyons voir à quoi ressemble cette hérarchie:

Niveau1	Niveau2	Niveau3	Commentaire
navigator			Informations sur le browser utilisé
window			Gestion de la fenêtre d'affichage
	parent, frames, self, top		Désignation de la sous-fenêtre
	location		Informations sur l'emplacement de la page
	history		Accès à l'historique (sites précédemment visités)
	document		Informations sur le contenu de la fenêtre (éléments qui composent la page)
		images	Référence des images présentes dans la page
		forms	Référence des formulaires présents dans la page
		links	Référence des liens présents dans la page
		anchors	Référence des ancrages présents dans la page

Comment accéder à un objet?

Pour accéder à un objet du navigateur, il faut parcourir la hiérarchie du navigateur, en partant du sommet (l'objet *window*), puis en parcourant tous les maillons jusqu'à atteindre l'objet désiré. La syntaxe est *window.objet1.objet2.objet3.objet_vise* (ici il y a trois objets intermédiaire *objet1 objet2 objet3* mais ce nombre peut varier de 0 à un très grand nombre d'objets, suivant l'embrication de vos objets dans la page...).

Pour accéder modifier une propriété de l'objet visé il suffit de rajouter un point, puis le nom de la propriété. Certaines propriétés sont modifiables, c'est-à-dire que dynamiquement on va pouvoir modifier un élément (du texte, une image, ...). Certaines propriétés sont par contre en lecture seule, c'est-à-dire qu'elles permettent uniquement de récupérer des informations mais qu'on ne peut pas les modifier...

J) L'objet Window

L'objet window est l'objet par exellence dans Javascript, car il est le parent de chaque obet qui compose la page web, il contient donc:

- l'objet document: la page en elle-même
- l'objet location: le lieu de stockage de la page
- l'objet history: les pages visitées précédemment
- l'objet frames: les cadres (division de la fenêtre en sous-fenêtres)

Les méthodes de l'objet window

L'objet *window* possède des méthodes relatives à l'ouverture et à la fermeture des fenêtres.

Les méthodes alert(), confirm() et prompt()

Les méthodes *alert()*, *confirm()* et *prompt()* sont des méthodes qui font apparaître une boîte de dialogue, elles sont expliquées en détail dans le chapitre Boîte de dialogue.

Les méthodes open(), et close()

Les méthodes *open()* et *close()* sont bien évidemment destinées à permettre l'ouverture et la fermeture de fenêtres. Toutefois la syntaxe de la méthode *open()* est longue car elle admet un nombre important de paramètres:

La méthode *open()* permet d'ouvrir une fenêtre, voici sa syntaxe:

```
window.open("URL","nom_de_la_fenetre","options_de_la_fenetre");
```

URL désigne l'url de la page qui sera affichée dans la nouvelle fenêtre, c'est-à-dire l'emplacement physique de celle-ci.

Les options de la fenêtre sont les suivantes:

option	**description**
directory = *yes/no*	Affiche ou non les boutons de navigation
location = *yes/no*	Affiche ou non la barre d'adresse
menubar = *yes/no*	Affiche ou non la barre de menu (fichier, edition, ...)
resizable = *yes/no*	Définit si la taille de la fenêtre est modifiable ou non
scrollbars = *yes/no*	Affiche ou non les ascenceurs (barres de défilement)
status = *yes/no*	Affiche ou non la barre d'état
toolbar = *yes/no*	Affiche ou non la barre d'outils
width = largeur (en pixels)	Définit la largeur
height = hauteur (en pixels)	Définit la hauteur

Ainsi, il est possible d'utiliser cette méthode avec n'importe quel gestionnaire d'événement, directement dans le code à exécuter ou bien dans une fonction.

- les options doivent être saisies les unes après les autres, séparées par des virgules, sans espace
- l'ensemble des options doit être encadré par les guillemets

Chacune des fenêtres doit cependant être fermée, il faut donc se servir de la méthode *close()* qui permet de fermer une fenêtre.

La méthode *close()* requiert le nom de la fenêtre comme argument, il suffit donc de créer un bouton (image, hypertexte, ou bouton de formulaire) qui permettra de fermer cette fenêtre.

