Javascript
N7 Cours

Michel Cabaré
Décembre 2000

TABLE DES MATIERES

QU'EST-CE JAVASCRIPT

Présentation

L'emploi de **javascript** est devenu aujourd'hui indispensable pour qui veut créer un site Web convivial et performant.

Auparavant, la majorité des traitements effectués dans les documents HTML étaient réalisés par l'action du serveur et le poste client était très passif, puisque celui-ci se bornait quasiment à afficher les résultats des requêtes que lui soumettait le dit serveur ! En effet, un programme ou un script CGI (Common Gateway Interface) était exécuté par le serveur qui transmettait le résultat au poste client, celui-ci interprétait l'envoi HTML et l'affichait par le navigateur. Cette situation surchargeait énormément les serveurs et le réseau...

Aujourd'hui, grâce au Javascript (introduit par Netscape 2.0), et encore plus récemment avec le Dynamic HTML (introduit par Microsoft), cette situation a complètement changé et le navigateur est maintenant capable "d'intelligence" puisqu'il est à même de traiter des scripts embarqués coté «client », de manière indépendante au serveur (réseau non utilisé), par un traitement local.

Il est vrai que le langage JAVA et aussi capable et même plus que le Javascript, mais celui-ci est réservé à des spécialistes. Il permet beaucoup plus mais est beaucoup plus lourd, car il nécessite une bonne connaissance de la programmation (Objet), un compilateur (il est compilé à l'inverse du Javascript), un débogueur etc.

Le couple HTML-Javascript est très souple, accessible à la plupart des Webmasters et sa popularité ne cesse de grandir car il permet un réelle autonomie au poste client, même s'il ne peut accéder au fichiers du poste client sur lequel il s'exécute !

Il ne s'agit pas ici vous donner une approche complète du langage Javascript (un livre entier serait nécessaire) mais de vous aider à en découvrir les notions de bases, pour ensuite les appliquer de manière très pratique

Objectifs

Javascript est un langage de script bien adapté àla création de petits programmes simples, comme des convertisseurs d'unité, des calculettes simples, du contrôle de formulaire, de l'animation de page...

Javascript n'est pas adapté pour mettre en place un affichage ou de la manipulation de donnée, mais permet d'interagir avec l'utilisateur de manière beaucoup plus souple que les langages plus complet tels que JAVA

Déclenchement par évènements

De par comment il est intégré aux navigateurs et au code HTML, Javascript peut programmer des réponses àdes évènements utilisateurs tels qu'un clic souris ou la saisie de donnée dans un champs de formulaire, faire apparaître de l'aide dans la barre d'état lorsque le pointeur de la souris passe sur une partie de la page,

Orienté Objet

Javascript intègre une forme limité de modèle orienté objet (voir plus loin) chapitre

Les plus et .Les moins :

Plus : interprété, la phase test-essais ne nécessite pas de compilation et est dons très rapide

Plus : indépendant de la plateforme utilisée

Plus : taille réduite et vitesse d'exécution élevée

Moins : code source non dissimulable

Moins : faible possibilités de débogage et fonctionnalités relativement limitées

Moins : nécessite une version relativement récente des navigateur

Moins : interprété, les nuances d'interprétation d'un navigateur àl'autre, voire d'un version àune autre peuvent être considérables

APPEL D'UN SCRIPT JAVASCRIPT

Les différents appels

Pour qu'un document HTML puisse exécuter des traitements par le biais de scripts Javascript, il faut en premier lieu que celui-ci comprenne les scripts écrits dans ce langage. En effet, le Javascript est un langage de scripts et le navigateur interprète ceux-ci lors du chargement de la page HTML :

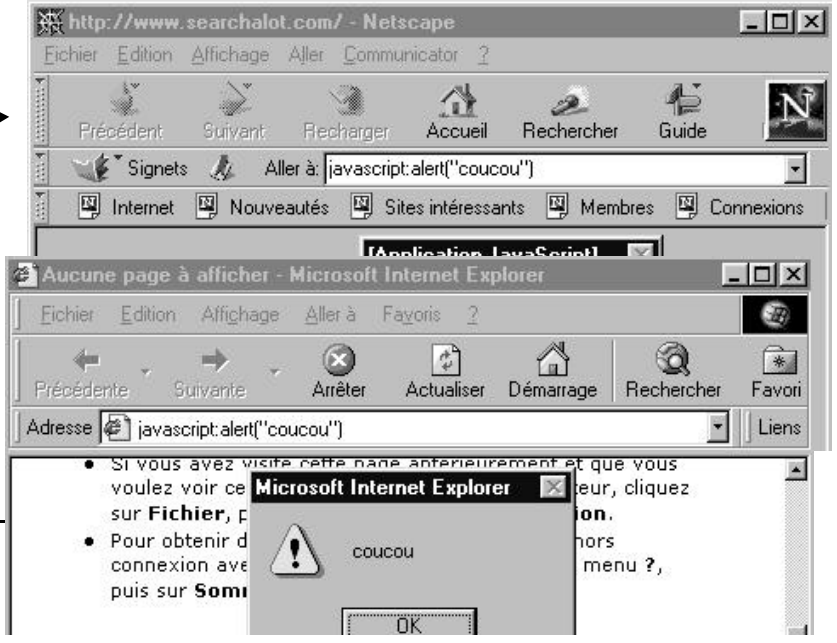
Nous allons le voir, plusieurs manières peuvent être employées pour l'insertion d'un script Javascript dans un document HTML :

- Dans le **Navigateur** : le script ou l'appel du script se faisant directement dans l'URL du navigateur
- Par un **Tag spécifique** : incorporant le script lui-même ou faisant appel au source du script
- Par **modification d'un Tag** : utilisant du javascript standard prédéfini ou appelant une fonction définie auparavant manuellement en javascript

Dans le navigateur :

La première manière, pour exécuter un script peut être de taper directement l'appel dans la zone de saisie de l'URL dans le navigateur utilisé, (voire de taper directement l'instruction javascript) : il suffit de le faire précéder du mot réservé **javascript:**

Appel dans Nestcape →



Appel dans

Explorer ⟶

On voit tout de suite que le comportement du navigateur face àune instruction Javascript peut varier (ici au niveau de la barre de titre de la fenêtre qui s'affiche)

DONC DE MANIERE CONSTANTE, UN SCRIPT DEVRA ETRE TESTE SOUS LES PRINCIPAUX TYPE DE NAVIGATEUR QUE LE CONCEPTEUR ESTIME DEVOIR "ACCEPTER"

Tag spécifique

plus classiquement, on insère un script àl'aide d'une balise spécifique <SCRIPT> qui doit être insérée dans le document HTML même.

TAG	**<SCRIPT> </SCRIPT>**
Paramètre	**LANGUAGE="JavaScript"** Précise quel est le langage du script (JavaScript, VBScript...) avec sa version éventuelle 1.2 par exemple
Paramètre	**SRC** précise l'URL du script qui doit être inséré. Cet attribut n'est employé que dans le cas où le contenu du script n'est pas inséré dans le document HTML lui-même mais dans un fichier d'extension ".js".

Par exemple :

```
SCRIPT LANGUAGE="JavaScript">
...contenu du script...
</SCRIPT>
```

Ceci peut être complété, pour raison de compatibilité avec les navigateurs de version inférieure, par l'adjonction des balises de commentaires HTML, qui permettront ainsi aux navigateurs plus anciens d'ignorer le contenu du script.

Cet ajout n'est pas obligatoire pour l'exécution du script cependant, il est recommandé de l'employer, lui-même agrémenté éventuellement par les mots (Début du script et Fin de script) si vous voulez rendre la lecture du code très lisible.

Script dans une page HTML:

Voir le fichier **COURSJS01.HTM** →

```
<BODY>
<script LANGUAGE="JavaScript">
<!--
                alert (' bonjours')
//-->
</script>
exemple de script minimaliste, mais existant !
</BODY>
```

Selon que vous voulez utiliser une version ou une autre du JavaScript, vous devrez mettre des balises différentes.

Version	Compatibilité	Balise
JavaScript 1.0	I.Explorer 3.0 - Netscape 2.0	<SCRIPT LANGUAGE="JavaScript1.0">
JavaScript 1.1	Netscape 3.0x , 4.0 et 4.x	<SCRIPT LANGUAGE="JavaScript1.1">
JavaScript 1.2	I.Explorer 4.x Netscape 4.0x à4.05	<SCRIPT LANGUAGE="JavaScript1.2">
JavaScript 1.3	I.Explorer 5.x Netscape 4.06 à4.74	<SCRIPT LANGUAGE="JavaScript1.3">

Script dans un fichier .js:

par exemple →

```
<script Language="JavaScript1.2" SRC="coursjs02.js"></SCRIPT>
```

Voir le fichier **COURSJS02.HTM** et le fichier **COURSJS02.JS**

avec dans le fichier :

```
// début du script -->
...contenu du script...
// Fin du script -->
```

ATTENTION : mettre le script en SRC "dévalide" la vérification de version possible via l'instruction Language, (au moins sur netscape 3.x)

Tag pré existant

Une autre manière pour insérer un script se fait par l'ajout d'un code spécifique dans la balise d'un élément HTML, il s'agira soir d'un appel direct soit d'un appel de script écris au préalable

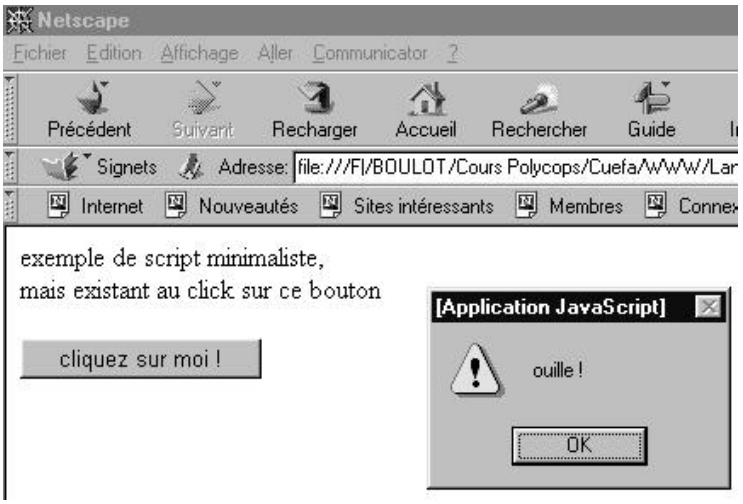
Appel direct

Il est possible d'employer une fonction intégrée du Javascript par le seul ajout de ce code dans la balise de l'élément mais cela restera très limité.

Par exemple :

Voir le fichier **COURSJS03.HTM**

```
<BODY>
exemple de script minimaliste, <BR>
mais existant au click sur ce bouton
<form>
<input type="button" value="cliquez sur moi !" onClick="alert('ouille !')" >
</form>
</BODY>
```



Ce code provoque l'apparition d'une fenêtre (alerte) par le clic (l'événement "onClick") sur le bouton du formulaire... Cette fenêtre spéciale, "alerte", est une fenêtre de dialogue dans laquelle une chaîne de caractères (message) est affichée. C'est une méthode de l'objet window. Pour cet exemple, il n'est donc pas nécessaire d'insérer un script dans le document

Appel de fonction

Mieux vaut l'ajout d'un appel de fonction spécifique dans la balise de l'élément qui lancera le traitement. Ce code servira très souvent àlancer une fonction définie dans

un script "conventionnel" c'est-àdire, inséré par les balises conventionnelles (<SCRIPT>) un peu plus haut.

Par exemple :

Voir le fichier **COURSJS04.HTM**

```
<script language="JavaScript">
<!--
function hurle()
{
 alert("ouille !, mais après tout aussi merci de m\'avoir donné l\'occasion de m\'exprimer")
}
//-->
</script>
</HEAD>
```

ce qui permet de définir au préalable un script parfois important sous le nom **hurle()**

suivit plus loin de

```
<BODY>
exemple de script dans une fonction, <BR>
mais existant au click sur ce bouton
<form>
<input type="button" value="cliquez sur moi !" onClick="hurle()" >
</form>
</BODY>
</HTML>
```

ce qui permet d'appeler le script défini plus haut par son nom **hurle()**

N.B: Vous avez remarqué un caractère "\" (barre oblique inverse), appelé caractère d'échappement, placé avant l'apostrophe placé devant le mot "image" ! En fait, l'apostrophe est un caractère qui fait partie de la syntaxe du code Javascript (comme les guillemets) et pour indiquer que cet apostrophe particulier ne doit pas être interprété comme tel, il faut placer ce caractère ("\") avant l'apostrophe (ou les guillemets)

Tag <NOSCRIPT>

Si l'utilisateur qui visionne vos documents HTML, comprenant des scripts n'a pas activé l'option appropriée, il est possible de lui donner un message particulier. En effet, tout texte inséré avec la balise spécifique <NOSCRIPT> verra œlui-ci affiché tandis que le navigateur de ceux pour qui l'option est activée ignorera le texte...

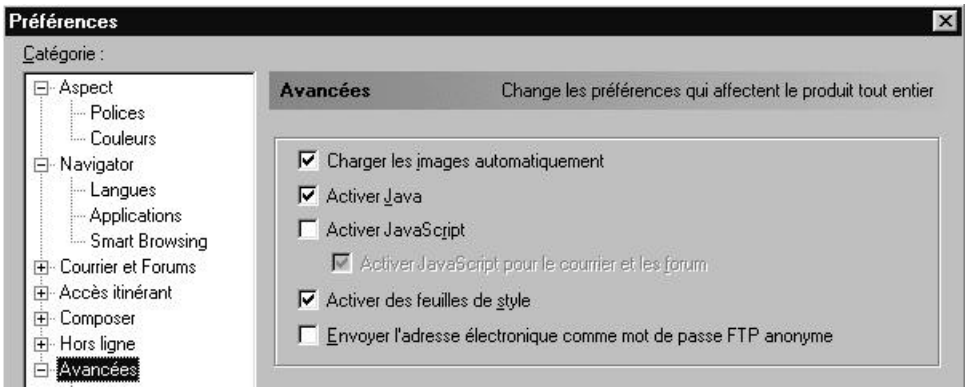
Essayer de modifier le fichier avec l'ajout de

```
<NOSCRIPT>
Javascript ou l'option appropriée non activée...
</NOSCRIPT>
```

Essayez de désactiver dans un navigateur l'interprétation Javascript pour visualiser ce tag…(solution en **COURSJS04B.HTM**)

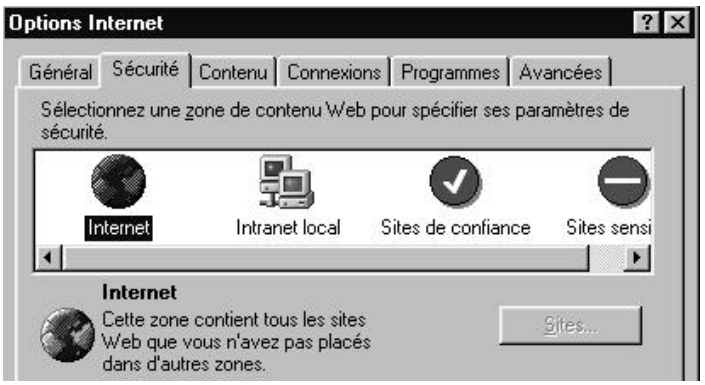
Dans Netscape on demandera le menu :

Edition / Préférences / Avancées

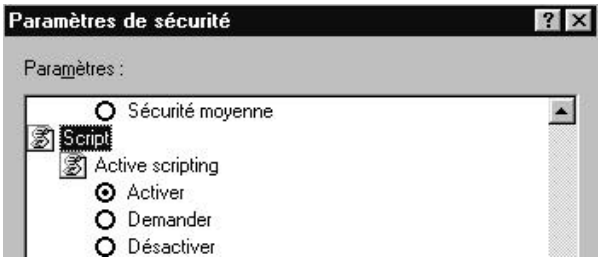


Dans Explorer cela dépends des zones de sécurité que l'on a mis en place (mais ne peut s'appliquer au fichier téléchargés localement…)

OUTILS / OPTIONS INTERNET / SECURITE



dans laquelle il faut demander de personnaliser le niveau



UN PEU DE SYNTAXE

Min-maj :

Html n'est pas regardant sur les problèmes de minuscule ou de majuscule, il lui suffit de retrouver la bonne "syntaxe", quelle que soit la casse

Javascript est sensible àla casse et fera donc clairement la différence entre

Toto, toto ou TOTO

Commentaires :

Les commentaire HTML débutent par

<!-- et se terminent par **-->**

Les commentaires Javascript existent de deux sortes

- ***//*** invalide la fin de ligne restante
- ***/**** ouvre un commentaire qui ira jusqu'àce que soit rencontré ****/***

Expression :

Toute expression se termine par un **;**

Certain navigateurs acceptent un retour àla ligne

Bloc :

On peut regrouper plusieurs expressions entre elles pour créer des "blocs"

```
{       expression 1;
        expression 2;
}
```

LES VARIABLES

Déclaration :

Pour représenter, manipuler, stocker ou modifier des valeurs, fournies dans un code Javascript, comme dans tout langage de programmation, il faut utiliser des variables.

En javascript, la déclaration des variables est très simple car très peu différenciée. Il n'est pas obligatoire, mais recommandé, de déclarer des variables par le mot clé "var".

L'identificateur de variable doit, pour être correct :

- être une chaîne de caractères
- commencer par une lettre de l'alphabet ou le caractère de soulignement ("_") (Le reste de la chaîne peut comporter des chiffres)

```
var adresse = "Rue Dupont"
var numero = 12
var _ex
var y
```

Dans ces exemples, il y a deux variables (**adresse** et **numero**) auxquelles ont été attribuées directement des valeurs, et deux autres (**_ex** et **y**) sans valeur initiale.

Mots réservés :

Voici la liste des mots réservés. Ce sont des mots que vous ne pouvez pas utiliser comme nom de variable, de fonctions ou autre identificateur.

abstract	boolean	break	byte
case	catch	char	class
const	continue	default	do
double	else	extends	false
final	finally	float	for
function	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			

Variable locale-globale :

On parle de variable locale lorsque la variable a été déclarée dans une fonction (locale). Cette variable n'est connue que dans la fonction

```
<script LANGUAGE="JavaScript">
        function message()
        {var adresse = "Paris"
        return adresse}
</SCRIPT>
```

On parle de variable globale lorsque la variable a été déclarée en dehors d'une fonction. Cette variable est connue àpartir de l'endroit où elle est déclarée

```
<script LANGUAGE="JavaScript">
                var adresse = "Rue Dupont"
                function message()
                {var adresse = "Paris"; return adresse}
</SCRIPT>
```

Dans cet exemple, en **COURSJS05.HTM** il est pris expressément le même nom (**adresse**) pour les variables globale et locale

- valeur de la variable globale: "Rue Dupont",
- valeur de la variable locale : "Paris".

Quelle confusion…

On peut donner le même nom àune variable globale et locale mais ce n'est pas recommandé... !

```
<script LANGUAGE="JavaScript">
        -       var adresse = "Rue Dupont"
                function message()
                {var adresse = "Paris"; return adresse}
                document.write("Valeur de la var globale <BR>" +
adresse)
                document.write("Valeur de la var locale <BR>" +
message() )
</script>
```

LES TYPES DE VARIABLES

Principe :

On l'àdéjàdit, il n'est pas obligatoire, mais seulement recommandé, de déclarer des variables.

A ce titre il faut savoir que Javascript n'étant pas un langage compilé mais interprété, leur type sera éventuellement choisit selon le contexte voulut. Le typage étant assez faible, une variable déclarée dan un type peut être utilisée dans un autre type sans que un message d'erreur n'apparaisse. Parfois un navigateur "plantera" mais sans que l'on sache véritablement pourquoi...

Finalement, la déclaration de variable servira surtout... au programmeur pour qu'il puisse s'y retrouver !

Il existe cependant différents types de variables :

- les *Chaînes de caractère* ("Rue Dupont") ou STRING
- *les Nombres*
- les *Booléens*
- le *type àvaleur unique* null

Chaîne de caractère - string :

Le type de la variable va dépendre de ce que nous allons y mettre (on dit aussi "**stocker**") Le plus souvent, pour mettre quelque chose dans une variable, on utilise l'instruction suivante :

***MonPrenom* = "**michel**"**

En faisant cela, nous rangeons la valeur "michel" dans la variable ***MonPrenom***. On dit que l'on "**affecte**" la "**valeur**" michel àla variable ***MonPremon***.

Le signe "**=**" est l'instruction qui permet de dire àl'interpréteur JavaScript : range la valeur "michel" dans le casier appelé ***MonPrenom***.

La valeur "michel" étant mise entre guillemets, indique àl'interpréteur JavaScript que la valeur "michel" est de type "**String**" (chaîne de caractères).