Pour un lien hypertexte, le code sera:

```
<A href="javascript:self.close('nom_de_la_fenetre_');">Cliquez ici pour fermer la fenêtre</A>
```

Pour un bouton (image), le code sera:

```
<A href="javascript:self.close('nom_de_la_fenetre_');"><img src="nom_de_l_image"></A>
```

Il est bien évidemment possible d'utiliser cette procédure avec tous les gestionnaires d'événement, en utilisant par exemple une syntaxe proche de celle-ci:

```
<Ahref="javascript:;"onMouseOver="self.close('nom_de_la_fenetre_');">
<img src="nom_de_l_image"></A>
```

K) Listings Divers

Listing 1: Le script An 2000 avec le calcul des minutes

```
l:  <HTML>
2:  <HEAD><TITLE>An 2000</TITLE></HEAD>
3:  <BODY>
4:  <H1>Compte à rebours jusqu'à Year 2000</H1>
5:  <HR>
6:  <SCRIPT LANGUAGE="JavaScript1.1">
7:  maintenant = new Date();
8:  an2000 = new Date("Jan 01 00:00:00 2000");
9:  secondes = (an2000 - maintenant) / 1000;
10: secondes = Math.round(secondes);
11: document.write("Secondes jusqu'à l'an 2000: " + seconds)
12: minutes - secondes / 60;
13: minutes = Math.round(minutes);
14: document.write("<P>Minutes jusqu'à l'an 2000: " + minutes);
15: </SCRIPT>
16: </BODY>
17: </HTML>
```

Listing 2: Un exemple complet d'utilisation de fonction

```
1:  <HTML>
2:  <HEAD>
3:  <TITLE>Fonctions</TITLE>
4:  <SCRIPT LANGUAGE="JavaScript">
5:    function Bonjour(untel) {
6:      window.alert("Bien le bonjour, " + untel);
7:    }
8:  </SCRIPT>
9:  </HEAD>
10: <BODY>
11: <hl>Exemple de fonction</h1>
12: <P>Vous allez étre salué deux fois de suite.</P>
13: <SGRIPT LANGUAGE="JavaScript">
14:   Bonjour("Anna");
15:   Bonjour("Fred");
16: </SCRIPT>
17: </BODY>
18: </HTML>
```

Listing 3: Une instruction if déclenchant une série d'actions

```
1: if (a == 1) {
2:   window.alert("a est égal à 1.");
3: }
4: else if (a == 2) {
5:   window.alert("a est égale à 2.");
6: }
7: else {
8:   window.alert("a est différent de 1 et 2.");
9: }
```

Listing 4: Le script de test des réponses de l'utilisateur

```
 1: <HTML>
 2: <HEAD><TITLE>R&eacute;action &agrave; la r&eacute;ponse de
    l'utilisateur </TITLE>
 3: </HEAD> 4: <BODY>
 5: <H1>R&eacute;action &agrave; la r&eacute;ponse de l'utilisateur</H1>
```

```
 6: <P>Entrez votre destination</P>
 7: <SCRIPT LANGUAGE="JavaScript1.2">
 8:   destination = prompt("Où voulez-vous aller ?");
 9:   switch (destination) {
10:     case "Netscape" :
11:       window.location="http://www.netscape.fr";
12:       break;
13:     case "Microsoft" :
14:       window.location="http://www.microsoft.fr";
15:       break;
16:     case "Yahoo" :
17:       window.location="http://www.yahoo.fr";
18:       break;
19:     default :
20:       window.location="http://www.campuspress.fr";
21:   }
22: </SCRIPT>
23: </BODY>
24: </HTML>
```

Listing 5: Script permettant de connaître les propriétés des navigateurs

```
<SCRIPT LANGUAGE="JavaScript">
      j=1;
      for(i in navigator){
            document.write("Property   nb   "+   j   +"   :   "+i   +   "   est
      "+navigator[i]);
            document.write("<br>");
            j++;
      }
</SCRIPT>
```

L) Exercices

Exercice 1 :
Calcul d'une moyenne de 4 nombres : afficher les quatre valeurs sur la page ainsi que le résultat du calcul.

Exercice 2 :
Demander à l'utilisateur d'entrer dix noms via la méthode window.prompt(), stocker ces valeurs dans un tableau, afficher ce tableau sur la page, trier le tableau et afficher sur la page le tableau trié.

Exercice 3 :
A l'aide de Dreamweaver, créer un formulaire comportant plusieurs champs (textes et numériques), des boutons radio, des checkbox et un bouton de validation. Vous enverrez ce formulaire à l'une de vos adresses e-mail. Le but de l'exercice est de vérifier que tous les champs ont été remplis, qu'il n'y a pas de lettres dans les champs numériques, etc. Pour vous aider, utilisez le formulaire de contact que vous avez fait sur votre site web. Si vous n'en avez pas, faites-en un.