Dorénavant, la variable ***MonPrenom***, tant qu'elle contiendra la valeur "michel" aura le type "**String**" (chaîne de caractères). Et ceci jusqu'àce qu'on décide de lui donner une valeur différente

Comme la plupart des langages, JS détermine par des apostrophes (**'**) ou des guillemets (**"**) le début et la fin de la Chaîne de caractères

Mais on ne peut mélanger les deux symboles

2 écritures correctes →

```
I = 'bonjours'
I = "bonjours"
```

écriture incorrecte →

```
I = 'bonjours"
```

Il existe des caractères spéciaux, qui peuvent être placés dans les constantes de chaîne de caractères (caractères entourés d'apostrophes ou de guillemets mais sans mélange) et qui permettent d'insérer :

Un passage àla ligne suivante	**\n**
Une tabulation horizontale	**\t**
Un retour en arrière	**\b**
Un retour chariot	**\r**
Un saut de page	**\f**

de même que pour faire apparaître certains caractères il faut préciser àl'interpréteur de ne pas … les interpréter

le caractère guillemet	**\"**
Le caractère apostrophe	**\'**
L'anti-slash	**\ **

Dans cet exemple, en **COURSJS06.HTM** notez uniquement l'utilisation du **\n**

Ainsi que l'imbrication des **'** et des **"** notez que si

'alert("Bonjour ")' et

"alert('Bonjour ')" sont

correct ni

"alert("Bonjour ")" ni

'alert('Bonjour ')' ne le

sont

Voici un exemple

```
<FORM>
        <input type="button"
        Value="Par exemple"
        onClick="alert('Bonjour \n et à bientôt')">
</FORM>
```

N.B: la règle générale étant quand même que tout ce qui est renvoyé par Javascript (ici Onclick) doit être entre " (guillemets)

on, écrira **onClick="alert('Bonjour \n et à bientôt')"**

plutôt que **onClick='alert("Bonjour \n et à bientôt")'**

Les nombres - number :

Si nous utilisons la variable appelée ***MonPoids*** et que nous lui affections la valeur 70 par l'instruction :

MonPoids = 70

Sans guillemets, la valeur 70 sera considérée comme un nombre. Nous dirons alors que la variable ***MonPoids*** est de type **Number** (nombre, ou numérique). Et elle le restera jusqu'à ce que nous décidions de lui affecter une autre valeur à notre convenance.

N.B: si l'on avait écrit : MonPoids = "70", le chiffre "70" entre guillemets, aurait signifié que "70" est une chaîne de caractères, et la variable aurait été de type **String**. Ce qui nous empêcherait par la suite d'effectuer des calculs sur le contenu de cette variable, puisque nous verrons que l'on ne peut effectuer d'opérations mathématiques sur les chaînes de caractères (variables de type String).

Il n'y a pas de différence en Javascript entre les nombres entiers et les réels, et en général toutes les opérations sur les nombres se font sur des réels

Particularité :

- Les nombres de base 8 (octale) doivent commencer par un 0 035137 : représentation octale du nombre décimal 14943 = (((*3**8 + *5*)*8 + *1*)*8 + *3*)*8 + *7*;
- Les nombres de base 16 (hexadécimale) commencent par 0x 0x3A5F : représentation hexadécimale du nombre décimal 14943 = ((3[*3*]*16+10[*A*])*16+5[*5*])*16+15[*F*]};
- Les autres nombres de base 10 (décimale) : 14943 : (((*1**10 + *4*)*10 + *9*)*10 + *4*)*10 + *3*

Correspondances entre **DÉCIMAL - HEXADÉCIMAL -** BINAIRE		
DÉCIMAL	HEXADÉCIMAL	BINAIRE
BASE - 10 -	BASE - 16 -	*BASE - 2 -*
0	0	0
1	1	1
2	2	10

3	3	11
4	4	100
5	5	101
6	6	110
7	7	111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Les booléens - boolean :

Voyons maintenant le cas d'un de type de variables moins évidents.

Imaginons une variable appelée ***JeMents***. et affectons lui la valeur true (vrai) par l'instruction :

***JeMents* = true**

La variable ne contient plus ni chiffre ni lettre, mais une valeur **logique**.

On dira alors que la variable est de type **Boolean** (booléen, ou logique). Il n'existe que deux valeurs **Boolean**, qui sont :

- "true" (vrai)
- "false" (faux)

Ces valeurs purement informatiques serviront à effectuer des tests, à vérifier si une situation est vraie ou fausse.

Les mots **true** et **false** sont des mots réservés JavaScript. On ne pourra donc pas les utiliser comme noms de variables.

Ne prennent que deux valeurs possibles, prises notamment quant au résultat des opérations de comparaison :

En cas de conversion forcée vers une valeur numérique, on aura alors

- "true" (vrai) donnant 1 (*)

- "false" (faux) donnant 0

(*) mais toute valeur différente de 0 dans une comparaison donnera true

PROCEDURES ET FONCTIONS

Définir et appeler des fonctions / procédures :

Les scripts placés dans le tag SCRIPT sont évalués àla fin du chargement de la page. Ces fonctions sont stockées, mais pas exécutées. Les fonctions, pour être exécutées doivent être appelées par un lien dans la page.

Il est important de faire la différence entre définir une fonction et appeler une fonction. Définir une fonction ne fait que la nommer, et dire quoi faire quand cette fonction est appelée. Appeler une fonction fait exécuter une fonction avec ses paramètres

La différence entre des fonctions ou des procédures se situe dans le fait qu'une fonction est un sous-programme (procédure) particulier qui a pour caractéristique de renvoyer une valeur, tandis qu'une procédure normale ne le fait pas.

Les fonctions devant être connues avant d'être utilisées, souvent on les définira dans la section d'en-tête de la page HTML (<HEAD> </HEAD>)

Aucune vérification de cohérence n'est faite sur les éventuels paramètres passés, ni sur leur nombre et encore moins sur leur type

Déclaration de procédure :

Toute procédure, pour être déclarée dans un script, doit posséder un **nom** et ensuite, un ou des paramètres: ces derniers sont optionnels

Ensuite, une accolade d'ouverture **{** et de fermeture **}** doivent englober l'ensemble des instructions.

```
function nomproc (paramètres1,...,paramètresN)
{
code de la procédure
}
```

Déclaration de fonction :

Toute procédure, pour être déclarée dans un script, doit posséder un nom et ensuite, un ou des paramètres: ces derniers sont optionnels

Ensuite, une accolade d'ouverture { et de fermeture } doivent englober l'ensemble des instructions.

Le renvoi se fait par le mot-clé : **return**.

```
function nomdfonct (paramètres1,...,paramètresN)
{
code de la fonction
return (resultat)
}
```

N.B: de manière générale, il est déconseillé de mettre plusieurs return à différents endroits dans une fonction, cela nuit énormément à la lisibilité. De plus certains navigateurs (netscape avant version 4.x) génèrent une erreur devant par exemple un **return**; suivit d'une instruction qui forcément ne s'exécutera jamais…

Appel de procédure :

Toute procédure, peut être utilisée dès qu'elle est connue

déclaration →

```
function proc1 (p1,p2)
{
code de la procédure
}
```

appel →

```
proc1(x,y)
```

Appel de fonction :

Toute fonction, peut être utilisée dès qu'elle est connue

déclaration →

```
function fonc1 (p1,p2)
{
code de la procédure
return(result)
}
```

appel avec stockage du résultat renvoyé →

```
a = fonc1(x,y)
```

Gestion des arguments :

De manière générale, dans une fonction il existe une variable prédéfinies **arguments** permettant de connaître la liste de tous les arguments passés en paramètre.

En essayant de travailler proprement, on peut alors écrire

```
Var     Argv = nom_fonction.arguments
```

la variable Argv un tableau contenant tous les paramètre

```
Var     Argc = Argv.length
```

la variable Argc contient le nombre d'arguments passés

En effet, si aucun argument n'est déclaré derrière **function**, la fonction garde tout de même l'accès àtous les arguments qui peuvent lui être transmis…

Deux cas peuvent se présenter :

- L'appel comporte **moins** de paramètres que la fonction ne le prévoit : le résultat sera forcément imprévisible quant àla valeur de ces paramètres
- L'appel comporte **plus** de paramètres que la fonction ne le prévoit : les valeurs sont ignorées, mais restent accessibles via **arguments**…

Voilàune déclaration de fonction **acceptant un nombre variable de paramètres**

COURSJS06B.HTM

```
function affiche()
    {
            var argv=affiche.arguments;
            var argc=argv.length;
            var cpt
            alert("vous avez passé " + argc + "arguments");
       for (cpt=1; cpt<=argc; cpt++)
            {
              alert("voila le " + cpt + "argument" + argv[(cpt-1)]);
            }
    }
```

et différents appels possibles

```
onClick = "affiche();">
```

```
onClick = "affiche(1,2);">
```

```
onClick = "affiche(1,'deux',3,'quatre',5);">
```

LES OPERATEURS

Arithmétiques :

n'ont de sens que s'ils sont utilisées sur des variables de type Number

- l'addition (**+**)
- la multiplication (*****)
- la soustraction(**-**)
- la division (***/***)
- le modulo (**%**) : reste de la division entière d'un entier par un autre

Concaténation :

l'opérateur de concaténation (**+**)

c'est exactement le même que celui utilisé pour l'addition simplement il porte sur des variables de type string

```
i = "bonjours" + 1
```

amène dans i la chaine "bonjours1"

N.B: Javascript convertit automatiquement les entiers en chaînes de caractère, ce qui vous permet de concaténer des entiers avec des chaînes, pour donner finalement une chaîne…

```
y = 50 + "Fr"
```

assigne àla variable y la chaîne "50 Fr"…

Comparaison :

En fonction du résultat du test de comparaison des opérandes, les opérateurs de relation donnent la valeur true ou false :

= =	True (vrai) dans le cas d'égalité des opérandes, false dans le cas inverse
!=	True (vrai) dans le cas d'inégalité des opérandes, false dans le cas inverse
>	true si l'opérande de gauche est supérieur à l'opérande de droite, false dans le cas inverse
>=	true si l'opérande de gauche est supérieur ou égal àl'opérande de droite, false dans le cas inverse
<	true si l'opérande de droite est supérieur à l'opérande de gauche, false dans le cas inverse
<=	true si l'opérande de droite est supérieur ou égal àl'opérande de gauche, false dans le cas inverse

Attention: 10 < "3" ou "10"< 2 devant des types différents est convertit en numérique (si possible).
Une comparaison de type dictionnaire ne se fera qu'entre des expressions de type caractère des deux cotés "xxx" < "yyy"

COURSJS07.HTM

```
<script LANGUAGE="JavaScript">
<!—
//  essayer "3"<10 puis 3<"10" et enfin "3"<"10" //
  if (3<10)
  alert("valeur true")
  else
  alert("valeur false")
//-->
</script>
```

3<10 **true**

"3"<10 **converti en 3<10** **true**

3<"10" **converti en 3<10** **true**

"3"<"10" **comparaison de chaîne** **false ("3" est >"10" comme "zoo" est plus loin dans le dictionnaire que "antirouille"...)**

Affectation :

certain "fanatiques" peuvent utiliser les conventions suivantes

Opération	Notation abrégée L'opérateur est mis devant le signe =
x = x + y	x += y
x = x - y	x -= y
x = x * y	x *= y
x = x / y	x /= y
x = x & 1	x &=b
x = x | b	x |= b
x = x ^ b	x ^= b
x = x << b	x <<= b
x = x >> b	x >>= b
x = x +1	++x ; x++
x = x - 1	--x ; x--

Le placement des opérateurs avant la variable (++a, --a) réalise une affectation antérieure à l'opération en cours. Le placement après réalise l'affectation après l'opération en cours.

Binaires :

Uniquement pour ceux que cela intéresse…

a = b & c	ET binaire entre les bits de b et de c (AND)
a = b | c	OU binaire entre les bits de b et de c (OR)
a = b ^ c	OU exclusif binaire entre les bits de b et de c (XOR)
a =!b	a = 0 si b est non nul, et a est non nul si b = 0 (NOT)
a = b << n	a est le décalage de n bits de b vers la gauche
a = b >> n	a est le décalage de n bits de b vers la droite
a >< b	les valeurs de a et b sont interchangées
a = ~b	complément binaire à 1, équivalent à (a ^ (-1)), inversion des bits

Logiques :

&&	ET logique (AND) donne true si les deux opérandes sont true et false dans le cas inverse
||	OU logique (OR) donne true si l'un des deux opérandes est true et false dans le cas inverse
!	la négation (NOT) donne l'inverse logique

Conditionnels :

Dans l'écriture

(condition) ? expression1 : expression2

Si la condition est vraie, dans ce cas, l'expression1 est réalisée, dans le cas inverse, c'est l'expression2 qui est réalisée.

Bien sûr il s'agit ici d'une forme simple des instructions conditionnelles classiques qui existent d'ailleurs en Javascript

N.B: **!0** [NOT 0] vaut n'importe quel nombre non nul, souvent -1 ou 1 selon les interpréteurs.

N.B: **false** est équivalent à**0**, et **true** àun **nombre non nul**.

N.B: Attention, le signe égal **=** d'affectation est différent du signe **==** de comparaison.

N.B: **null** n'est pas équivalent àun nombre nul (zéro) ou àla chaîne vide "" mais est égal à'non-affecté" ou vide

Noter l'exercice suivant en **COURSJS08.HTM**

"bonjour" s'affiche car :

- false =0
- true différent de 0

"Aurevoir" s'affiche car :

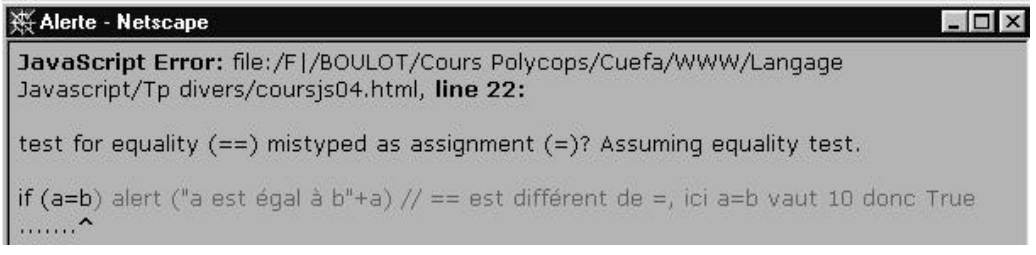
- true différent de 0

"non égal " s'affiche car compare 0 et 10

"égal" s'affiche avec Internet Explorer mais

"non égal" s'afficherait avec Netscape < 4.05 car il "corrige" l'erreur ?

```
<script LANGUAGE="JavaScript">
<!—
a = 122
if (a) alert("bonjour"); // TRUE est non nul

a = 0
if (!a) alert("Aurevoir "); // !0 vaut souvent 1 (TRUE)


b=10
if (a==b) alert ("égal")// comparaison classique
else
alert("non égal")


if (a=b) alert ("égal")              // == est différent de =, ici
else                                        //a=b vaudrait 10 donc True
alert("non égal")


alert("a vaut "+a+"et b vaut "+b)

var c
c="" // essayer de passer cette affectation en commentaire
if (c == null) alert ("la variable est vide et vaut "+c)
else
alert ("la variable est initialisée et vaut "+c)
//NULL n'est pas équivalent à un zéro, mais à "non-affecté",
//la chaine = ""; est une chaîne sans caractère mais initialisée
//-->
</script>
```



STRUCTURES DE CHOIX

if else :

Les mots clés pour un si sont **if** **else** avec l'indentation suivante

```
if (test)                    ou plus simplement    if (test)
  {...                                               {...
   ...                                               ...
      }                                                  }
else
      {...
       ...
      }
```

On utilise cette forme d'écriture de test, seulement si nous désirons que le programme fasse quelque chose de bien précis dans les deux cas. Aussi bien lorsque la condition est vraie que si elle est fausse. Nous pouvons recommencer à tester une autre condition dans le cas où la première n'est pas vraie. C'est ce que l'on appelle les "if imbriqués".	Dans cette forme d'écriture des tests, les actions sont exécutées seulement si la condition est vraie, et il ne se passe strictement rien dans le cas contraire, c'est-àdire si la condition n'est pas vraie (fausse). Le programme reprend simplement l'exécution des instructions qui suivent l'accolade fermante } après la fin du test, s'il y en a. C'est bien pratique, car cela allège l'écriture des tests.

Les imbrications sont possibles, et le **else** est apparié au dernier **if** n'ayant pas de **else**.

Les opérateurs logiques Et (&&) et OU (||) restent disponibles pour écrire des test plus complexes.

STRUCTURES DE REPETITION

While :

Crée une boucle qui répète l'exécution du code tant que la condition est vraie. Si la condition est vraie dès le début, le programme n'entre pas dans la boucle.

Syntaxe :

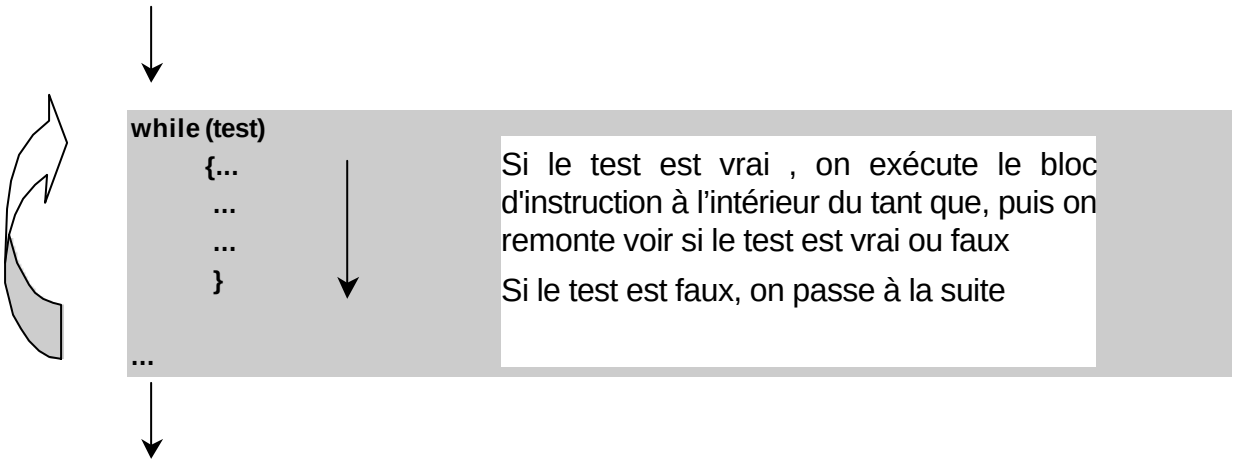
```
while (condition) {
code
}
```

Paramètre :

condition la condition de sortie de la boucle.

Fonctionnement :

Les mots clés pour un tant que sont **while** avec l'indentation suivante



N.B.: s'utilise quand àpriori on ne sait pas combien de fois on va "boucler". En général dans les instructions répétées il y en a forcément une qui àun moment donné fera passer le test = Vrai (Sinon cela risque de ne jamais s'arrêter.)

do while :

Crée une boucle qui répète l'exécution du code tant que la condition est vraie. àla différence du while, le programme entre toujours au moins une fois dans la boucle.

Syntaxe :

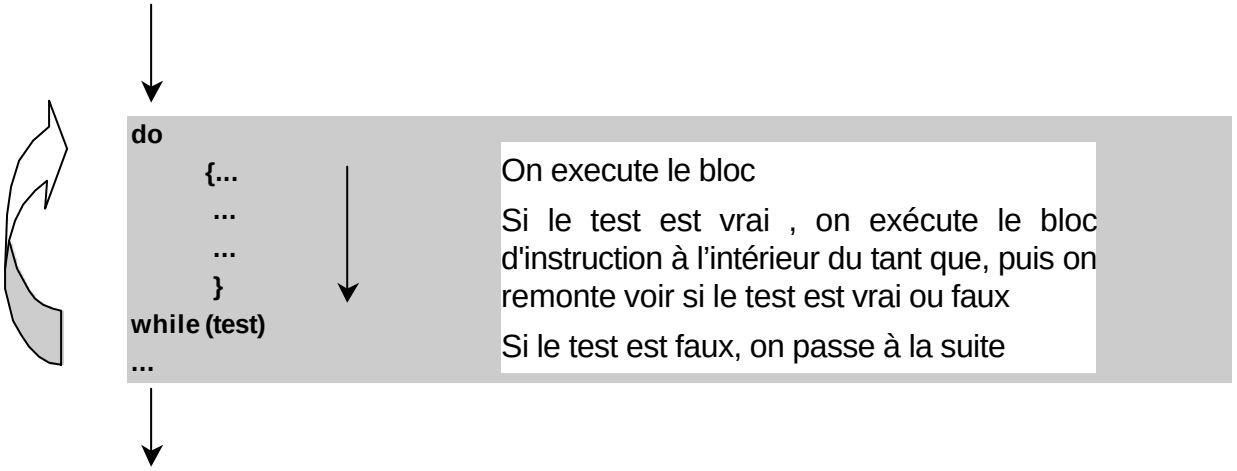
```
do{
code
}while (condition);
```

Paramètre :

condition la condition de sortie de la boucle.