Exrecice 4 :
Sur votre site web, créez une page que vous nommerez "galerie" sur laquelle vous placerez différentes images (leurs vignettes = les thumbnails). Vous écrirez ensuite un script qui, lorsque vous cliquez sur une des vignettes, fasse apparaître l'image originale dans une nouvelle fenêtre comportant un bouton permettant de la fermer.

M) Supplément - Les COOKIES

Les Cookies.

Nous allons voir ici à quoi servent les cookies et comment stocker et récupérer des informations grâce à eux.
En annexe de ce document, vous trouverez un exemple complet.

1. A quoi servent les cookies ?

Les cookies sont très utilisés, par tous les sites commerciaux et par de plus en plus de sites personnels. La raison est simple. Un cookie permet de stocker de manière permanente des informations sur le poste du visiteur, informations qui pourront être récupérées lors des futures visites.
Voyons quelques unes des principales informations stockées dans les cookies :
Le nombre de visites, la date de la dernière visite,...
Un identifiant et un mot de passe pour une reconnaissance automatique du visiteur
Une liste de mots-clés utilisés dans les moteurs de recherche pour cibler les publicités à afficher (Exemple : beaucoup de moteurs de recherche)
Une liste de paramètres de préférences de navigation pour personnaliser la page présentée.
Des informations à transférer d'une page à l'autre du site.

2. Comment STOCKER des informations ?

Le type d'informations à stocker n'a aucune influence sur le code. Un cookie est un fichier de texte qui contient une chaîne de caractères contenant les informations concaténées.
Voici la fonction qui permet de stocker une information dans un cookie

```
function SetCookie (name, value)
{
      var argv=SetCookie.arguments;
      var argc=SetCookie.arguments.length;
      var expires=(argc > 2) ? argv[2] : null;
      var path=(argc > 3) ? argv[3] : null;
      var domain=(argc > 4) ? argv[4] : null;
      var secure=(argc > 5) ? argv[5] : false;
      document.cookie=name+"="+escape(value)+((expires==null)? "" : (";
      expires="+expires.toGMTString()))+ ((path==null) ? "" : ("; path="+path))+
      ((domain==null) ? "" : ("; domain="+domain))+ ((secure==true) ? "; secure" : "");
 }
```

La valeur à stocker est associée à un nom de cookie.
La date d'expiration permet de définir la longévité du cookie. Si elle est omise, le cookie est détruit à la fermeture du navigateur.
Le path indique simplement d'où vient le cookie.
Le nom de domaine permet d'identifier le cookie parmi tous ceux stockés sur la machine.
La variable secure indique si l'accès au cookie est protégé.

Donc pour stocker la valeur Arthur dans la variable prenom il suffit d'appeler la fonction comme ceci :

```
var pathname=location.pathname;
var myDomain=pathname.substring(0,pathname.lastIndexOf('/')) +'/';
var date_exp = new Date();
date_exp.setTime(date_exp.getTime()+(365*24*3600*1000));
// Ici on définit une durée de vie de 365 jours
SetCookie("prenom","Arthur",date_exp,myDomain);
```

3 - Comment RECUPERER les informations ?

Voyons un cookie qui contiendrait les informations suivantes :

```
prenom = Arthur
nb_visite = 3
```

Pour récupérer l'information prenom, il faut extraire de la chaîne de caractères composant le cookie, le nom de la variable soit prenom. La valeur de prenom est la chaîne de caractères située immédiatement après et séparée par ';' et par ' '.

Voici le code des fonctions nécessaires à la récupération d'une information :

```
function getCookieVal(offset)
{
      var endstr=document.cookie.indexOf (";", offset);
      if (endstr==-1)
        endstr=document.cookie.length;
      return unescape(document.cookie.substring(offset, endstr));
}

function GetCookie (name)
{
        var arg=name+"=";
        var alen=arg.length;
        var clen=document.cookie.length;
        var i=0;
        while (i<clen)
        {
                var j=i+alen;
                if (document.cookie.substring(i, j)==arg)
                return getCookieVal (j);
                i=document.cookie.indexOf(" ",i)+1;
                if (i==0) break;
        }
        return null;
}
```

Si la variable demandée n'est pas contenue dans le cookie, elle est considérée comme valant null. Pour récupérer la variable prenom, il suffit d'appeler la fonction :

```
le_prenom=GetCookie("prenom");
```