Fonctionnement :

Les mots clés pour un tant que sont **do while** avec l'indentation suivante



Exemple :

```
i=0;
do{
i++;
document.write(i);
} while (i<10);
```

for :

Crée une boucle qui exécute le code x fois. Attention aux boucles infinies :

Syntaxe :

for ([valeur initiale] ; [condition] ; [incrément]) {

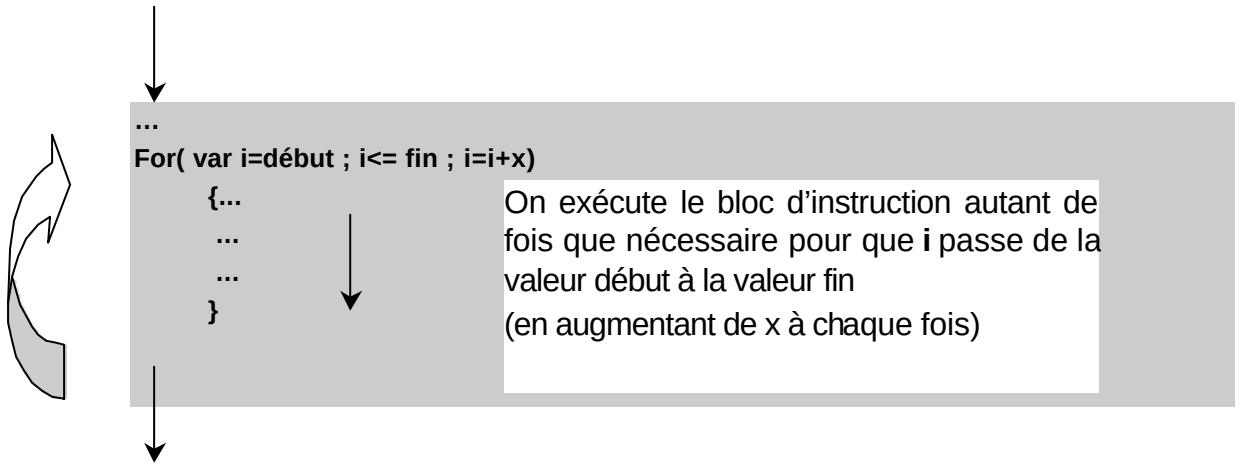
code

}

Paramètre :

- valeur initiale : valeur initiale de la variable compteur.
- condition : condition de sortie de la boucle.
- incrément : pour incrémenter ou décrémenter le compteur.

Fonctionnement :



N.B: s'utilise quand àpriori on sait combien de fois on va "boucler".

Ce qui se trouve entre les parenthèses se lit de la manière suivante :

i = début; veut dire : en partant avec la valeur **début** au compteur i

i <= fin; veut dire : tant que le compteur aura une valeur inférieure ou égale à**fin**;

i=i+1 veut dire : en incrémentant (en faisant grimper en français normal) le compteur i de **x** àchaque fois.

switch :

Permet au programme d'évaluer une expression et de la comparer àune valeur d'un "case ".

N.B: la version minimale nécessaire est Javascript 1.2 (c.a.d. Netscape 4.0x) ou Jscript 3.0 (c.a.d. Explorer 4.x)

Syntaxe :

```
switch (expression) {
case label :
code;
break;
case label :
code;
break;
...
default : code;
}
```

Le **break** est obligatoire afin de permettre àl'interpréteur de savoir où se trouve la fin du code d'un "**case**".

Exemple :

```
switch (langue) {
case "français" :
alert("Bonjour");
break;
case "english" :
alert("Hello");
break;
default : alert("langue inconnue");
}.
```

LES FENETRES DE DIALOGUE

Interface avec l'utilisateur :

Il existe plusieurs fonctions JavaScript pour faire apparaître des fenêtres et communiquer avec l'utilisateur. La méthode open() par exemple permet de créer une nouvelle fenêtre de navigateur, tandis que d'autres méthodes servent à faire afficher des fenêtres de dialogue.

Les fenêtres de dialogue sont utiles pour communiquer avec l'utilisateur. On peut simplement faire afficher un message ou encore demander une réponse de la part de l'usager.

Principaux types existants:

Méthode	Description
alert()	Affiche un message dans une fenêtre de dialogue comportant un bouton OK.
confirm()	Affiche une question dans une fenêtre de dialogue comportant les boutons OK et Annuler.
prompt()	Affiche un message dans une fenêtre de dialogue où celle-ci nécessite une réponse de la part de l'utilisateur.

Pour une compréhension plus complète des notions de méthode, d'objet se reporter au chapitre " **notion de P.O.O** " (page **Erreur ! Signet non défini.**)

alert()

La méthode **alert()** est surtout utile pour donner de l'information àl'utilisateur.

Affiche une boîte de message d'avertissement et un bouton OK. Cette méthode est utilisée pour avertir l'utilisateur seulement, il n'a aucune décision àprendre.

Même si la méthode alert utilise l'objet window, on n'a pas besoin de spécifier une fenêtre de référence lors de l'appel.

Syntaxe

alert("unMessage")

unMessage est une chaîne de caractères ou une propriété d'un objet existant.

Exemple

En voici un exemple:

```
<SCRIPT LANGUAGE="JavaScript">
<!-- Debut script
        alert("Bienvenue en formation");
//  Fin script -->
</SCRIPT>
```

confirm()

La méthode **confirm()** est utile lorsque l'on désire poser une question àl'utilisateur dont la réponse est positive ou négative.

Affiche une boîte de message de confirmation, un bouton OK et un bouton ANNULER. Il faut utiliser cette méthode lorsque l'utilisateur doit prendre une décision.

La méthode confirm retourne TRUE si l'utilisateur active le bouton OK et retourne FALSE si l'utilisateur active le bouton ANNULER.

Même si la méthode confirm utilise l'objet window, vous n'avez pas besoin de spécifier une fenêtre de référence lors de l'appel.

Syntaxe

confirm("unMessage")

unMessage est une chaîne de caractères ou une propriété d'un objet existant.

Exemple

En voici un exemple:

Dans l'exemple du fichier **COURSJS09.HTM**, une question est posée à l'utilisateur.

La méthode **alert()** est ensuite utilisée pour faire afficher le résultat. C'est donc une variable qui détient la valeur retournée par la fenêtre de dialogue, dans ce cas-ci la variable **reponse**.

```
<SCRIPT LANGUAGE="JavaScript">
<!-- Debut script
reponse=confirm("Répondez par ok ou annuller");
alert("vous avez répondu la valeur " + reponse);// Fin script -->
</SCRIPT>
```

prompt()

La méthode **prompt** sert àrecueillir de l'information de la part de l'usager. En effet, une case de texte invite l'utilisateur àentrer l'information.

Affiche une boîte de dialogue avec un message et un champ de saisie.

Même si la méthode prompt utilise l'objet window, vous n'avez pas besoin de spécifier une fenêtre de référence lors de l'appel.

Syntaxe

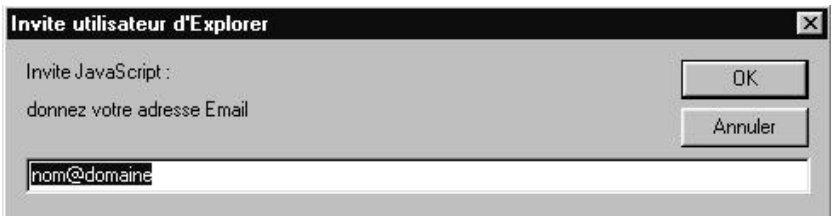
prompt(Message, [valeurDéfaut])

Message est une chaîne de caractères ou une propriété d'un objet existant. La chaîne de caractères représente le message.

valeurDéfaut est une chaîne de caractères, un entier ou une propriété d'un objet existant qui représente la valeur de défaut du champ d'entrée. Si vous ne donnez pas de valeur initiale pour **valeurDéfaut**, la boîte de dialogue affichera la valeur: **<undefined>.**

Exemple

L'exemple suivant montre une utilisation possible:



Dans l'exemple du fichier **COURSJS10.HTM**

```
<SCRIPT LANGUAGE="JavaScript">
<!-- Debut script
reponse=prompt("Veuillez entrer votre adresse de courrier électronique","nom@domaine");
alert(reponse);
//  Fin script -->
</SCRIPT>
```

Lorsque l'utilisateur n'entre aucune valeur, la fenêtre retourne la valeur par défaut, soit "e-mail" (le 2è paramètre).

Si l'utilisateur entre une valeur, c'est celle-ci qui sera stockée dans la variable **reponse**.

Si l'usager clique sur le bouton Annuler, la valeur **null** sera renvoyée.

Si aucune valeur n'est passée en deuxième paramètre, et que l'utilisateur valide sans effectuer de saisie, la valeur **undefined** sera renvoyée.

T.P: Détecter une version javascript

LES TABLEAUX

Nature des tableaux :

Les tableaux en javascript ne sont pas forcément un ensemble de variables de même types, comme dans les langages de programmation classiques, il s'agit en fait de tableaux associatifs, dans lesquels on stocke un ensemble d'association pouvant être très dissemblables, il s'agit d'un ensemble de valeurs désigné par un seul nom de variable

Tableaux numériques

Les tableaux nécessitent une déclaration préalable à leur utilisation. Par exemple un tableau pourrait s'appeler Eleve et serait déclaré par l'instruction

Cette déclaration n'est possible qu'à partir de la version 1.2 →

```
Eleve = ["durand","dupont","Ernest"]
```

Les éléments d'un tableau sont désignés par leur index, c'est àdire le nombre placé entre crochet, les tableaux commencent à0 et donc le ième élément porte l'index [i-1]

mais il est possible de déclarer au préalable un tableau pour augmenter la lisibilité du programme

Cette déclaration est toujours possible →

```
Eleve = new Array(3)
```

On utilise ici l'opérateur new sur un objet prédéfini Array

et on pourrait initialiser classiquement ce tableau avec un code du genre

```
for ( var j=0; j<3; j++)
{
        Eleve[j]=""
}
```

N.B: un tableau constitué avec des indices numériques reste donc lisible "classiquement" comme dans tout langage de programmation

COURSJS11.HTML

On utilise ici la méthode "classique"

```
Eleve = new Array(3)
Eleve[0]="durand";
Eleve[1]="dupont";
Eleve[2]="Ernest";
for ( var j=0; j<3; j++)
{
document.write("Valeur de tab[" + j + "] = " + Eleve[j] +"<BR>");
}
for ( var j=0; j<3; j++)
{
  Eleve[j]="e"
}
for ( var j=0; j<3; j++)
{
document.write("Valeur de tab[" + j + "] = " + Eleve[j] +"<BR>");
}
```

Si on souhaitait utiliser la déclaration de tableau accélérée du genre

```
Eleve=["durand","dupont","Ernest"]
```

que faudrait-il faire ?

et bien simplement remplacer

```
Eleve = new Array(3)
Eleve[0]="durand";
Eleve[1]="dupont";
Eleve[2]="Ernest";
```

par

```
Eleve=["durand","dupont","Ernest"]
```

mais aussi en toute logique spécifier

```
<script LANGUAGE="JavaScript1.2">
```

La solution est proposée en **COURSJS11B.HTML**

Tableaux associatifs

On l'a dit, en javascript les tableaux sont en fait sont des tableaux associatifs, permettant de manipuler des objets de types différents avec la même appellation

La déclaration d'un tableau associatif ne peut se faire qu'avec l'appel de l'opérateur new sur l'objet pré-défini array...

Un tableau associatif se déclare donc de manière identique (en fait tous les tableaux sont associatifs en javascript)

```
Tabass = new Array(3)
```

L'affectation d'un élément du tableau peut se faire avec

```
Tabass["nom1"] = valeur
Tabass["nom2"] = valeur
Tabass["nom3"] = valeur
```

Pour faire afficher toutes les éléments d'un objet on utilise l'instruction **for in**

dont la syntaxe est

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

et on peut parcourir ce tableau avec un code du genre

```
for ( var in Tabass)
{
        alert(Tabass[var]);
}
```

N.B: un tableau constitué avec des indices non numériques, donc associatif, ne peut être lu classiquement et doit être parcouru uniquement de la manière décrite ci-dessus

N.B: l'ordre de parcours n'est pas garanti !

javascript utilise des tableaux associatifs sur les objets, les images les formulaires d'une page HTML

Regardons l'exemple

COURSJS12.HTML

Ici le tableau est clairement associatif

```
tab = new Array(3)
tab["elem1"] = "voilà"
tab["elem2"] = "un tableau"
tab["elem3"] = "associatif"
```

et on ne peut plus utiliser une méthode de parcours "classique" →

```
for ( var j=0; j<3; j++) // cela ne marche plus
{
document.write("Valeur de tab[" + j + "] = " + tab[j] +"<BR>");
}
```

soit on liste tous les éléments →

```
                    // c'est lourd si les éléménts sont nombreux
document.write("Valeur de tab[elem1] = " + tab["elem1"] +"<BR>");
document.write("Valeur de tab[elem2] = " + tab["elem2"] +"<BR>");
document.write("Valeur de tab[elem3] = " + tab["elem3"] +"<BR>");
```

soit on s'adapte ! →

```
for (ind in tab) // mais ici l'ordre de parcours n'est pas garanti !
{
document.write("Valeur d'un element x de tab "  + tab[ind] +"<BR>");
}
```

Dans un script, on souhaite manipuler des élèves, sous forme de tableau associatif.

Un élève est caractérisé par un nom, un age et une note,

Créer un script dans lequel on déclare un eleve nommé eleve, dont le nom est "durand", l'âge est 45 et la note est 12.5

montrez comment on peut afficher à l'écran toutes les valeurs caractérisant cet élève...

La solution se trouve en **COURSJS12B.HTML**

Propriétés-méthodes des tableaux :

L'objet tableau Array en javascript dispose de plusieurs propriétés et de méthodes, notamment

length : propriété permettant de connaître la taille d'un tableau

reverse() : méthode permettant d'inverser un tableau

sort() : méthode permettant de trier un tableau selon l'ordre ascii

join() : méthode permettant de créer à partir de tous les éléments d'un tableau un chaîne caractère

regarder les exemples suivant fournis en **COURSJS13.HTM**

```
tab = new Array(5)
tab[0] = "durand"
tab[1] = 45
tab[2] = 12.5
tab[3] = 65
tab[4] = 4

tab.reverse();
for (ind in tab)
{
document.write("Valeur d'un element x de tab " + tab[ind]+"<BR>");
}
tab.sort();
for (ind in tab)
{
document.write("Valeur d'un element x de tab "  + tab[ind] +"<BR>");
}
document.write("liste des valeurs"  + tab.join() +"<BR>");
taille =  tab.length;
document.write("taille de tab "  + taille +"<BR>");
```

NOTIONS D'EVENEMENTS

Qu'est-ce un évènement:

Un des principaux intérêts de javascript est de pouvoir modifier le comportement du navigateur grâce aux actions de l'utilisateur.

On s'appuie pour cela sur la notion d'**événement,** qui représentent les principales actions de l'utilisateur: le passage de la souris, un clic de la souris, chargement d'une image

Comme on peut le voir dans le tableau ci-dessous, le liste est longue et parfois même très riche…

Evènement	Déclenché lorsque...
Abort	lors d'une interruption de chargement d'une image
Blur	Un élément n'est plus sélectionné avec le clavier ex: on clique hors d'un champs de formulaire
Change	Un élément est désélectionné et a été modifié ex: on modifie un champs de formulaire
Click	L'utilisateur clique sur un élément de la page ex: click sur un lien hypertexte
DblClick	L'utilisateur double-clique sur un élément
DragDrop	Lorsque on fait glisser un objet dans le navigateur
Error	Un élément n'a pas été chargé ou une erreur se produit lors de l'exécution d'un script
Focus	Un élément est sélectionné avec le clavier
KeyDown	L'utilisateur appuie sur une touche sur un élément
KeyPress	L'utilisateur a appuyé sur une touchee, puis l'a relâchée
KeyUp	L'utilisateur relâche une touche

Load	Le navigateur ouvre une page HTML ou charge une image
MouseDown	L'utilisateur appuie sur le bouton de la souris
MouseMove	L'utilisateur déplace la souris sur un élément
MouseOut	L'utilisateur fait ressortir la souris d'un élément
MouseOver	L'utilisateur place la souris sur un élément
MouseUp	L'utilisateur relache le bouton de la souris
Move	Une fenêtre est déplacée
Reset	un formulaire est réinitialisé
Resize	Une fenêtre est redimensionnée
Select	du texte est sélectionné dans un champs de formulaire
Submit	un formulaire est envoyé
Unload	une page est déchargée (lors du chargement d'une nouvelle page à la place

Gestionnaire d'évènement:

Pour utiliser les évènements disponibles en javascript, on àrecours àdes **gestionnaires d'événement** qui s'utilisent de la manière suivante:

<TAG_HTML param_html gest_événement = "script">

❶ balise HTML standard avec ses éventuels paramètres

❷attribut de la balise destiné à la gestion d'évènement xxx : commence par **on**xxx

❸ programme javascript devant s'exécuter lors de la survenue de l'évènement

❶ Toute les balises HTML acceptent tous leurs paramètres standard, le fait que une gestion d'événement soit prévue ne change rien àla syntaxe purement HTML de la balise

❷ Bien sûr toutes les actions ne sont pas possibles sur toutes les parties de la page HTML ou du navigateur, et on à ainsi des "couples" qui permettent de connaître les combinaisons possibles "objet-événement" qui pourront déclencher des scripts Javascript

❸ le programme javascript peut être :
une commande javascript unique
une série d'instruction séparée par des **;** voire des blocs **{ }**
une fonction définie dans l'en-tête du programme

fichier
COURJS14.HTML

```
<BODY>
<script LANGUAGE="JavaScript">
<!--
function m1() {
        alert("premier message")
}
function m2() {
        alert("deuxième message")
}
function m3() {
        alert("troisième message")
}
function appm() {
        m1()
        m2()
        m3()
}
//-->
</script>
voici un <A HREF="bidon.htm" onMouseover="alert('bonjour')">lien</A> un peut ... vivant
<BR>

voici un <A HREF="bidon.htm" onMouseover="alert('bonjour'); alert('au revoir')">lien</A> un peut plus ... vivant
<BR>

voici un <A HREF="bidon.htm" onMouseover="appm()">lien</A> carrément ... exhubérant
</BODY>
```

fonctions m1(), m2(), m3() et appm()

commande javascript

série d'instructions séparés ;

appel àune fonction …

évènement par défaut et spécifiques:

Pour la plupart de ces évènements, le navigateur possède déjàun comportement par défaut.

Par exemple l'événement **onClick** sur un lien hypertexte provoque le chargement de la page associée au lien.

Lorsque l'on ajoute un gestionnaire d'événement à un objet, au moment où l'événement survient, le navigateur exécute:

1. d'abord votre gestionnaire
2. puis celui prévu par défaut !

Si on ne veut pas que le navigateur exécute son gestionnaire d'événement par défaut, il suffit de passer en paramètre l'expression javascript **return(false)**

N.B: une exception existe à ce propos, concernant l'événement **onMouseOver** pour la propriété **status de l'objet window (cf Tp autre gestion d'événement cité plus loin... infra)**

Quelques évènements javascript:

N.B: La colonne "version" indique la version minimum du navigateur, qu'il s'agisse de Netscape ou d'Explorer. Ces informations sont indicatives...

Evènement	Balises autorisées	Evénement déclenché lorsque...	Version
onBlur	LABEL, INPUT, SELECT, TEXTAREA, BUTTON	Un élément n'est plus sélectionné avec le clavier	3.0
onChange	INPUT, TEXTAREA	Un élément est désélectionné et a été modifié	4.0
onClick	Presque toutes les balises	L'utilisateur clique sur un élément	4.0
onDbClick	Presque toutes les balises	L'utilisateur double-clique sur un élément	4.0
onError	Presque toutes les balises	Un élément n'a pas été chargé	3.0
onFocus	LABEL, INPUT, SELECT, TEXTAREA, BUTTON	Un élément est sélectionné avec le clavier	3.0
onKeyDown	Presque toutes les balises	L'utilisateur appuie sur une touche sur un élément	4.0
onKeyPress	Presque toutes les balises	L'utilisateur appuie, puis relâche une touche	4.0
onKeyUp	Presque toutes les balises	L'utilisateur relâche une touche sur une événement	4.0
onLoad	BODY, FRAMESET	Le navigateur a ouvert la page	3.0
onMouseDown	Presque toutes les balises	L'utilisateur appuie sur le bouton de la souris	4.0
onMouseMove	Presque toutes les balises	L'utilisateur déplace la souris sur un élément	4.0
onMouseOut	Presque toutes les balises	L'utilisateur fait ressortir la souris d'un élément	4.0
onMouseOver	Presque toutes les balises	L'utilisateur place la souris sur un élément	4.0
onMouseUp	Presque toutes les balises	L'utilisateur relache le bouton de la souris	4.0
onReset	FORM	Un formulaire est réinitialisé	3.0

onSelect	INPUT, TEXTAREA	L'utilisateur sélectionne du texte	4.0
onSubmit	FORM	Un formulaire est envoyé	3.0
onUnLoad	BODY, FRAMESET	Le navigateur ferme la page HTML	3.0

Sur annulation (onAbort)

Cet événement se produit lorsque l'usager annule le chargement d'une image. Ceci peut se produire lorsque l'usager appuie sur un lien ou sur le bouton "Stop" du navigateur au cours du chargement de l'image.

L'on peut retrouver cet événement dans la commande suivante:

- Image <IMG>

Exemple:

```
<IMG SRC="unURL" onAbort="nomFonction">
```

Sur sortie du focus (onBlur)

Cet événement se produit lorsque l'utilisateur sort d'une boîte de dialogue en appuyant sur la touche TAB du clavier ou en utilisant la souris. La fonction appelée peut servir àvalider les informations entrées.

L'on peut retrouver cet événement dans les commandes suivantes:

- Boîte liste <SELECT>
- Boîte àliste déroulante <SELECT>
- Boîte texte <INPUT TYPE="text">
- Boîte Mot de passe <INPUT TYPE="password">
- Boîte texte multiligne <TEXTAREA>

Exemple:

```
<INPUT TYPE="text" VALUE="uneValeur" NAME="unNom" onBlur="nomFonction">
```

Sur changement (onChange)

Cet événement se produit lorsque l'utilisateur modifie la valeur d'une d'une boîte texte.

L'on peut retrouver cet événement dans les commandes suivantes:

- Boîte liste <SELECT>
- Boîte àliste déroulante <SELECT>
- Boîte texte <INPUT TYPE="text">
- Boîte Mot de passe <INPUT TYPE="password">
- Boîte texte multiligne <TEXTAREA>

Exemple:

```
<INPUT TYPE="text" VALUE="uneValeur" NAME="unNom" onChange="nomFonction">
```

Sur clic Souris (onClick)

Cet événement se produit lorsque l'utilisateur appuie sur un bouton ou sur une option dans un formulaire ou sur un lien hypertexte.

L'on peut retrouver cet événement dans les commandes suivantes:

- Le bouton formulaire <INPUT TYPE="button">
- Le bouton recommencer <INPUT TYPE="reset">
- Le bouton soumettre <INPUT TYPE="submit'>
- La boîte de marquage <INPUT TYPE="checkbox">
- La boîte de sélection <INPUT TYPE="radio">
- Le lien externe ou interne <A HREF="..."> </A>

Exemple:

```
<INPUT TYPE="button" VALUE="uneValeur" onClick="nomFonction">
```

Sur erreur (onError)

Cet événement se produit brsqu'il y a une erreur au cours du chargement d'une image.

L'on peut retrouver cet événement dans la commande suivante:

- Image <IMG>

Exemple:

```
<IMG SRC="unURL" onError="nomFonction">
```

Sur focus (onFocus)

Cet événement se produit lorsque l'utilisateur appuie sur une boîte de dialogue avec la souris ou la touche TAB du clavier.

L'on peut retrouver cet événement dans les commandes suivantes:

- Boîte liste <SELECT>
- Boîte àliste déroulante <SELECT>
- Boîte texte <INPUT TYPE="text">
- Boîte Mot de passe <INPUT TYPE="password">
- Boîte texte multiligne <TEXTAREA>

Exemple:

```
<INPUT TYPE="text" VALUE="uneValeur" NAME="unNom" onFocus="nomFonction">
```

Sur chargement (onLoad)

Cet événement se produit lorsque le navigateur a terminé le chargement d'une page HTML ou de tous les cadres (FRAMES) de la commande <FRAMESET> ou d'une image.

L'on peut retrouver cet événement dans les commandes suivantes:

- Le corps du document <BODY>
- Les cadres <FRAMESET>
- L'image <IMG>

Exemple:

```
<BODY onLoad="nomFonction">
```

Sur souris hors zone (onMouseOut)

Cet événement se produit à chaque fois que l'utilisateur place la souris en dehors d'une zone hypertexte.

L'on peut retrouver cet événement dans les commandes suivantes:

- L'image en coordonnées <AREA>
- Lien externe <A HREF="..">
- Lien interne <A HREF="#..">

Exemple:

```
<AREA SHAPE=".." COORDS=".." HREF=".." onMouseOut="nomFonction()">
```

Sur souris au-dessus (onMouseOver)

Cet événement se produit à chaque fois que l'utilisateur place la souris au-dessus d'un lien hypertexte ou une zone hypertexte.

L'on peut retrouver cet événement dans les commandes suivantes:

- Le lien externe ou interne <A HREF="..."> </A>
- L'image en coordonnées <AREA>

Exemple:

```
A HREF="unURL" onMouseOver="nomFonction()">   </A>
AREA SHAPE=".." COORDS=".." HREF=".." onMouseOver="nomFonction()">
```

Sur recommencement (onReset)

Cet événement se produit lorsque l'utilisateur reset un formulaire.

L'on peut retrouver cet événement dans la commande suivante:

- La commande Formulaire <FORM>

Exemple:

```
<FORM ACTION="URL" METHOD=POSTouGET onReset="nomFonction()">
```

Sur sélection (onSelect)

Cet événement se produit lorsque l'utilisateur sélectionne la valeur d'une boîte texte ou d'une boîte texte multiligne.

L'on peut retrouver cet événement dans les commandes suivantes:

- Boîte texte <INPUT TYPE="text"> ou <INPUT TYPE="password">
- Boîte texte multiligne <TEXTAREA>

Exemple:

```
<INPUT TYPE="text" VALUE="uneValeur" NAME="unNom" onSelect="nomFonction">
```

Sur envoi (onSubmit)

Cet événement se produit lorsque l'utilisateur envoie un formulaire. L'on peut retrouver cet événement dans la commande suivante:

- La commande Formulaire <FORM>

Exemple:

```
<FORM ACTION="URL" METHOD=POSTouGET onSubmit="nomFonction()">
```

Sur déchargement (onUnload)

Cet événement se produit lorsque l'utilisateur sort d'une page HTML .

L'on peut retrouver cet événement dans les commandes suivantes:

- Le corps du document <BODY>
- Les cadres <FRAMESET>

Exemple:

```
<BODY onUnload="nomFonction">
```

 **T.P: Gestion d'évènement sur liens**

 **T.P: Gestion d'évènement sur champs**

 **T.P: Confirmer un lien hyper-texte**

NOTIONS DE P.O.O.

Objets – Méthodes - Propriétés:

Objet - Classe:

Pour illustrer le concept d'objet, avec ses propriétés et ses méthodes, prenons un exemple de la vie courante . Un objet est une chose, au sens courant du terme, par exemple une voiture. Et un objet peut contenir d'autres objets, par exemple un moteur, une portière... On parle aussi de classe

Méthode :

On peut définir les actions possibles avec cet objet, grâce cette fois-ci, non pas à des adjectifs, mais à des verbes. C'est ce que l'on appelle des méthodes. Ainsi avec une voiture, on peut démarrer, accélérer, freiner, tourner... L'ensemble des méthodes d'un objet représente toutes les actions que l'on peut effectuer avec cet objet.

On ne peut faire qu'une seule chose avec une méthode, c'est l'appliquer sur l'objet souhaité. Cependant les méthodes possèdent souvent des paramètres (arguments) qui précisent la façon dont on veut que la méthode s'applique. Ainsi la méthode Freiner applicable à l'objet voiture pourrait se paramétrer par doucement, sec, pile...

Propriété :

Pour décrire un objet on utilise des adjectifs qui le caractérisent, ce sont les propriétés de cet objet : couleur, année, marque, nombre de siège... L'ensemble des propriétés d'un objet représentent tous les aspects de cet objet. On peut s'en servir soit pour avoir une information sur l'objet (lecture) soit pour modifier l'objet (écriture). Cependant certaines propriétés sont en lecture seule, c'est à dire non modifiables, par exemple l'année de construction de notre voiture.

Programmation évènementielle :

Un événement est une action qui peut être détectée sur un objet, comme un clic souris, ou la frappe d'une touche. l'idée en général consiste à vouloir écrire un code

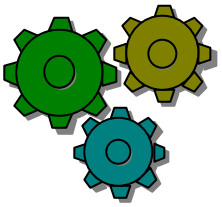
spécifique javascript en réponse àcet événement. On parle alors de programmation évènementielle…

Un exemple de modèle objet :

Objet VOITURE

(conteneur de moteur, siège…)

Propriétés de l'objet VOITURE :	**nom**	**type**
	Couleur	Lect/écrit
	Année	Lecture seule
	Marque	Lecture seule
	Nb sièges	Lect/écrit
	Vitesse	Lect/écrit
Méthodes de l'objet VOITURE :	**nom**	**paramètres**
	Démarrer	Vite, lent,norm
	Accélérer	Vite, maxi
	Freiner	Sec, Doux,pile
	Tourner	gauche,Droite
Méthodes de l'objet VOITURE :	**nom**	**renvoit type**
(Particulières)	Sièges	Sièges (coll)
		Siège (obj)
	Moteur	Moteur (obj)



Objet MOTEUR

(conteneur de …)

Propriétés de l'objet MOTEUR :	**nom**	**type**
	Puissance	Lecture seule
	Nb cylind	Lecture seule
	Vidange	Lect/écrit

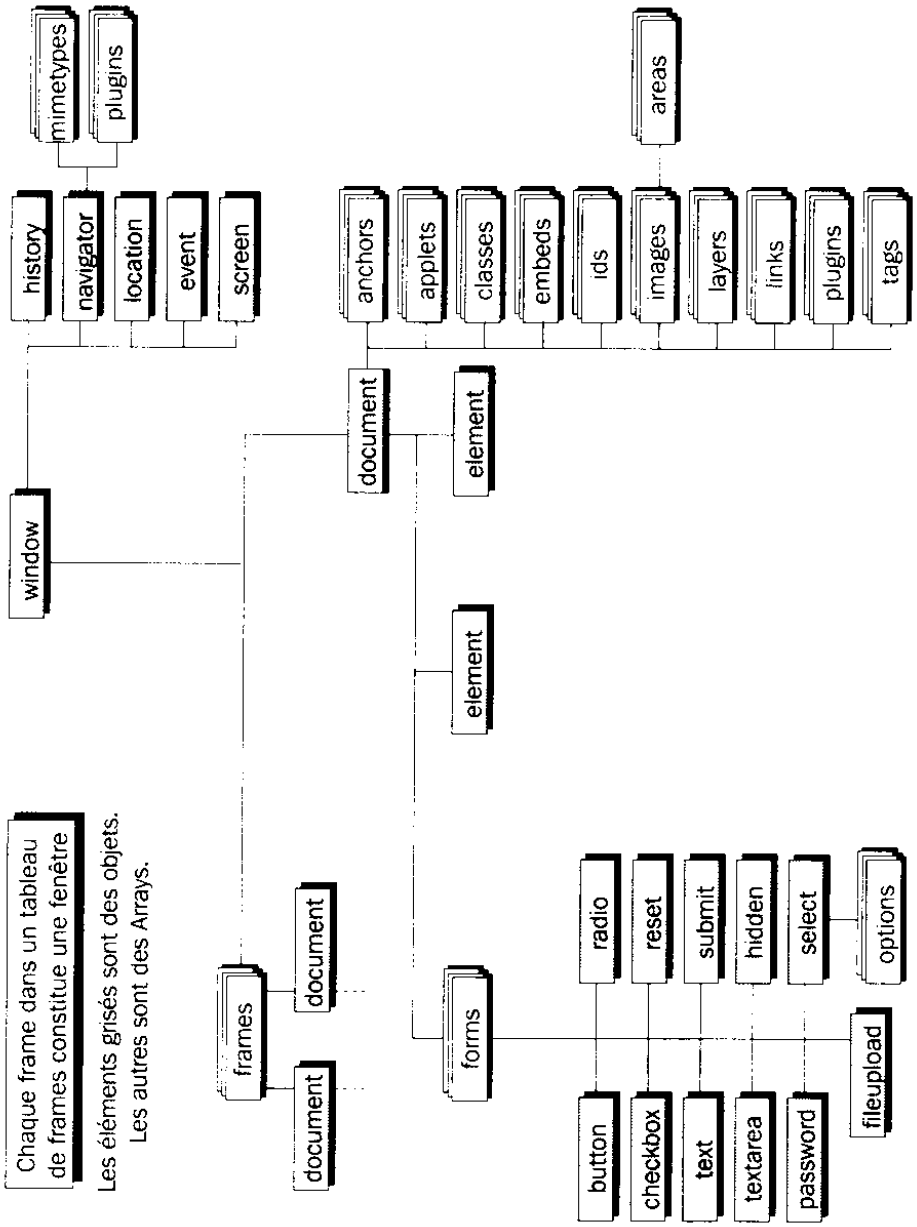
Pour récupérer la date de vidange, il faudrait écrire :

Date = Voiture.Moteur.Vidange

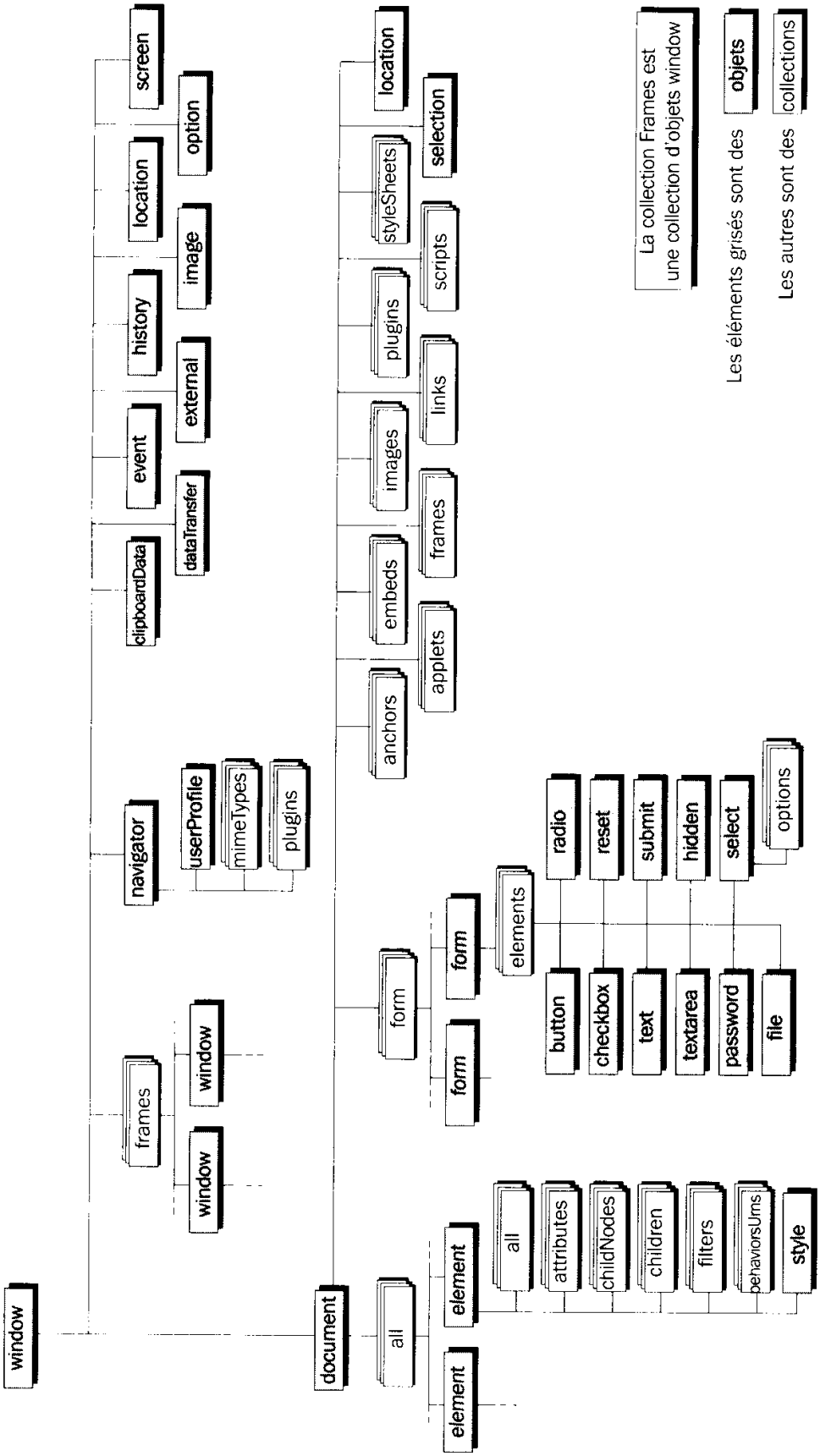
Pour modifier la date de vidange , il faudrait écrire :

Voiture.Moteur.Vidange = 01/04/96

Modèle objet window Netscape :



Modèle objet windowExplorer :



PRINCIPAUX OBJETS JAVASCRIPT

Objets et versions :

Javascript est composé d'objets intégrés (ou classes) pour travailler avec les éléments d'une page HTML, qui s'enrichissent au fur et à mesure que les versions évoluent., il existe des constructeurs qui permettent de créer simplement un objet de la classe en question. En effet en syntaxe normale on écris un "constructeur" par

var nom_objet = new Objet()

avec Objet() le constructeur de la classe dont on veut créer un élément

Trois versions principales référencées existent aujourd'hui:

- 1.0 : comportant 5 objets de base : **Date**, **math**, **string navigator**, **window**,
- 1.1 : ajoutant 1 objets : **function**
- 1.2 : ajoutant 2 objets : **arguments**, **screen**

Les deux versions "récentes" sont :

- 1.3 : n'apportant aucune amélioration de "base" utilisable facilement
- 1.4 : actuellement implémentée dans aucun navigateur

les classes intégrées dans les 3 premières versions de base, fournissant donc des méthodes et des propriétés permettant de travailler avec ces objets sont les suivantes

Version 1.0	Version 1.1	Version 1.2
Date Math window Navigator string	Date Math window Navigator string Function	arguments Date Math window Navigator screen string Function

Voyons les principaux objets

Date

Cet objet permet de travailler avec des dates alors que le type date n'existe pas pour les variables.

Ses **propriétés** sont les suivantes

Version 1.0	Version 1.1 & 1.2
Date	Date prototype

Les **méthodes** applicables àl'objet **Date** sont les suivantes

Version 1.0 & 1.1 & 1.2	
getDate() getDay() getHours() getMinutes() getMonth() getSeconds() getTime() getTimezoneOffset() getYear() parse(messageDate) setDate(valeurDate) setHours(valeurHeure) setMinutes(valeurMin) setMonth(valeurMois) setSeconds(valeurSec) setTime(valeurTemps) setYear(valeurAnnee) toGMTString() toLocaleString() UTC(annee,mois,jour[,heure][,min][,sec])	

Aucun **événement** n'est applicable àl'objet **Date**

Exemple

Pour créer un objet de type Date il faut écrire :

```
var nom_objet = new Date()
```

Une fois un objet Date construit, on peut l'utiliser par les nombreuses méthodes qu'il possède

imaginons vouloir afficher deux boutons permettant d'afficher respectivement l'année, et le jour et le mois en cliquant dessus

on écrira respectivement une fonction **an()** et **joursmoi()** permettant d'effectuer ces calculs àpartir d'un objet Date

les méthodes àutiliser sont **getYear()**, **getDate()** et **getMonth()**

la solution se trouve en **COURSJS15.HTML** (mais cherchez tout seul…)

voilàde quoi aurait l'air l'écriture des deux boutons avec leur gestion d'événement

```
<form>
<input type=button value="vous voulez savoir l'année ?" onclick="an()"> <br>
<input type=button value="vous voulez savoir le jour et le mois ?"
onclick="jourmois()">
</form>
```

le script javascript, placé en début de page, doit contenir une création d'objet de type Date, puis une application des méthodes nécessaires…

Cette méthode retourne l'année pour une date spécifiée. La valeur retournée est l'année moins 1900 (mais 2000 sous IE...)

```
function an(){
                var datage = new Date()
                alert("année 19"+ datage.getYear())
}
```

Cette méthode retourne le jour du mois pour une date spécifiée. La valeur retournée est un entier entre 1 et 31

```
function jourmois(){
                var datage = new Date()
                alert("le "+ datage.getDate() + "du mois" +
                                (datage.getMonth()+1) )
}//-->
```

Cette méthode retourne le mois pour une date spécifiée. La valeur retournée est un entier entre 0 et 11, où zéro représente janvier et 11 représente décembre.

ainsi pour obtenir un "mois" plus lisible, il faut lui ajouter 1, (la valeur) ce qui s'obtient en additionnant +1

```
(datage.getMonth()+1)
```

N.B: Si on oubliais les parenthèses, on concatènerait avec 1 convertit en chaîne car d'autres chaînes apparaissent dans l'expression…

```
alert("le "+ datage.getDate() + "du mois" + datage.getMonth()+1 )
```

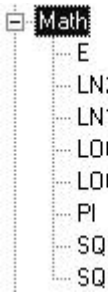
T.P: Calcul durée chargement de page

Math

Cet objet permet de manipuler des constantes et des fonctions mathématiques.