Maintenant, vous savez tout. Il ne reste plus qu'à faire preuve d'imagination. La gestion des cookies est toujours la même, mais leur utilisation est illimitée :-)

Pour plus d'informations : http://www.netscape.com/newsref/std/cookie_spec.html

```
<html>
<head>
<title>Utilisation d'un cookie avec JavaScript</title>

<script language="JavaScript">

// Déclaration des variables 'domaine' et 'date d'expiration'
var pathname=location.pathname;
var myDomain=pathname.substring(0,pathname.lastIndexOf('/')) +'/';
var date_exp = new Date();
date_exp.setTime(date_exp.getTime()+(365*24*3600*1000));

// Voici les 3 fonctions de gestions des cookies
function getCookieVal(offset)
{
        var endstr=document.cookie.indexOf (";", offset);
        if (endstr==-1)
                endstr=document.cookie.length;
        return unescape(document.cookie.substring(offset, endstr));
}
function GetCookie (name)
```

```
{
	var arg=name+"=";
	var alen=arg.length;
	var clen=document.cookie.length;
	var i=0;
	while (i<clen)
	{
		var j=i+alen;
		if (document.cookie.substring(i, j)==arg)
			return getCookieVal (j);
			i=document.cookie.indexOf(" ",i)+1;
			if (i==0) break;
	}
	return null;
}
function SetCookie (name, value)
{
// un cookie a besoin d'un nom, d'une valeur, d'un nom de domaine, d'une date d'expiration

	var argv=SetCookie.arguments;
	var argc=SetCookie.arguments.length;
	var expires=(argc > 2) ? argv[2] : null;
	var path=(argc > 3) ? argv[3] : null;
	var domain=(argc > 4) ? argv[4] : null;
	var secure=(argc > 5) ? argv[5] : false;
	document.cookie=name+"="+escape(value)+
		((expires==null) ? "" : ("; expires="+expires.toGMTString()))+
		((path==null) ? "" : ("; path="+path))+
		((domain==null) ? "" : ("; domain="+domain))+
		((secure==true) ? "; secure" : "");
}

function disp(txt)
{
	document.write(txt);
}



function init()
{
	var nb=GetCookie("nb");
	var nom=GetCookie("nom");
}
function sto()
{
//	 Fonction appelée par le bouton "Stocker une information"
//	   Le nom de l'information est précédée de "_" pour ne pas interférer avec les noms
utilisés par le site JScript.
	var nom=document.forms[0].elements[0].value;
	var valeur=document.forms[0].elements[1].value;
	if (nom!="")
	{
			if (valeur!="")
			{
				SetCookie("_"+nom,valeur,date_exp,myDomain);
			}
	}
	else alert("Il n'y a pas de nom !");
	document.forms[0].elements[0].value="";
	document.forms[0].elements[1].value="";
}
function get()
{
// Fonction appelée par le bouton "Récupérer une information"
// Le nom de l'information est précédée de "_" pour ne pas interférer avec les noms utilisés
par le site JScript.
	var nom=document.forms[0].elements[3].value;
	if (nom!="")
	{
		var valeur=GetCookie("_"+nom);
		if (valeur!=null)
		{
			document.forms[0].elements[4].value=valeur;
		}
		else document.forms[0].elements[4].value="null";
```

```
        }
        else document.forms[0].elements[4].value="";
}
</script>
</head>

<body bgcolor="#CCCCCC">

<form method="post" action="">

<table width=500 border=1>
        <tr>
                <td colspan=2><b>Enregistrer une information</b></td>
        </tr>

        <tr>
                <td width=300>Entrez le nom de l'information</td>
                <td width=200><input type="text"></td>
        </tr>

        <tr>
                <td>Entrez la valeur de cette information</td>
                <td><input type="text"></td>
        </tr>

        <tr>
        <td colspan=2><input type="button" value="Stocker l'information" onClick="sto()"></td>
        </tr>
</table>

<p>
<hr>
<p>

<table width=500 border=1>
        <tr>
                <td colspan=2><b>Récupérez  une information</b></td>
        </tr>
        <tr>
                <td width=300>Entrez le nom de l'information</td>
                <td width=200><input type="text"></td>
        </tr>
        <tr>
                <td>L'information vaut</td>
                <td><input type="text"></td>
        </tr>
        <tr>
        <td      colspan=2><input      type="button"      value="Retourner      l'information"
onClick="get()"></td>
        </tr>
</table>
<p>
<input type=reset value="effacer les valeurs">

</form>
</body>
</html>
```