Ses **propriétés** sont les suivantes

Version 1.0 & 1.1 & 1.2



Les **méthodes** applicables àl'objet **Math** sont les suivantes

Version 1.0 & 1.1 & 1.2	
abs(n) acos(n) asin(n) atan(n) atan2(x,y) ceil(n) cos(n) exp(n) floor(n) log(n) max(n1,n2) min(n1,n2) pow(base,exposant) random() round(n) sin(n) sqrt(n) tan(n)	

Aucun **événement** n'est applicables àl'objet **Math**

Exemple

On n'a pas besoin de créer de variables pour manipuler un objet de type Math, on dit que la classe est déjà**instanciée**

On peut utiliser les propriétés (en lecture) de l'objet pour les constantes, ou utiliser les méthodes qu'il possède pour effectuer des calculs

imaginons vouloir afficher deux boutons permettant d'afficher respectivement Pi, et un nombre aléatoire en cliquant dessus

on écrira respectivement une fonction **pi()** et **alea()** permettant d'effectuer ces calculs àpartir d'un objet Math

la propriété àutiliser est **Pi**, la méthode est **random()**

la solution se trouve en **COURSJS16.HTML** (mais cherchez tout seul…)

voilàde quoi aurait l'air l'écriture des deux boutons avec leur gestion d'événement

```
<BODY>
<form>
<input type=button value="valeur de Pi ?" onclick="pi()"> <br>
<input type=button value="valeur aléatoire entre 0 et 1 ?" onclick="alea()">
</form>

</BODY>
```

le script javascript, placé en début de page, peut directement utiliser une propriété ou une des méthodes nécessaires…

Cette propriété représente le rapport de la circonférence d'un cercle sur son diamètre, environ 3.14159. Utilisée en lecture seulement.

```
<script LANGUAGE="JavaScript">
<!--

function pi(){
                alert("Pi vaut "+ Math.PI)
}
```

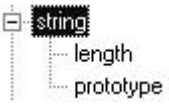
Cette méthode mathématique retourne un nombre choisi au hasard entre 0 et 1.

```
function alea(){
                alert("au hasard "+ Math.random())
}//-->
</script>
```

string

Cet objet permet de travailler avec des chaînes de texte et de les manipuler,

Sses **propriétés** sont les suivantes

Version 1.0	Version 1.1 & 1.2
	

Les **méthodes** applicable àl'objet **string** sont les suivantes

Version 1.0 & 1.1	Version 1.2
anchor(attributNom) big() blink() bold() charAt(index) fixed() fontcolor(couleur) fontsize(taille) indexOf(valeurCherche[,index]) italics() lastIndexOf(valeurCherche[,index]) link(url) small() strike() sub() substring(index1,index2) sup() toLowerCase() toUpperCase()	anchor(attributNom) big() blink() bold() charAt(index) charCodeAt([index]) concat(string2) fixed() fontcolor(couleur) fontsize(taille) fromCharCode(num1, num2, ..., numn) indexOf(valeurCherche[,index]) italics() lastIndexOf(valeurCherche[,index]) link(url) match(expression) replace(expression, nouvChaine) search(expression) slice(debutIndex,[finIndex]) small() split([separateur], [limite]) strike() sub() substr(debut, [taille]) substring(index1[,index2]) sup() toLowerCase() toUpperCase()

Aucun **événement** n'est applicables àl'objet **String**

Exemple

Pour créer un objet de type String il faut écrire :

```
var nom_objet = new String('xxxxx')
```

Une écriture du style suivant est tolérée :

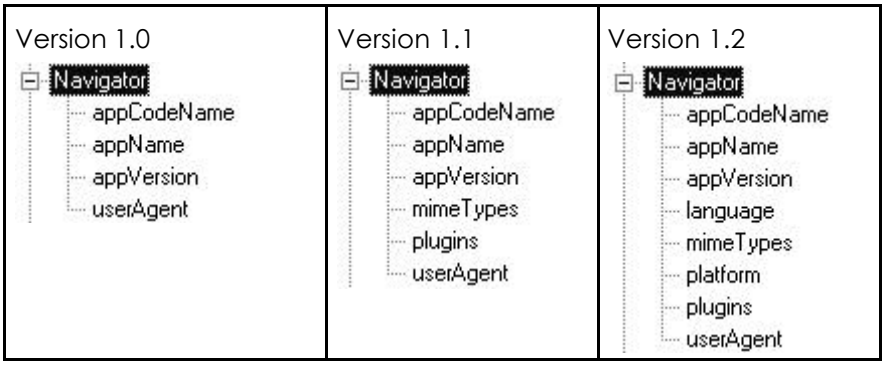
```
var nom_objet = 'xxxxx'
```

Une fois un objet string construit, on peut utiliser sa propriété ou ses nombreuses méthodes qu'il possède. La manipulation des chaînes, nécessite souvent de connaître la notion de tableau, de récupérer des valeurs saisies dans un boite de dialogue et est assez particulière

navigator

Cet objet permet de travailler avec les caractéristiques du browser.

Ses **propriétés** sont les suivantes



Aucune **méthode** ne s'applique àl'objet **navigator**

Aucun **événement** n'est applicables àl'objet **navigator**

Exemple

On n'a pas besoin de créer de variables pour manipuler un objet de type **navigator**, on dit que la classe est déjà**instanciée**

On peut utiliser les propriétés (en lecture) de l'objet pour avoir des informations sur le type de navigateur

imaginons vouloir afficher deux boutons permettant d'afficher respectivement le nom du navigateur, et sa version en cliquant dessus

on écrira respectivement une fonction **nom()** et **ver()** permettant d'afficher ces informations àpartir d'un objet Math

les propriétés àutiliser sont **appName** et **appVersion**

la solution se trouve en **COURSJS17.HTML** (mais cherchez tout seul…)

voilàde quoi aurait l'air l'écriture des deux boutons avec leur gestion d'événement

```
<BODY>
<form>
<input type=button value="nom du navigateur ?" onclick="nom()"> <br>
<input type=button value="version du navigateur ?" onclick="ver()">
</form>

</BODY>
```

le script javascript, placé en début de page, peut directement utiliser les méthodes nécessaires…

Cette propriété spécifie le nom du navigateur. Utiliser en lecture seulement.

Cette propriété spécifie la version du navigateur. Utiliser en lecture seulement.

```
<script LANGUAGE="JavaScript">
<!--

function nom(){
                alert("Navigateur "+ navigator.appName)
}

function ver(){
                alert("version "+  navigator.appVersion)
}//-->
</script>
```

T.P: Détecter le type de navigateur

Window

Permet de travailler sur la fenêtre du navigateur, par fenêtre dans un navigateur, on entends au sens large du terme, c'est àdire pour toute fenêtre (frame) àl'écran.

Il possède des **propriétés** qui sont énumérées ci-dessous, mais contient également des objets, au nombre variable selon les versions. principalement on notera les 4 objets: **document**, **frame, history**, **location**, qui sont détaillés plus loin dans le chapitre "**objet dans window**" page 75

N.B: certaines propriétés particulières permettent d'atteindre d'autres objets faisant partie de la classe Windows, et notamment les objets:

- document
- frame
- history
- location

, puis locationbar, menubar…

Version 1.0 & 1.1

- window
  - defaultstatus
  - document
  - [frame|frames['nom'|n]]
  - history
  - length
  - location
  - name
  - parent
  - self
  - status
  - top
  - window

Version 1.2

- window
  - defaultstatus
  - document
  - [frame|frames['nom'|n]]
  - history
  - innerHeight
  - innerWidth
  - length
  - location
  - locationbar
  - menubar
  - name
  - opener
  - outerHeight
  - outerWidth
  - pageXOffset
  - pageYOffset
  - parent

Les **méthodes** applicable àl'objet **window** sont les suivantes

Version 1.0

```
alert(message)
close()
confirm(message)
open(messageURL,messageNomFenetre,|
prompt(message[,valeurParDefaut])
setTimeout('fonction', msec)
clearTimeout(TimeOutID)
```

Version 1.1

```
alert(message)
blur()
close()
confirm(message)
focus()
open(messageURL,messageNomFenetre,[
prompt(message[,valeurParDefaut])
scroll(x,y)
setTimeout('fonction', msec)
clearTimeout(TimeOutID)
```

Version 1.2

```
alert(message)
back()
blur()
captureEvents(typeEvenement)
clearInterval(intervallD)
close()
confirm(message)
disableExternalCapture()
enableExternalCapture()
find(['chaine1'][,true|false][,true|false])
focus()
forward()
handleEvent(evenement)
home()
moveBy(horizontal,vertical)
moveTo(x,y)
open(messageURL,messageNomFenetre,|
print()
prompt(message[,valeurParDefaut])
releaseEvents(typeEvenement)
resizeBy(horizontal,vertical)
resizeTo(largeur,hauteur)
routeEvent(evenement)
scroll(x,y)
scrollBy(horizontal,vertical)
scrollTo(x,y)
setInterval(fonction, msec[arg1,...,argn])
setTimeout('fonction', msec[arg1,...,argn])
stop()
clearTimeout(TimeOutID)
```

Les **évènements** applicables àl'objet **window** sont les suivants

Version 1.0	Version 1.1	Version 1.2
onLoad onUnload	onBlur onError onFocus onLoad onUnload	onBlur onDragDrop onError onFocus onLoad onMove onResize onUnload

Événements "spéciaux"	déclenché lorsque:	standard
onBeforeUnload	l'utilisateur est sur le point de quitter un document: l'événement est déclenché juste avant l'événement onUnload.	IE
onDragDrop	une opération de glisser/déposer est effectuée sur la fenêtre.	N
onMove	la fenêtre est déplacée.	N
onReadyStateChange	équivalent àonLoad.	IE
onScroll	le document est défilé	IE

En résumé, Les **évènements** fréquemment applicables à l'objet **window** sont les suivants

- **onLoad :** le chargement de la fenêtre est fini
- **onUnload :** l'utilisateur quitte une fenêtre
- **onFocus :** la fenêtre reçoit le focus
- **onBlur :** la fenêtre perd le focus

  Lorsqu'une fenêtre est activée par un cliq souris ou par la touche "tab", on dit alors qu'elle reçoit le focus. À l'inverse, une fenêtre possédant le focus le perdra lorsqu'une autre fenêtre deviendra active.

  Ainsi, l'événement **onFocus** est déclenché lorsque qu'une fenêtre est activée pendant que l'événement **onBlur** s'exécute quand la fenêtre active perd le focus au profit de l'autre. Il faut noter qu'une fenêtre qui possède le focus par défaut, sans intervention de l'utilisation ne déclenche pas l'exécution de l'événement **onFocus**.

- **onError:** Lorsque une erreur survient

  L'événement **onError** est déclenché lorsqu'une erreur se produit dans l'exécution du code JavaScript de la page. On peut alors remplacer la fenêtre de dialogue qui s'affiche par défaut et fournir ainsi des indications beaucoup plus précises àl'utilisateur.

Ou définir les événements

Tous les événements décrits plus haut peuvent tout simplement être définis dans le marqueur BODYdu document HTML comme ceci:

```
<BODY événement=fonction JavaScript>
```

Par exemple, voici comment définir l'événement onLoad:

```
<BODY onLoad="chargementDocument()">
```

Exemple

On n'a pas besoin de créer de variables pour manipuler un objet de type Window, car comme pour l'objet Math javascript instancie automatiquement un objet Windows pour chaque fenêtre du navigateur ouverte, et dans le cas de frames, un objet pour chaque frame

Chaque objet Windows ainsi crée "possède" 3 "sous-objets :

- l'objet **history** : représentant l'ensemble des URL déjàvisitées, au travers de la propriété homonyme history
- l'objet **location** : représentant les caractéristiques de l'URL au travers de la propriété homonyme location
- l'objet **document** : représentant les caractéristiques du document HTML affiché et contenant lui même une multitude d'objets

Comme il s'agit ici de présenter simplement l'objet Windows dans sa globalité, on verra plus loin l'utilisation des objets qui sont inclus dans la classe windows

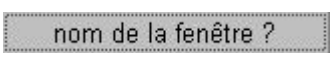
Il existe plusieurs manières de faire référence à un objet Windows, mais la plus classique reste

```
window.name="essai"
```

nomme la fenêtre "essai"

imaginons vouloir faire deux choses :

1. afficher un bouton permettant d'afficher le nom de la fenêtre.
2. faire apparaître, et ce dès le chargement de la page, dans la barre d'état le libellé par défaut "super, ça marche"



on écrira respectivement une fonction **nom()** et **gerestatus()** permettant respectivement de nommer la fenêtre (et de l'afficher via alert) et de donner un status par défaut dès l'ouverture de la page HTML

les propriétés àutiliser sont **name** et **status**

- **name** – Propriété des objets Anchor, Button, Checkbox, FileUpload, Form, Hidden, Image, Layer, Password, Plugin, Radio, Reset, Select, Submit, Text, Textarea et window

  Syntaxe: **RéfFenêtre.name** et/ou **RéfFenêtre.frames.name** est une façon valide de faire référence àune fenêtre, tel que décrit dans l'objet window

- **status** – Propriété de l'objet window. Cette propriété spécifie un message dans la barre d'état, tel que l'adresse URL, lorsque l'utilisateur passe le pointeur de la souris sur un hyperlien.

  Syntaxe: **réfFenêtre.status**

l'événement àutiliser pour la fonction gerestatus() est **onLoad**

la fonction gerestatus() est définie avant l'appel

L'on peut retrouver cet événement dans les commandes suivantes:

Corps <BODY>

Cadres <FRAMESET>

```
<HTML>
<TITLE></TITLE>
<script LANGUAGE="JavaScript">
<!--
function nom(){
                    window.name="essai"
                    alert("cette fenêtre s'appelle "+ window.name)
}
function gerestatus(){
                    window.status="super, ça marche"
}
//-->
</script>


<BODY onLoad="gerestatus()">

<FORM>
<INPUT type="button" VALUE="nom de la fenêtre ?" onclick="nom()">
<br>
<A HREF="bidon.htm">lien</A>
</FORM>
</BODY>
</HTML>
```

la solution se trouve en **COURSJS18.HTML**

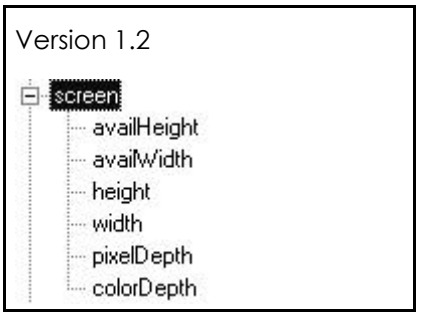
 **T.P: Gestion de Status**

 **T.P: Afficher un résultat dans le status**

screen (à partir de la version 1.2)

Cet objet donne des propriétés permettant de travailler avec les caractéristiques du moniteur vidéo du lecteur

ses **propriétés** sont les suivantes



Aucune **méthode** ne s'applique àl'objet **screen**

Aucun **événement** n'est applicables àl'objet **screen**

Exemple

On n'a pas besoin de créer de variables pour manipuler un objet de type **screen**, car comme pour l'objet Math javascript **instancie** automatiquement un objet **screen** lors du premier démarrage du navigateur

Il s'agit d'un objet simple dont on lira les propriétés pour en tirer un certain nombre de renseignements sur le moniteur du "lecteur"

T.P: Détecter la résolution d'écran

LES PRINCIPAUX OBJETS DANS WINDOW

Windows conteneur de :

L'objet **window** est lui aussi un objet des plus important du fait qu'il contient d'autres objets très complexes, on appelle cela un conteneur

les objets contenus dans la classe window dépendent des versions:

javascript Version **1.0** & **1.1**

- **document**
- **frame**
- **history**
- **location**

javascript Version **1.2**

- **document**
- **Frame**
- **history**
- **location**
- **locationbar / menubar / personalbar / scrollbars / statusbar / toolbar/ tags**

Pour des raisons de place et de compatibilité, nous nous limiterons àla version 1.0-1.1, soit les 4 objets principaux.

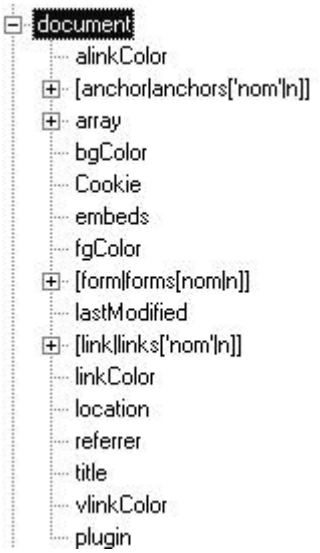
- **document,**
- **frame**
- **history**
- **location**

document

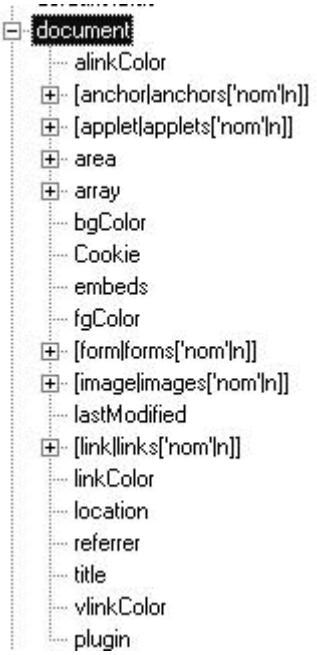
Cet objet permet de travailler avec la page HTML (formulaires, ancres…)

Il possède des propriétés qui sont énumérées ci-dessous, mais contient également des objets. principalement on notera les 4 objets **anchor**, **array**, **form**, **link**, qui sont détaillés plus loin dans le chapitre "**objet dans document**" (page 97)

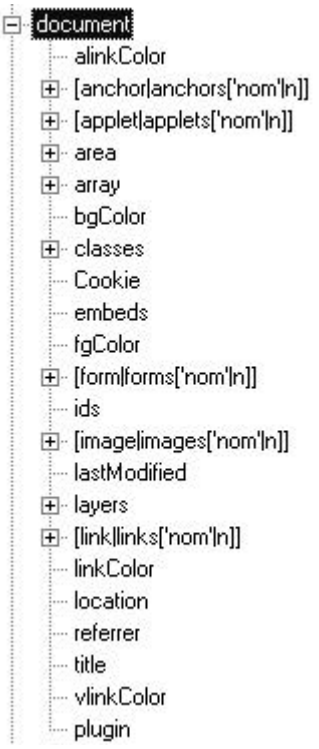
Version 1.0



Version 1.1



Version 1.2



Les **méthodes** applicables àl'objet **document** sont les suivantes

Version 1.0 & 1.1	Version 1.2
clear() close() open([text/html | text/plain | image/gif | image write(message1 [,message2,....,messageN]) writeln(message1 [,message2,....,messageN])	captureEvents(typeEvenement) clear() close() getSelection() handleEvent(evenement) releaseEvents(typeEvenement) routeEvent(evenement) open([text/html | text/plain | image/gif | image write(Message1 [,Message2,....,MessageN]) writeln(Message1 [,Message2,....,MessageN])

Les **évènements** applicables àl'objet **document** sont les suivants

Version 1.0 & 1.1	Version 1.2
	onClick onDblClick onKeyDown onKeyPress onKeyUp onMouseDown onMouseUp

Exemple document.bgColor :

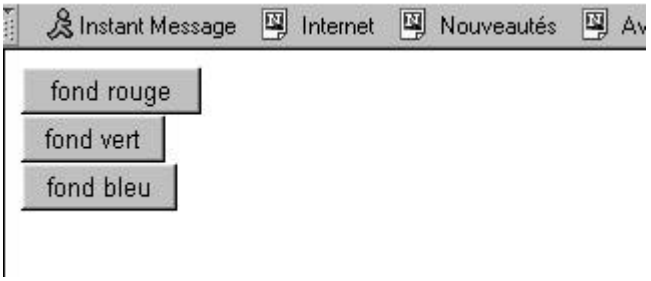
La propriété **bgColor** permet de modifier la couleur de fond de l'objet document

elle est très simple àutiliser

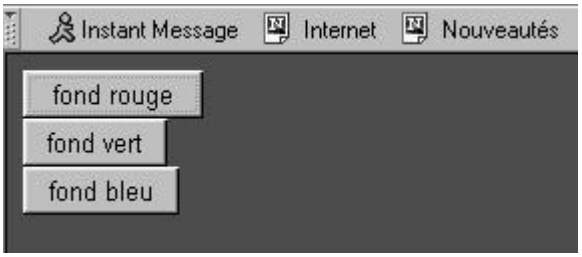
soit une fonction du type **colore()**

```
function colore(couleur){
      if(couleur=="rouge")
            window.document.bgColor="FF0000";
      if(couleur=="vert")
            window.document.bgColor="00FF00";
      if(couleur=="bleu")
            window.document.bgColor="0000FF";
}
```

on pourra alors créer facilement des boutons pour chaque couleur, qui appellerons la fonction avec le bon argument sur l'évènement clic souris...



appel respectivement de **colore('rouge')**, **colore('vert')** ou **colore('bleu')**

donnant 

essayer de coder cet exercice, la réponse se trouvant dans le fichier **COURJS23.HTML**

Exemple document.cookie :

Les **cookies** permettent au navigateur de stocker des informations de manière à pouvoir les récupérer lors de la prochaine visite et donc permettent de "faire le lien" entre deux sessions par un même client sur un site Web (connexion indépendantes et anonyme du point de vue protocole http). Il faut bien se rappeler que les cookies sont stockés sur l'ordinateur qui àchargé la page HTML et non pas sur le serveur sur lequel ces pages sont hébergées.

Les cookies sont de fichiers textes nommés automatiquement àpartir du nom donné par l'utilisateur lors de son ouverture de session et des caractère supplémentaires.(tels que l'URL du fichier d'où ils proviennent…)

Un cookie n'est jamais stocké au hasard par le navigateur, pour des raisons de sécurité en effet l'emplacement des cookies et défini par défaut, précisément par chaque navigateur (voir plus loin tableau récapitulatif)

De plus il s'agit de fichiers texte, absolument inoffensif donc pour la sécurité du client qui les accueille, et visualisables par un simple éditeur de texte

si le contenu en texte d'un cookie est libre, en général àl'exception des caractères , (virgule) ; (point virgule) et espace, qui sont interdits. La virgule en ce qui nous concerne jouera le rôle de séparateur, il existe quand même une "normalisation" pour certains champs…

name=value : permet de "nommer" notre cookie de manière à le retrouver parmi le autres éventuellement de l'objet cookie disponible en javascript

path=chemin : L'URL àlaquelle s'applique le cookie, par défaut il s'agit du dossier en cours, ce qui fait que pour toutes les pages d'un même dossier, tous les cookies éventuels sont mis dans le même fichier texte, les uns àla suite des autres. Il sera donc probablement nécessaire soit de "parcourir" un cookie du début à la fin à la recherche de toutes les informations, soit de rechercher notre information dans le cookie (caractérisée par notre "nom")

expires=date : si ce champs n'est pas précisé, le cookie est effacé àla fermeture du navigateur et La date doit être convertie en utilisant l'heure de Greenwich grâce àla méthode toGMTString

N.B: Un Cookie comportant des champs **name** et **path** identique ne peut être archivé qu'en un seul exemplaire !

L'objet qui permet de lire les cookies est accessible via '**document.cookie**". Si un cookie correspondant au chemin (path) du document que l'on visualise existe, (est présent sur le disque du client…), son contenu sera automatiquement mis dans cet objet. Ce qui fait que cet objet est donc une chaîne de caractère , pouvant être assez longue, contenant tous les cookies que l'utilisateur àrécolté sur cette page…

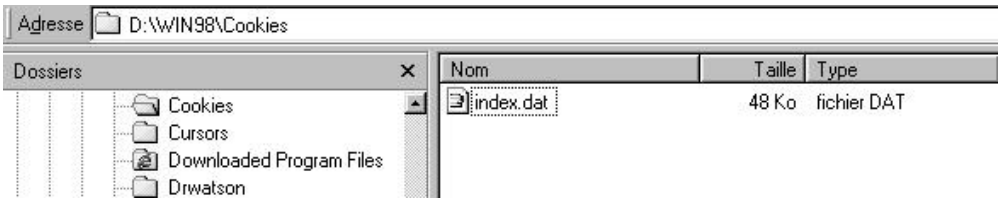
Pour utiliser les cookies, vous devez avoir 2 fonctions au minimum :

une fonction qui crée le cookie

une fonction qui lit le contenu de l'objet cookie

Création de Cookies

Si on prends le cas de Internet explorer sous Windows, les fichiers textes contenant les cookies sont dans un dossier Win\Cookies



la fonction suivante setCookie()

fichier COURJS25.HTML

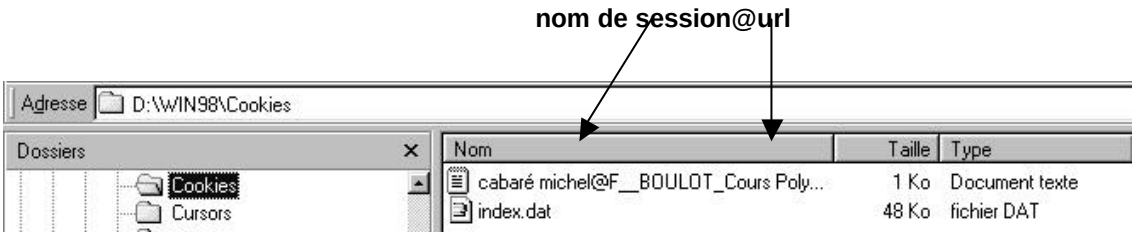
```
<script LANGUAGE="JavaScript">
function setCookie(nom, valeur, jours){
//nom est le nom du cookie
//valeur est la valeur a stocker dans le cookie
//jours est le nombre de jours son l'expiration
var expireDate = new Date();
expireDate.setTime(expireDate.getTime() + (jours * 24 * 3600 * 1000));
//création du cookie
document.cookie = nom + "=" + valeur + ";
expires=" + expireDate.toGMTString();
}
```

ainsi que la fonction **cre()** qui l'appelle après avoir demandé àl'utilisateur de donner un contenu au cookie (**valeur**), en nommant le cookie "**nom**" et en lui donnant une durée de **12** jours…

```
function cre()
{
valeur = prompt("donner la valeur à stocker","valeur du cookie");
setCookie("nom", valeur, 12);
}
</script>
```

l'appel de cette fonction cre() se faisant au chargement de la page…

```
<BODY onLoad="cre()">
```



une visualisation du cookie permet de vérifier l'exactitude de l'info en ce qui concerne le nom et la valeur stockée…

si on ferme le navigateur, et que l'on re-ouvre la page, on aura 2 chaines texte structurées de la même manière dans le fichier cookie…(àcondition d'avoir changé au préalable le nom du cookie, sinon il y a risque certain d'écrasement…)

Lecture de Cookies

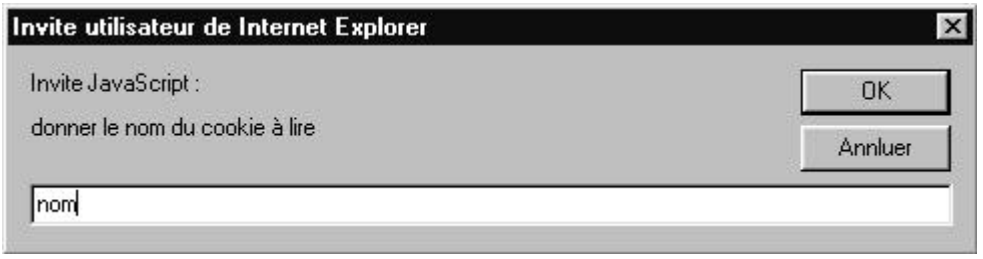
On trouveras en **CoursJs25.html** la fonction suivante getCookie()

```
function getCookie(nom){
//on vérifie si il y a un cookie
if (document.cookie.length > 0){
debut = document.cookie.indexOf(nom + "=");
//on vérifie si la valeur qu'on recherche est dans le cookie
if (debut != -1) //!= veut dire différent
{
debut += nom.length + 1;
fin = document.cookie.indexOf(";", debut);
if (fin == -1) fin = document.cookie.length;
return unescape(document.cookie.substring(debut, fin));
}
}
return null;
//la valeur n'a pas été trouvée...
}
```

ainsi que la fonction **lit()** qui l'appelle après avoir demandé àl'utilisateur de donner le **nom** du cookie que l'on souhaite lire…(ici toujours "nom"....)

```
function lit()
{
noml = prompt("donner le nom du cookie à lire");
lu = getCookie(noml);
alert (lu);
}
```

l'appel de cette fonction **lit()** se faisant sur un lien fictif



Contrôler les Cookies

Pour un utilisateur du Web, il est possible de contrôler les cookies émis depuis les sites vers son navigateur (et donc son poste) de la manière suivante :

Microsoft Internet Explorer 3.x :

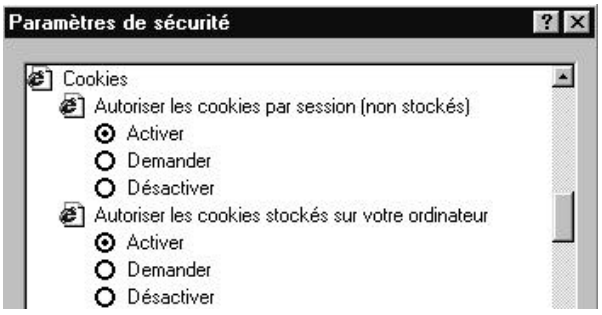
Affichage / options avancées et cliquer sur la case "**avertir avant d'accepter les cookies**"

Microsoft Internet Explorer 4.x :

Affichage / options internet cliquer sur le bouton "**avancées**" rechercher l'icône représentant un point d'exclamation jaune sous le libéllé "**securité**" et sélectionner l'une des options pour les cookies

Microsoft Internet Explorer 5.x :

Outils / options internet cliquer sur l'onglet '**sécuritée**" choisir ou créer un nom de site et demander '**personnaliser le niveau**" et sélectionner l'une des options pour les cookies

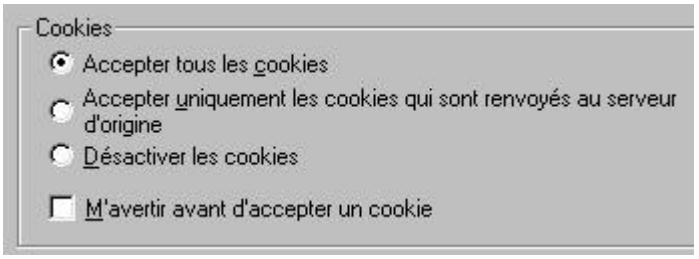


Netscape Communicator 3.x :

Option / Préférence Réseau et dans l'onglet "**protocole**" cocher éventuellement la case "**avertir avant d'accepter un cookie**"

Netscape Communicator 4.x :

Edition / Préférences et cliquer sur la case "**avancées**" et faire ses choix dans la zone des cookies



Où sont les Cookies ?

Pour un utilisateur du Web, il est possible de gérer les cookies, une fois que ceux-ci sont sur sa machine. (essentiellement les supprimer…)

leur localisation dépends évidemment du navigateur , Internet explorer ou Netscape et du système d'exploitation sur lequel on se trouve:

Pour Internet Explorer (toutes versions):

Sous Windows 95/98 : dans le dossier où windows est installé, et qui s'appelle par défaut X:\Windows il existe un dossier Cookies. Ce qui donne pour un disque C le chemin **C:\Windows\Cookies**

Sous Windows NT : dans le dossier où windows est installé, et qui s'appelle par défaut X:\Winnt il existe un dossier pour chaque utilisateur nommé \NomUtil. A l'intérieur se trouve un dossier Cookies. Ce qui donne pour un disque C, pour un utilisateur nommé paul le chemin **C:\Winnt\Paul\Cookies**

Pour Netscape navigator 3.x:

Netscape enregistre les cookies dans un seul fichier au format texte nommé "**cookie.txt**". Ce fichier contient l'ensemble des cookies

Lors d'une installation standard, les applications sont installées dans un dossier **\Program Files**,

Netscape s'installe dans **\Netscape\Navigator** pour les versions 3.x

Sous Windows 95/98 : Dans ce dossier il existe un fichier Cookies.txt. Ce qui donne pour un disque C le chemin **C:\Program Files\Netscape\Navigator\Cookies.tx**

Sous Windows NT : il existe un dossier pour chaque utilisateur nommé \Users\NomUtil Dans ce dossier il existe un fichier Cookies.txt. Ce qui donne pour un disque C pour un

utilisateur nommé paul le chemin **C:\Program Files\Netscape\Navigator\Users\Paul\Cookies.txt**

Pour Netscape communicator 4.x:

Netscape s'installe dans **\Netscape\Communicator** à partir de la version 4.x

Sous Windows 95/98 : Dans ce dossier il existe un fichier Cookies.txt. Ce qui donne pour un disque C **C:\Program Files\Netscape\Communicator\Cookies.tx**

Sous Windows NT : il existe un dossier pour chaque utilisateur nommé \Users\NomUtil Dans ce dossier il existe un fichier Cookies.txt. Ce qui donne pour un disque C pour un utilisateur nommé paul le chemin **C:\Program Files\Netscape\ Communicator \Users\Paul\Cookies.tx**

Plusieurs contenus dans un Cookie

essayer de modifier le fichier **CoursJs25.html** de manière àpouvoir mémoriser àpartir de cette page, un nom de famille, à chaque prénom donné lors de la saisie du cookie...

Concrètement au lieu de donner toujours le nom "nom" au cookie, on va créer un cookie qui contiendra plusieurs zones, que l'on nommera du prénom du visiteur, et pour lesquelles la valeur saisie sera le nom

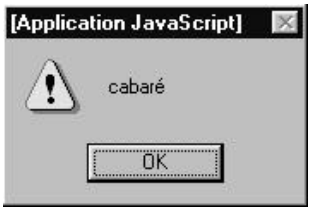
exemple : nom dans le cookie : "michel", valeur associée "cabaré".

nom dans le cookie : "laurent", valeur associée "lallias".

fichier COURJS25B.HTML

De manière àce que ensuite lorsque l'on demande de lire le cookie

on obtienne



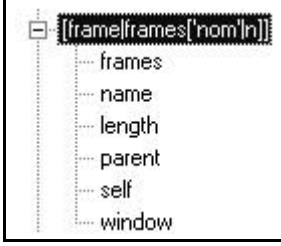
T.P: mémorisation d'un nom (cookie)

T.P: compteur de visites (cookie)

frame

Cet objet donne des informations sur chaque cadre composant un affichage àbase de frames.

Version 1.0 & 1.1 & 1.2



Les **méthodes** applicables àl'objet **frame** sont les suivantes

Version 1.0 & 1.1	Version 1.2
clearTimeout(timeOutID) setTimeout('fonction', msec)	clearInterval(intervalID) clearTimeout(timeOutID) print() setInterval(fonction, msec, [arg1, ..., argn]) setTimeout('fonction', msec, [arg1, ..., argn])

Les **évènements** applicables àl'objet **frame** sont les suivants

Version 1.0	Version 1.1	Version 1.2
	onBlur onFocus onLoad onUnLoad	onBlur onFocus onLoad onMove onResize onUnLoad

En javascript, le nom de l'objet Frame est le même que celui de l'attribut Name spécifié dans la balise <FRAME> de la page HTML.

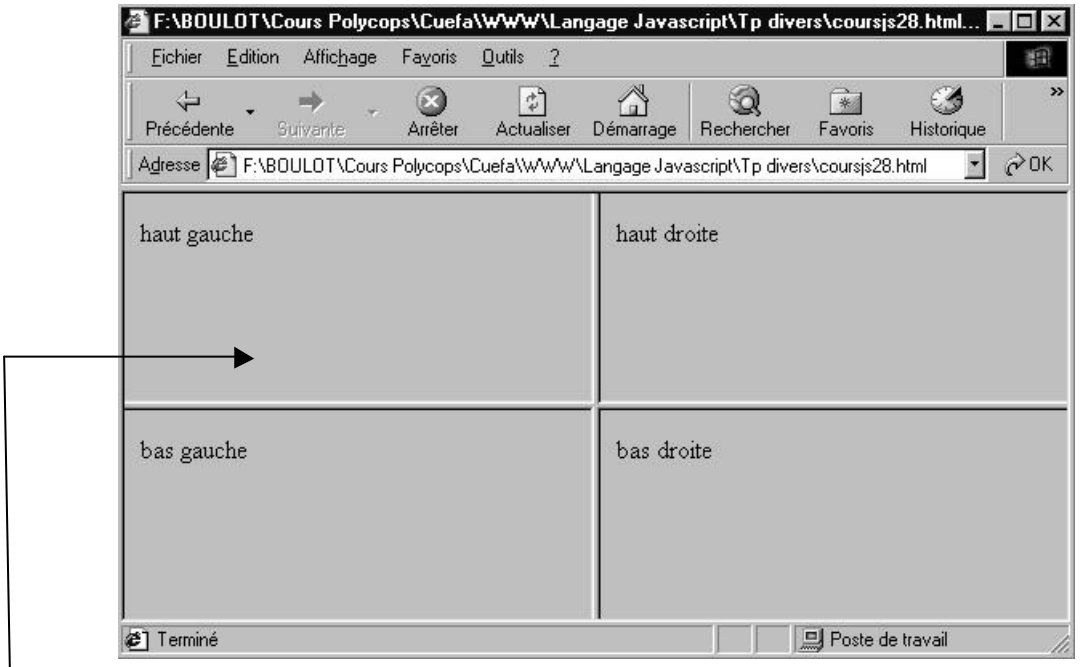
Dans le cas d'un cadre,

- **Window** et **Self** font référence àce cadre courant (alors que d'habitude il font référence àla fenêtre courante) alors que le mot clé
- **Parent** fait référence àla fenêtre principale. Chaque objet Frame est une objet enfant de l'objet **Parent**
- **Top** fait référence àla fenêtre parente de niveau le plus élevé
- **lenght** est une propriété donnant le nombre de frame dans l'objet

Exemple frame[x].name :

Dans le code suivant situé en **Coursjs28.html**

```
<Frameset rows="*,*" Cols="*,*">
<Frame name="hautgauche" Src="coursjs28_1.html">
<Frame name="hautdroite" Src="coursjs28_2.html">
<Frame name="basgauche" Src="coursjs28_3.html">
<Frame name="basdroite" Src="coursjs28_4.html">
</Frameset>
```



Un programme javascript situé en **coursjs28_1.html** fera référence aux autres cadres de plusieurs manières possibles :

- par le mot réservé **parent** :

  parent.hautdroite concerne **Coursjs28_2.html**

  parent.basgauche concerne **Coursjs28_3.html**

  parent.basdroite concerne **Coursjs28_4.html**

- par le mot réservé **window** ou **self**:

  window concerne **Coursjs28_1.html (courant)**

  self concerne **Coursjs28_1.html (courant)**

- par le tableau **frames[x]:**

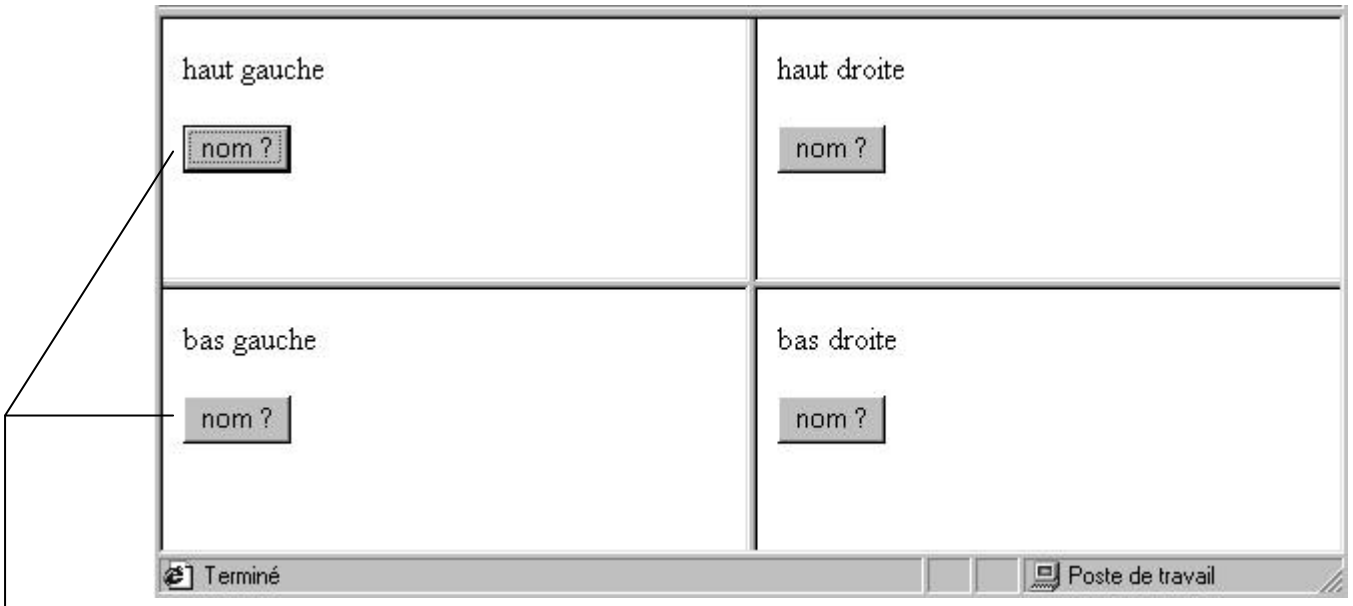
  dans lequel 0 désigne la première balise <Frame> du document

  parent.frames[0] concerne **Coursjs28_1.html**

  parent.frames[1] concerne **Coursjs28_2.html**

  parent.frames[2] concerne **Coursjs28_3.html**

parent.frames[3] concerne **Coursjs28_4.html**



Si dans chaque page on incorpore un bouton sur lequel lorsque l'on clique on exécute la fonction **donne()** suivante :

```
function donne()
{
      alert(" windows.name = " + window.name);
      alert(" self.name = " + self.name);
      for (cpt=0;cpt<4;cpt++)
      {
        alert(" parent.frames[" + cpt + "] = " + parent.frames[cpt].name);
      }
 }
```

Exemple top.frames.lenght :

que fait le programme suivant ?

```
if (top.frames.lenght!=0) top.location=self.document.location;
```

inséré dans la page appelé par un site, il permet de supprimer les frames de ce site

history

Cet objet donne des informations sur l'historique des URL qui ont étés parcourues par le navigateur

Version 1.0 & 1.1 & 1.2


Les **méthodes** applicables àl'objet **history** sont les suivantes

Version 1.0 & 1.1 & 1.2	
back() forward() go(n|'url')	

Aucun **événement** n'est applicables àl'objet **history**

Exemple

On n'a pas besoin de créer de variables pour manipuler un objet de type History, car comme pour l'objet Math javascript instancie automatiquement un objet History lors du premier démarrage du navigateur

Il s'agit d'un objet simple dont on utilisera les méthodes pour charger une des URL déjàparcourue par le navigateur

rappel : pour tester une petite ligne de javascript on peut la taper directement depuis la fenêtre URL du navigateur, précédée de la mention **javascript:**

Ainsi l'écriture

```
history.back();
```

charge l'URL précédente de l'historique (fonction identique au bouton précédent d'un navigateur)

Alors que l'écriture

```
history.forward();
```

charge l'URL suivante de l'historique (fonction identique au bouton suivant d'un navigateur)

N.B: l'appel àune instruction javascript du type **history.back()**; est le seul moyen lorsque dans un site web on àplusieurs manières d'arriver sur une page de proposer àl'utilisateur de revenir d'ou il vient !

Exemple history.back / forward :

faites un fichiers dans lequel on fait apparaître deux boutons aux fonctionnalités identiques àcelles des deux boutons précédent-suivant d'un navigateur...

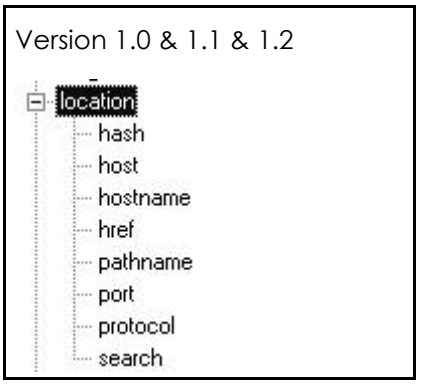
la solution est en **coursjs29.html**

```
<script LANGUAGE="JavaScript">
<!--
function retour(){
                history.back()
}
function suivant(){
                history.forward()
}
//-->
</script>
<form>
<input type=button value="retour" onClick="retour()">
<input type=button value="suivant" onClick="suivant()"> <br>
</form>
```

location

Cet objet donne des informations sur l'URL en cours

Version 1.0 & 1.1 & 1.2



Deux **méthodes** sont applicables àl'objet **location**

- **reload()** qui recharge le document en cours
- **replace**() qui remplace l'adresse actuelle par une nouvelle URL

  Cette méthode remplace l'URL du document courant dans l'historique du navigateur par l'URL fournie en argument. Après l'utilisation de la méthode replace, l'utilisateur ne peut revenir à l'URL précédent via le bouton PRÉCÉDENT (BACK) du navigateur

Aucun **événement** n'est applicables àl'objet **location**

Exemple

On n'a pas besoin de créer de variables pour manipuler un objet de type Location, car comme pour l'objet Math javascript **instancie** automatiquement un objet **location** lors du premier démarrage du navigateur

Il s'agit d'un objet simple dont on lira les propriétés pour en tirer un certain nombre de renseignements sur l'Url du document affiché àl'écran

rappel : pour tester une petite ligne de javascript on peut la taper directement depuis la fenêtre URL du navigateur, précédée de la mention **javascript:**

Ainsi l'écriture

```
alert(location.pathname);
```

renvoi le chemin du fichier actuellement affiché

Par exemple rajoutez l'écriture suivante

```
alert(" pathname =" + parent.frames[cpt].location.pathname);
```

dans les fichiers **Coursjs28_1.html, Coursjs28_2.html, Coursjs28_3.html, Coursjs28_4...**

alors que l'écriture

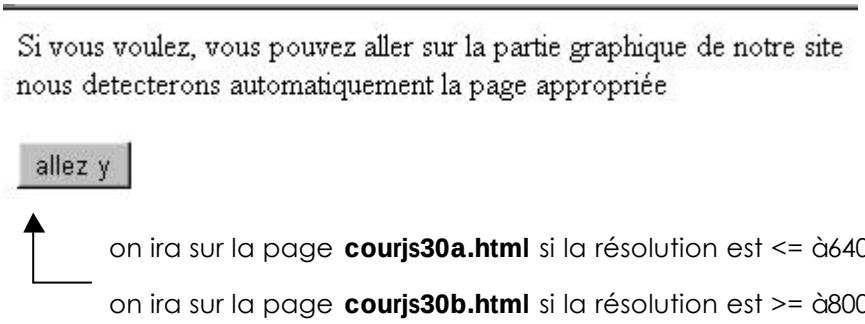
```
alert(location.href);
```

renvoi l'URL du du fichier actuellement affiché

Exemple location.replace :

on peut également par le biais de la méthode Replace() changer l'URL courante, et proposer ainsi un changement de page àla volée

On souhaite automatiquement faire en sorte que le visiteur soit dirigé vers une page donnée selon la résolution de son navigateur



la solution en **courjs30.html** a àpeu près l'aspect suivant :

```
<script LANGUAGE="JavaScript">
<!--
 function redirige()
 {
  if (screen.width <=640)
    location.replace("file:///C:/coursjs30a.html");
  else
     location.replace("file:///C:/coursjs30b.html");
 }

//-->
</script>

 Si vous voulez, vous pouvez aller sur la partie graphique de notre site <BR>
 nous detecterons automatiquement la page appropriée
<form>
  <input type = "submit" value="allez y" onClick="redirige()">
 </form>
</BODY>
```

N.B: bien que correct du point de vue javascript, ce script ne fonctionne pas avec toutes les versions de Internet explorer…Il faut dévalider le gestionnaire par défaut via "return false"

T.P: saisie d'une URL

T.P: Création d'une bannière de navigation

OUVRIR UNE NOUVELLE FENETRE

Il existe plusieurs fonctions JavaScript pour faire apparaître des fenêtres et communiquer avec l'utilisateur. La méthode **open()** par exemple permet de créer une nouvelle fenêtre de navigateur

La méthode open

La méthode **open()** permet d'ouvrir une nouvelle fenêtre de navigateur dans laquelle peut être chargé ou généré un document HTML. L'apparence de la nouvelle fenêtre peut être personnalisée de façon àfaire afficher ou non certaines caractéristiques, telles que la barre d'outils, la barre de localisation, etc

On se sert de la méthode open pour créer une nouvelle fenêtre de navigateur comportant ses propres caractéristiques.

Tous les paramètres sont optionnels. Ainsi dans **COURSJS19.HTM**

```
open ();
```

fonctionne, mais celle-ci également

```
open ("","fenetre","");
```

ou encore celle-ci

```
f = open ("","fenetre","");
```

Paramètres de la méthode open:

```
[objet window] = window.open ( [url] [,nom] [,caractéristiques])
```

Paramètre	Description
URL	adresse du document qui doit être chargé dans la nouvelle fenêtre
Nom	Il faut utiliser cette variable lorsque l'on fait référence aux propriétés, méthodes et contenu d'une fenêtre. (exemple cible (TARGET) pour d'autres documents HTML.)
Caractéristiques	liste d'options de la fenêtre (barre outils, grandeur …) Si plusieurs options sont définies, elles doivent être séparées par des virgules non suivies d'espace

caractéristiques :

À l'aide de la liste des propriétés, on peut personnaliser l'apparence d'une fenêtre créé:

Propriétés	Valeurs possibles	valeur par défaut	Description
directories	yes, no	no	barre des répertoires fournissant des liens vers divers sites Web
heigth	nombre en pixels		Spécifie la hauteur de la fenêtre en pixels
left	nombre en pixels		distance entre la bordure gauche de la fenêtre et le début de l'écran
location	yes,no	no	ajoute un champ de saisie permettant d'indiquer l'URL ou le chemin d'accès local d'une page Web àconsulter. En fait, la barre d'adresses
menubar	yes,no	no	ajoute une barre des menus dans le haut de la fenêtre
resizable	yes,no	no	cette option permet à l'utilisateur de redimensionner la fenêtre.
scrollbars	yes,no	no	ajoute des barres de défilement horizontale et verticale lorsque le contenu déborde des dimensions de la fenêtre.
status	yes,no	no	ajoute la barre d'état au bas de la fenêtre
toolbar	yes,no	no	ajoute la barre d'outils standard àla fenêtre du navigateur
top	nombre en pixels		distance entre la bordure supérieure de la fenêtre et l'écran
width	nombre en pixels		Spécifie la largeur de la fenêtre en pixels

Plus quelques ajouts récents :

alwaysLowered Si la valeur spécifiée est yes, cette option créée une nouvelle fenêtre flottante qui sera affichée derrière toutes les autres fenêtres du navigateur, qu'elles soient actives ou non. Il s'agit ici d'une option spéciale qui doit être utilisée dans un script signé. (JavaScript 1.2)

alwaysRaised Si la valeur spécifiée est yes, cette option créée une nouvelle fenêtre flottante qui sera affichée devant toutes les autres fenêtres du navigateur, qu'elles soient actives ou non. Il s'agit ici d'une option spéciale qui doit être utilisée dans un script signé. (JavaScript 1.2)

dependent Si la valeur spécifiée est yes, cette option créée une fenêtre enfant de la fenêtre courante. Une fenêtre enfant ferme lorsque sa fenêtre parent est fermée. (JavaScript 1.2)

hotkeys Si la valeur spécifiée est no (ou 0), cette option désactive la plupart des touches de raccourcis pour les fenêtres n'ayant pas de barre de menus. Les touches de sécurité et de fermeture de l'application demeurent toutefois disponibles. (JavaScript 1.2)

innerHeight Spécifie la hauteur de la zone de contenu de la fenêtre en pixels. Pour créer une fenêtre plus petite que 100 x 100 pixels, le script doit être signé. Remplace l'option Height, cette dernière faisant toujours partie de la norme pour compatibilité. (JavaScript 1.2)

innerWidth Spécifie la largeur de la zone de contenu de la fenêtre en pixels. Pour créer une fenêtre plus petite que 100 x 100 pixels, le script doit être signé. Remplace l'option Width, cette dernière faisant toujours partie de la norme àdes fins de compatibilité. (JavaScript 1.2)

outerHeight Spécifie la dimension verticale extérieure de la fenêtre en pixels. Pour créer une fenêtre plus petite que 100 x 100 pixels, le script doit être signé. (JavaScript 1.2)

personalbar Si la valeur spécifiée est yes, cette option ajoute la barre d'outils personnelle affichant les boutons du dossier de signets personnels de l'utilisateur. (JavaScript 1.2)

screenX Spécifie la position de la fenêtre àpartir de la gauche de l'écran. Pour placer une fenêtre hors des limites de l'écran, le script doit être signé. (JavaScript 1.2)

screenY Spécifie la position de la fenêtre àpartir du haut de l'écran. Pour placer une fenêtre hors des limites de l'écran, le script doit être signé. (JavaScript 1.2)

titlebar Si la valeur spécifiée est yes, cette option créée une fenêtre dotée d'une barre de titre. Pour définir une fenêtre sans barre de titre (no), le script doit être signé. (JavaScript 1.2)

z-lock (JavaScript 1.2) Si la valeur spécifiée est yes, cette option créée une nouvelle fenêtre ne pouvant passer devant les autres fenêtres du navigateurs même lorsqu'elle est activée. Le script doit être signé pour utiliser cette option. (JavaScript 1.2)

Par exemple on provoquera l'affichage d'une nouvelle fenêtre avec les propriétés spécifiées dans la liste:

```
fenetreA=window.open("test.html","contenu",
"status=no,location=yes,
toolbar=yes,directories=no,resizable=yes,
width=600,height=400,top=100,left=100");
```

tous les paramètres sont activables àtrue avec une valeur de **1** ou **yes** et àfalse avec une valeur de **0** ou **no**

Fermer une fenêtre

La méthode **close** est utilisée pour fermer une fenêtre; dans l'exemple,

```
f = open ("","fenetre","");
```

la fenêtre est représenté par l'objet **f**. et f pourra être fermée par

```
f.close();
```

La méthode close peut aussi s'appliquer à l'objet **window**, ce qui provoque la fermeture de la fenêtre courante:

```
window.close();
```

dans le fichier **COURSJS20.HTM** on crée un fenêtre puis on demande àl'utilisateur s'il souhaite la fermer ou non.

On ouvre ici le fichier **COURSJS20b.HTM** …

```
<script LANGUAGE="JavaScript">
<!--
f=window.open("coursjs20b.htm","contenu","status=no,location=yes,toolbar=yes,directories=no,resizable=yes,width=600,height=400,top=100,left=100");
rep = confirm("voulez vous supprimer cette fenêtre ?")
if (rep == true)
   f.close()
//-->
</script>
```

Écrire dans une fenêtre

Il est possible de générer le contenu d'une fenêtre ou d'un cadre àl'aide de la méthode **write** ou **writeln**.

On peut composer une page HTML entière et même y ajouter du code JavaScript. voir fichier **COURSJS21.HTM**

La méthode **open()**, appliquée à l'objet **document**, doit être utilisée avant de pouvoir écrire dans la fenêtre.

La méthode **close()** doit être employée lorsque l'opération d'écriture est terminée.

```
f=window.open("","contenu","status=no,location=yes,toolbar=yes,directories=no,resizable=yes,width=600,height=400,top=100,left=100");
f.document.open();
f.document.write ("<H1>Nouvelle fenêtre</H1>");
f.document.close();
rep = confirm("voulez vous supprimer cette fenêtre ?")
if (rep == true)
   f.close()
```

T.P: Ouverture / Création de fenêtre

T.P: Agrandir une mini-image dans une fenêtre

EVENEMENTS TEMPORISES

méthode setTimeout()

Cette méthode de l'objet window évalue une expression après qu'un délai en millisecondes soit expiré. Elle peut être annulée via l'appel à la méthode clearTimeout().

Syntaxe

TimeoutID=window.setTimeout(expression, msec)

Avec

TimeoutID est un identificateur utilisé seulement pour annuler l'évaluation avec la méthode **clearTimeout**().

expression est une chaîne de caractères ou une propriété d'un objet existant.

msec est une valeur numérique en unités millisecondes.

Exemple :

On créer une page demandant àl'utilisateur de cliquer sur un bouton pour continuer, on tombe sur une page affichant la bienvenue

Mais si au bout de 10 secondes, celui-ci n'a pas manifesté sa présence,

la page apparaît automatiquement

fichier en **coursjs22.htm**

```
<script LANGUAGE="JavaScript">
<!--
 function accueil()
 {
   f = window.open("coursjs22b.htm","acc");
      return true;
 }
//-->
</script>
```

le fichier **coursjs22b.htm** est la page qui doit être affichée

dans la variable **delai** on àle moyen d'annuler la temporisation…

…dans le cas où l'utilisateur àcliqué

```
</HEAD>
<BODY onLoad="delai = window.setTimeout('accueil()',5000);">
cliquez sur le bouton pour continuer
<FORM METHOD=POST>
      <INPUT TYPE=button VALUE="continuer"
                  onClick="window.clearTimeout(delai);accueil();">
</FORM>

</BODY>
```

LES OBJETS DANS DOCUMENT

document conteneur de :

L'objet **document** est un objet très important du fait qu'il contient d'autres objets intéressants et qu'il représente la totalité de la page HTML

les objets contenus dans la classe document dépendent des versions:

Version 1.0	Version 1.1	Version 1.2
• **anchor**	• **anchor**	• **anchor**
• **array**	• **applet**	• **applet**
• **form**	• **area**	• **area**
• **link**	• **array**	• **array**
	• **form**	• **classes**
	• **image**	• **form**
	• **link**	• **image**
		• **layers**
		• **link**

Pour des raisons de place et de compatibilité, nous nous limiterons àla version 1.0 soit les 4 objets principaux.

On dira quelques mots des objets intéressant apparus avec les nouvelles versions, soit les objets **image** et **area** (version 1.1) et l'objet **layers** (version 1.2)

anchor

Cet objet permet de travailler avec les ancres de la page HTML

Version 1.0 & 1.1 & 1.2

Aucune **méthode** n'est applicables àl'objet **anchor**

Aucun **événement** n'est applicables àl'objet **anchor**

array

Cet objet permet de travailler avec les tableaux de la page HTML

Version 1.0 & 1.1 & 1.2

Les **méthodes** applicable àl'objet **array** sont les suivantes

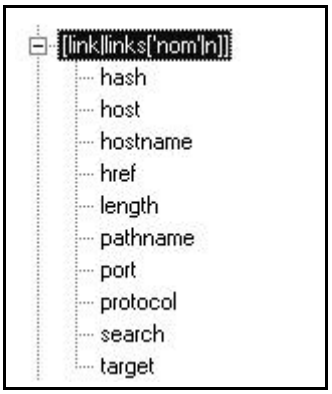
Version 1.0	Version 1.1	Version 1.2
	join([,messageSeparateur]) reverse() sort(fonctionComparaison)	concat(arrayNom2) join([messageSeparateur]) reverse() slice(debut,[fin]) sort(fonctionComparaison) toString()

Aucun **événement** n'est applicables àl'objet **array**

link

Cet objet permet de travailler avec les liens de la page HTML

Version 1.0 & 1.1 & 1.2



Aucune **méthode** n'est applicables àl'objet **link**

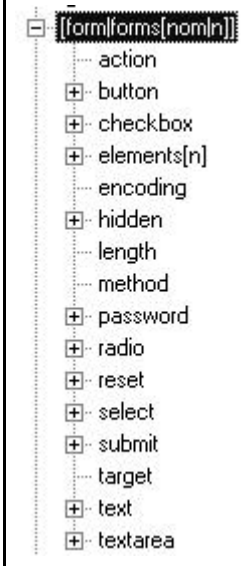
Les **évènements** applicables àl'objet **link** sont les suivants

Version 1.0	Version 1.1	Version 1.2
onClick onMouseOver	onClick onMouseOut onMouseOver	onClick onDblClick onKeyDown onKeyPress onKeyUp onMouseDown onMouseOut onMouseOver onMouseUp

form

Cet objet permet de travailler avec les formulaires de la page HTML

Il possède des propriétés qui sont énumérées ci-dessous, mais contient également des objets, au nombre variable selon les versions. principalement on notera les 4 objets **button**, **checkbox**, **hidden**, **password**, **radio**, **reset**, **select**, **submit**, **text**, **textarea**, qui sont détaillés plus loin dans le chapitre "**objet dans form**" (page 107)

Version 1.0	Version 1.1 & 1.2
	

Les **méthodes** applicable àl'objet **form** sont les suivantes

Version 1.0 & 1.1	Version 1.2
	reset() submit()

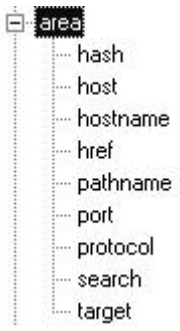
Les **évènements** applicables àl'objet **form** sont les suivants

Version 1.01	Version 1.1 & 1.2
	onReset onSubmit

area (à partir de version 1.1)

Cet objet permet de travailler avec les zones graphiques de la page HTML

Version 1.1 & 1.2



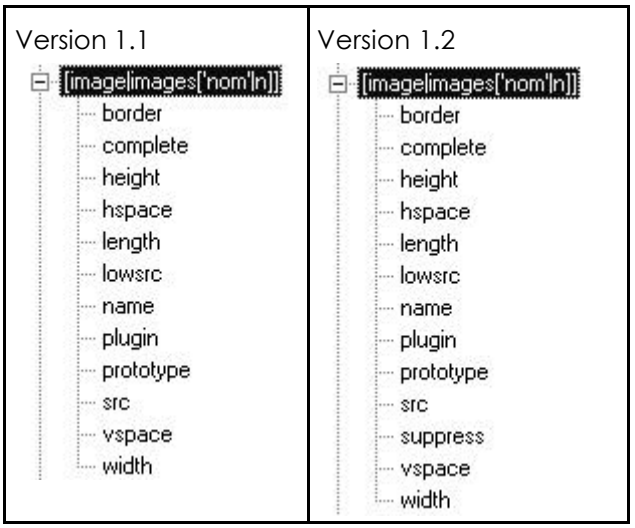
Aucune **méthode** n'est applicables àl'objet **area**

Les **évènements** applicables àl'objet **area** sont les suivants

Version 1.0 & 1.1	Version 1.2
	onClick onDblClick onMouseOut onMouseOver

image (à partir de version 1.1)

Cet objet permet de travailler avec les images de la page HTML



Aucune **méthode** n'est applicables àl'objet **image**

Les **évènements** applicables àl'objet **image** sont les suivants

Version 1.1	Version 1.2
onAbort onError onLoad	onAbort onError onKeyDown onKeyPress onKeyUp onLoad

Exemple image.src :

L'exemple que l'on va manipuler est celui permettant de faire un "survol de bouton"...

Cette fonctionnalité suppose que le navigateur intègre une version javascript 1.1 minimum, ce qui place la barre àExplorer 3.x et Netscape 3.x minimum

On va utiliser la propriété **src** de l'objet **image** pour changer "a la volée" l'image affichée dans une page HTML avec une syntaxe du type

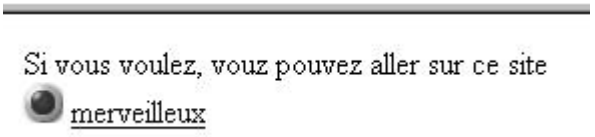
```
nomimage.src ="new image.gif"
```

On va également utiliser l' évènements **onMouseOut,** se produisant àchaque fois que le curseur de souris sort des limites de la surface d'un lien hypertexte ou encore d'une région d'une image en coordonnées. et l'événement **onMouseOver** qui se produit à

chaque fois que le curseur de souris se trouve sur des limites de la surface d'un lien hypertexte ou encore d'une région d'une image en coordonnées

Dans le fichier **COURSJS26.HTM**

on affiche très classiquement ce lien précédé d'une puce animée jaune/bleue




```
<BODY BGCOLOR="#FFFFFF" TEXT="#000000" >
<script LANGUAGE="JavaScript">
<!--
//-->
</SCRIPT>

 Si vous voulez, vouz pouvez aller sur ce site <BR>

<IMG SRC="gif/puceb.gif" name="puce1">  <A HREF = "r  " target = "blank">
merveilleux </A>

</BODY>
```

on nomme l'image…"puce1"

et target="blank" sert uniquement à annuler facilement un clik sur le lien…

Cette image étant nommée "**puce1**", l'instruction suivante incorporée dans la balise du lien <HREF…> permet donc chaque fois que la souris passe sur ce lien de changer le fichier source de l'image àafficher (et donner un effet de puce changeante...)

```
onMouseOver="puce1.src='gif/Pucer.gif'"
```

pucer.gif est une puce rouge

idem sur cette instruction lorsque le pointeur de la souris se déplace hors du lien…(la puce reprends sa couleur primitive)

```
onMouseOut="puce1.src='gif/Puceb.gif'"
```

puceb.gif est une puce bleue

construisez donc le fichier qui va permettre de vérifier que l'on pointe bien le lien "merveilleux" en changeant la couleur de la puce….

la solution est en **COURSJS26b.HTM**

```
<BODY BGCOLOR="#FFFFFF" TEXT="#000000" >
<script LANGUAGE="JavaScript">
<!--
//-->
</SCRIPT>

 Si vous voulez, vouz pouvez aller sur ce site <BR>

<IMG SRC="gif/Puceb.gif" name="puce1">  <A HREF = "r  " target = "blank"
```

```
onMouseOver="puce1.src='gif/Pucer.gif'"
onMouseOut="puce1.src='gif/Puceb.gif'"> merveilleux </A>
</BODY>
```

Mise en paramètre des fichiers image :

C'est très compliqué et peu "maintenable" de truffer le source d'appels directs a des fichiers images...

dans notre cas en effet, si un jour on souhaite changer les fichiers **pucer.gif** et **puceb.gif** cela revient àdevoir intervenir àplusieurs endroits dans le source !

Reprenons alors le fichier **COURSJS26b.HTM** de manière à déclarer au début des variables pointant sur ces 2 images, de manière àpouvoir les changer facilement :

Il va donc falloir

- 2 variables de type image instanciées :

```
image_select = new Image(); image_select.src="gif/pucer.GIF";
image_nonselect = new Image(); image_nonselect.src="gif/puceb.GIF";
```

- 1 fonction pour donner l'image indiquant la sélection (et une fonction pour donner l'image indiquant une non-sélection):

```
function allume(nom) {
switch (nom) {
  case "Bouton1" :
    document.Bouton1.src = image_select.src;
      break;
  case "Bouton2" :
    document.Bouton2.src = image_select.src;
    break;
  case "Bouton3" :
    document.Bouton3.src = image_select.src;
      break;
  }
}
```

- 1 fonction pour construire la page HTML àla volée avec l'image voulue :

```
<script LANGUAGE="JavaScript1.1">
document.write('<IMG SRC=');
document.write(image_nonselect.src);
document.write(' name="Bouton1">');
</script>
```

- et enfin faire des appels simples et indépendants des images :

```
<A HREF = "r " target = "blank" onMouseOver="allume('Bouton1')"
onMouseOut="eteint('Bouton1')"> merveilleux </A>
```

puis

```
<A HREF = "r " target = "blank" onMouseOver="allume('Bouton2')"
onMouseOut="eteint('Bouton2')"> sublime </A>
```

etc...

Une solution se trouve en **COURSJS26c.HTM**

2 variables de type image instanciées →

1 fonction pour donner l'image indiquant la sélection →

1 fonction pour donner l'image indiquant la non sélection →

écriture des boutons à la volée... →

```
<script LANGUAGE="JavaScript1.1">
<!--
image_select = new Image(); image_select.src="gif/pucer.GIF";
image_nonselect = new Image(); image_nonselect.src="gif/puceb.GIF";

function allume(nom) {
switch (nom) {
  case "Bouton1" :
    document.Bouton1.src = image_select.src;
      break;
  case "Bouton2" :
    document.Bouton2.src = image_select.src;
    break;
  case "Bouton3" :
    document.Bouton3.src = image_select.src;
      break;
  }
}
function eteint(nom) {
switch (nom) {
  case "Bouton1" :
    document.Bouton1.src = image_nonselect.src;
      break;
  case "Bouton2" :
    document.Bouton2.src = image_nonselect.src;
    break;
  case "Bouton3" :
    document.Bouton3.src = image_nonselect.src;
      break;
  }
}
//-->
</SCRIPT>
Si vous voulez, vouz pouvez aller sur un de ces site <BR>
<script LANGUAGE="JavaScript1.1">
document.write('<IMG SRC=');
document.write(image_nonselect.src);
document.write(' name="Bouton1">');
</script>
<A HREF = "r " target = "blank" onMouseOver="allume('Bouton1')"
onMouseOut="eteint('Bouton1')"> merveilleux </A>
<P>
<script>
document.write('<IMG SRC=');
document.write(image_nonselect.src);
document.write(' name="Bouton2">');
```

les appels sont simples et indépendant des fichiers image utilisés !

```
</script>
<A HREF = "r " target = "blank" onMouseOver="allume('Bouton2')"
onMouseOut="eteint('Bouton2')"> sublime </A>
<P>
<script>
document.write('<IMG SRC=');
document.write(image_nonselect.src);
document.write(' name="Bouton3">');
</script>
<A HREF = "r " target = "blank" onMouseOver="allume('Bouton3')"
onMouseOut="eteint('Bouton3')"> génial </A>
<P>
```

Attention : pour des raisons de chemin sur des fichiers locaux, ce fichier ne marche pas avec Netscape, il faudrait replacer les sources dans un dossier nommé de manière standardisée (nom de dossier sans espace...)

en effet sous Explorer les espaces sont codés en %20

```
alert(image_nonselect.src);
```

alors que sous Netscape non...

```
alert(image_nonselect.src);
```

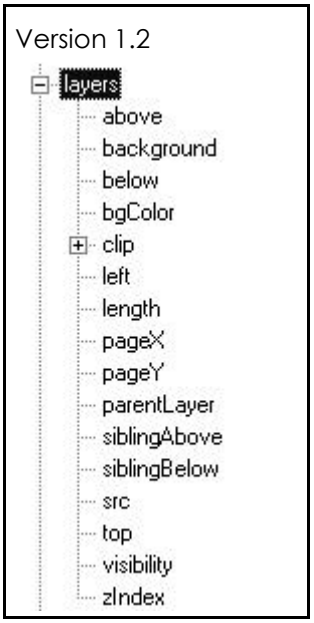
 **T.P: Changement d'image sur passage souris**

 **T.P: Menu visuels avec images changeantes**

layers (à partir de version 1.2)

Cet objet permet de travailler avec les couches (calques) de la page HTML

Version 1.2



Aucune **méthode** n'est applicables àl'objet **layer**

Les **évènements** applicables àl'objet **layers** sont les suivants

Version 1.2	
onBlur onFocus onLoad onMouseOut onMouseOver	

LES OBJETS DANS FORM

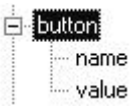
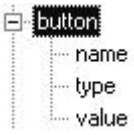
form conteneur de :

L'objet **form** est un objet important car il contient la totalité des objets utilisables dans un formulaire HTML

Version 1.0	Version 1.1 & 1.2
button	**button**
checkbox	**checkbox**
elements	**elements**
hidden	**FileUpload**
password	**hidden**
radio	**password**
reset	**radio**
select	**reset**
submit	**select**
text	**submit**
textarea	**text**
	textarea

button

Cet objet permet de travailler avec les "boutons" d'un formulaire de la page HTML

Version 1.0	Version 1.1 & 1.2
	

Les **méthodes** applicables àl'objet **button** sont les suivantes

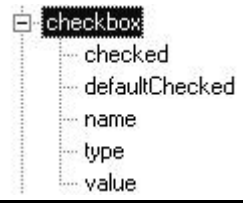
Version 1.0 & 1.1 & 1.2	
click()	

Les **évènements** applicables àl'objet **button** sont les suivants

Version 1.0	Version 1.1 & 1.2
onClick	onBlur onClick onDblClick onFocus onMouseDown onMouseUp

checkbox

Cet objet permet de travailler avec les "cases àcocher" d'un formulaire de la page HTML

Version 1.0	Version 1.1 & 1.2
	

Les **méthodes** applicables àl'objet **chekbox** sont les suivantes

Version 1.0 & 1.1 & 1.2	
click()	

Les **évènements** applicables àl'objet **chekbox** sont les suivants

Version 1.0 & 1.1	Version 1.2
onClick	onBlur onClick onDblClick onFocus onMouseDown onMouseUp

FileUpload (a partir de version 1.2)

Cet objet permet de travailler avec les ancres de la page HTML

Version 1.2
FileUpload name value

hidden

Cet objet permet de travailler avec les champs cachés d'un formulaire de la page HTML

Version 1.0	Version 1.1 & 1.2
hidden name value	hidden name type value

password

Cet objet permet de travailler avec les champs texte "password" d'un formulaire de la page HTML

Version 1.0	Version 1.1 & 1.2
password defaultValue name value	password defaultValue name type value

Les **méthodes** applicables àl'objet **password** sont les suivantes

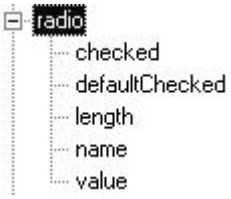
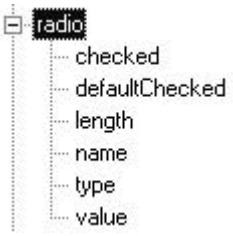
Version 1.0 & 1.1 & 1.2	
focus() blur() select()	

Les **évènements** applicables àl'objet **password** sont les suivants

Version 1.0 & 1.1	Version 1.2
	onBlur onChange onFocus onKeyDown onKeyPress onKeyUp onSelect

radio

Cet objet permet de travailler avec les "boutons radio" d'un formulaire de la page HTML

Version 1.0	Version 1.1 & 1.2
	

Les **méthodes** applicable àl'objet **radio** sont les suivantes

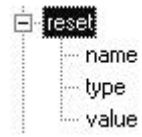
Version 1.0 & 1.1 & 1.2	
click()	

Les **évènements** applicables àl'objet **radio** sont les suivants

Version 1.0 & 1.1	Version 1.2
onClick	onBlur onClick onDblClick onfocus onMousedown onMouseUp

reset

Cet objet permet de travailler avec le champs Reset d'un formulaire de la page HTML

Version 1.0	Version 1.1 & 1.2
	

Les **méthodes** applicable àl'objet **reset** sont les suivantes

Version 1.0 & 1.1 & 1.2	
click()	

Les **évènements** applicables àl'objet **reset** sont les suivants

Version 1.0 & 1.1	Version 1.2
onClick	onBlur onClick onDblClick onFocus onMouseDown onMouseUp

select

Cet objet permet de travailler avec un champs de type SELECT d'un formulaire de la page HTML

Version 1.0	Version 1.1 & 1.2
	

Les **méthodes** applicable àl'objet **select** sont les suivantes

Version 1.0 & 1.1 & 1.2	
blur() focus()	

Les **évènements** applicables àl'objet **select** sont les suivants

Version 1.0 & 1.1 & 1.2	
onBlur onChange onFocus	

submit

Cet objet permet de travailler avec le champs submit d'un formulaire de la page HTML

Version 1.0	Version 1.1 & 1.2
	

Les **méthodes** applicable àl'objet **submit** sont les suivantes

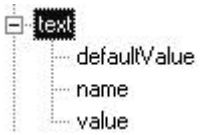
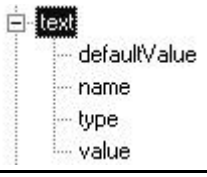
Version 1.0 & 1.1 & 1.2	
click()	

Les **évènements** applicables àl'objet **submit** sont les suivants

Version 1.0 & 1.1	Version 1.2
onClick	onBlur onClick onDblClick onFocus onMouseDown onMouseUp

text

Cet objet permet de travailler avec les champs "text" d'un formulaire

Version 1.0	Version 1.1 & 1.2
	

Les **méthodes** applicable àl'objet **text** sont les suivantes

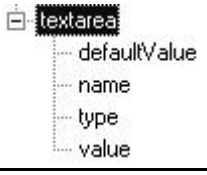
Version 1.0 & 1.1 & 1.2	
focus() blur() select()	

Les **évènements** applicables àl'objet **text** sont les suivants

Version 1.0 & 1.1	Version 1.2
onBlur onChange onFocus onSelect	onBlur onChange onFocus onKeyDown onKeyPress onKeyUp onSelect

textarea

Cet objet permet de travailler avec les champs "textarea" d'un formulaire

Version 1.0	Version 1.1 & 1.2
	

Les **méthodes** applicable àl'objet **textarea** sont les suivantes

Version 1.0 & 1.1 & 1.2	
focus() blur() select()	

Les **évènements** applicables àl'objet **textarea** sont les suivants

Version 1.0 & 1.1	Version 1.2
onBlur onChange onFocus onSelect	onBlur onChange onFocus onKeyDown onKeyPress onKeyUp onSelect

TRAVAILLER AVEC UN FORMULAIRE

Interêts :

Plus on effectue de manipulation sur le client àpropos du formulaire, et plus on décharge le serveur, ce qui ne peut que globalement être positif pour tout le monde

Plusieurs types de manipulations peuvent être intéressantes àeffectuer localement

- Un premier objectif peut être celui d'effectuer des contrôles de validité sur les saisies effectuées dans les champs de formulaire, mais avant envois de la requête sur le serveur (cela évite un rejet de toute façon).

- Un deuxième objectif peut être celui de demander confirmation lors de la demande d'envois (submit) d'un formulaire, ou d'interdire l'envois de ce formulaire lorsque certains champs critiques ne seraient pas remplis par l'utilisateur

- Un troisième objectif pourrait être constitué par des calculs simples effectués automatiquement lors de la saisie du formulaire, c'est àdire que lors d'un saisie de quantité et de prix unitaires, le total en cours se calcule automatiquement

- En fin pourquoi ne pas proposer pour les cas simples carrément un traitement du formulaire en local, c'est àdire via la messagerie et sans passer par l'exécution de script sur le serveur

Accéder à des champs de formulaire :

Il va falloir être capable de déclencher une fonction lorsque la valeur a été saisie dans le champs, ce qui peut être le cas de l'événement **onBlur** qui est déclenché lors de la perte de focus sur un champs. Voir le fichier **COURSJS31.HTML**

```
<script LANGUAGE="JavaScript">
<!--
function affiche(form1){
alert("vous avez saisi " + form1.nb.value);
return ;
};
//-->
</script>
```

nb est le champs nommé "nb"

value est une propriété d'un champs text de formulaire

La fonction **affiche()** récupère "form" un formulaire en paramètre. c'est nous qui le décidons (et qui le traitons ensuite comme un objet de type formulaire par la suite)

sur **onBlur** on appelle **affiche()** avec le formulaire en paramètre. **this** représentant l'objet courant...

```
<FORM>
    merci de saisir quelque chose
     <INPUT TYPE=TEXT NAME="nb" onBlur="affiche(this.form)" VALUE="0"
SIZE=5 MAXLENGTH=5> <br>
 </FORM>
```

Le champs du formulaire s'appelle "nb"

En appel à**affiche()** on passe un objet de type formulaire

C'est un paramètre réel, il doit exister

NB: lors de l'écriture, on peut tout aussi bien écrire

```
onBlur="affiche(this.form)"
```

form est 1 propriété de this, l'objet courant

que

```
onBlur="affiche(document.forms[0])"
```

forms[0] 1° éléments du tableau des formulaires du document

Essayez ces variantes

solution en **COURSJS31B.HTML**

ou bien si on a nommé le formulaire par

```
<FORM name="test">
onBlur="affiche(document.test)
```

test est 1 objet nommé du document

ATTENTION : Noter que l'écriture **this.test** n'est pas admise !

Calculs avec des chaînes :

Dès que l'on veut effectuer des opérations en javascript, le problème de conversion se pose car les données éventuellement récupérées dans un champs de formulaire sont en fait des chaînes de caractère

Par conséquent il est nécessaire de les convertir,

conversion en entier :

c'est la fonction **parseInt()** qui assure la conversion d'un chaîne en un entier classique

```
parsInt("13");
```

donnera le nombre 13

et

```
parsInt("13,5");
```

donnera le nombre 13

conversion en flottant (décimaux) :

c'est la fonction **parseFloat()** qui assure la conversion d'un chaîne en un flottant classique (nombre àvirgule)

```
parsFloat("13");
```

donnera le nombre 13.00

et

```
parsFloat("13,5");
```

donnera le nombre 13.5

ANNEXE : VERSIONS

JSCRIPT-JAVASCRIPT-ECMASCRIPT

Correspondances avec ECMASCRIPT :

Si la norme est édictée par l'organisme de Normalisation ECMA, dont les dernière spécifications sont disponibles àl'adresse suivante :

http://www.ecma.ch

Il s'agit de vérifier de quelle manière les 2 principales variations proposées par Netrscape et Microsoft se positionnent :

Le tableau suivant précise les versions de JScript et JavaScript compatibles avec ECMAScript :

Version de JavaScript	Compatible ECMAScript ?	Version de JScript	Compatible ECMAScript ?
1.0	Non	1.0	Non
1.1	Oui	2.0	Non
1.2	Non	3.0	Oui
1.3	Oui	4.0	Oui
1.4	Oui	5.0	Oui

Versions Jscript :

Jscript est un langage poussé par Microsoft àl'intérieur d'un concept beaucoup plus global , dénommé WSH, àsavoir Windows Scripting Host

Les dernière informations étant disponibles sur le site MSDN àl'adresse suivante :

http://msdn.microsoft.com/scripting

Windows Script 5.5

Windows 95, Windows 98, Windows NT 4.0 and Windows 2000 - US English Version

Last Updated: 07/12/2000

The following files will install Windows Script containing Visual Basic Script Edition (VBScript.) Version 5.5, JScriptR Version 5.5, Windows Script Components, Windows Script Host 2.0, and Windows Script Runtime Version 5.1. If you are looking an older version of Microsoft Windows Script, you can also download version 5.1.

Le tableau suivant indique avec quels produits chaque version de JScript a été livrée jusqu'à présent. Tous ces produits sont commercialisés par Microsoft :

	Numéro de version				
Application	**1.0**	**2.0**	**3.0**	**4.0**	**5.0**
IE3.0	•				
IIS 3.0		•			
IE4.0			•		
IIS4.0			•		
WSH 1.0			•		
Visual Studio 6.0				•	
IE5.0					•
IIS5.0 (Windows 2000)					•

Versions Javascript :

Javascript est la version développée par Netscape, est supportée par Micorsoft via Jscript

Javacript est disponible coté client, et coté serveur, mais uniquement sur les produits distribués par Nestcape…

Ce petit tableau essaye de rassembler de quelle manière la norme **JAVASCRIPT** est supportée nativement par les produits **NETSCAPE**, et via **JSCRIPT** par les produits Microsoft…

	Numéro de version				
Application	**1.0**	**1.1**	**1.2**	**1.3**	**1.4**
Navigator 2.0x	•				
IE 3.0x	•	à peu près			
Navigator 3.0x		•			
Enterprise Server 2.0		•			
Navigator 4.0 - 4.05			•		
Enterprise Server 3.0			•		
IE 4.0x			•		
Navigator 4.06 - 4.61				•	
IE 5.0x				•	à peu près
Navigator 5.0 « Gecko »					•

statmarket.com déclare le 9/05/1999

Browser	%
MSIE 4.x	44.14%
MSIE 5.x	23.86%
Netscape 4.x	22.78%
MSIE 3.x	3.63%
Netscape 3.x	3.05%
WebTV	1.41%

ANNEXE : RESSOURCES INTERNET

Normes et spécifications :

Si la norme est édictée par l'organisme de Normalisation ECMA, dont les dernière spécifications sont disponibles àl'adresse suivante :

Norme ECMA : **http://www.ecma.ch**

Spécifications Microsoft : **http://msdn.microsoft.com/scripting**

Spécifications Nestcape : **http://www.netscape**

Sites et Forum :

http://www.experts-exchange.com

http://www.editeurjavascript.com

http://www.allhtml.com