JavaScript

Table Des Matières

1 - Tour d'horizon

1.1 - Introduction

Sous ce nom, certes trompeur, se cache un langage qui n'a rien à voir avec le langage . Certains vous diront que JavaScript c'est du light, que c'est pour les pros et que JavaScript pour les débutants. En fait, JavaScript est un langage, Java en est un autre, chacun ayant ses propres objectifs et s'adressant tous deux à des programmeurs. Par contre, les bases de JavaScript reposent sur un autre langage de script : ECMAScript (bien que historiquement, ce soit l'inverse). Aujourd'hui, toutes implémentations de JavaScript se doit de respecter la spécification de ECMAScript (ECMA-262). Les raisons en sont simples : JavaScript trouvera sa place aussi bien du coté des serveurs WEB, que du coté des clients (les navigateurs). Les objectifs n'étant naturellement pas les mêmes, il fut nécessaire de définir un noyau minimal, pouvant être utilisé dans toutes les situations.
A l'heure actuelle, ce langage est utilisé comme complément au langage HTML, pour tout ce qui relève du traitement dynamique, c'est-à-dire de tout ce qui ne peut être résolu qu'au niveau du navigateur (et non du serveur). En effet, le code JavaScript, embarqué avec le code, ne sera traité que par le navigateur recevant le document. Il est aussi utilisé dans un autre cadre : l'administration des serveurs (dans ce cas des possibilités d'utiliser les entrées/sorties et les accès au WEB sont possibles).
Ainsi, grâce à JavaScript, vous pourrez changer le contenu d'une image, selon que vous soyez dans une zone de la fenêtre du navigateur ou dans une autre, afficher des boîtes de dialogue dans telle ou telle situation, contrôler la validité des données d'un formulaire contrôler l'historique des pages déjà visitées et une multitude d'autres choses.

1.1.1 - Historique

JavaScript, premier langage de script sur le web (il en existe d'autres aujourd'hui, comme VBScript), fut développé par la société Netscape*, et supporté dès la version 2.0 de leur navigateur. Très vite la société Microsoft*, l'adopta aussi (à partir de la version 3.0). Désormais, les nouvelles versions des deux grands navigateurs, supportent toutes les deux ce langage qui a bien évolué depuis. Toutefois, il faut faire attention : l'interprétation ou non du langage est une option paramétrable du navigateur (pour la suite de ce cours, vérifiez qu'elle soit bien activée).

1.1.2 - Survol rapide du langage.

- Les principales caractéristiques de ce langage sont les suivantes :
- quelques types de base : les nombres (pas de distinctions apparentes entre entiers et flottants), les chaînes de caractères, les booléens, le type référence nulle.
- langage faiblement typé : une variable peut à tout moment changer de type.
- syntaxe proche de celle du langage C : quasiment les mêmes opérateurs, les mêmes instructions.
- le langage est basé sur un modèle objets rudimentaire : il n'y a pas de notion de classe, pas d'héritage.
- quelques objets de bases : manipulation de dates, de tableaux, calcul mathématiques.
- pas de possibilités d'utiliser les entrées/sorties (sauf pour le clavier et l'écran), du moins du point de vue des clients. Pas d'accès au réseau non plus, du moins du point de vue des clients.

1.2 - Un mot sur les entrées/sorties

JavaScript s'en remet à l'hôte qui l'héberge pour tous ses besoins d'entrées/sorties. L'exemple suivant permet d'afficher la phrase « Bonjour tout le monde ! » sur une seule ligne à l'écran à l'intérieur d'une page HTML :

```
<HTML>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
    document.write('Bonjour le monde !\n');
</SCRIPT>
</BODY>
</HEAD>
```

1.3 - Les variables et les données

Vous pouvez stocker des informations dans les scripts JavaScript. En voici quelques exemples :

```
var nouvelle_valeur;
var cout = null;
var nouveau_prix = 10.95;
adresse = « 23 Petite Rue »;
var resultat = 'Resultat inconnue';
resultat = true;
resultat = 1.08e+23;
resultat = 0x2Fe3;
resultat = 0377;
nom =  Jimmy Zed ;
var premier = 1, second = 2, dernier = 10;
```

La déclaration des variables se fait par le mot clé var. Le type de la variable, lui, n'importe pas : une variable peut à un instant être d'un type, puis à l'instant suivant être d'un autre type. Le type de la variable est déterminé par l'affectation .

```
var toto="Coucou"; // toto est de type chaîne de caractères.
var toto=10;       // toto prend maintenant le type nombre.
var tata;          // tata est une variable dont le type n'a
                   // pas encore été spécifié
```

Remarques très importantes :

- le langage fait la différence entre les lettres majuscules et les lettres minuscules. En conséquence, le mot var et différent du mot Var, ou du mot VAR. Faites donc attention en écrivant les mots clés du langage. Pour la même raison, deux variables sont différentes si une seule des lettres diffère.
- on est pas obligé d'utiliser le mot clé var pour déclaré une variable car la première fois qu'une variable est utilisée, si elle n'est pas encore déclarée, elle le devient immédiatement. Toutefois, il est préférable d'utiliser le mot clé var.

1.4 - Les types primitifs

Les types primitifs sont de simples blocs de construction qui servent à l'assemblage de toutes les données en JavaScript.

1.4.1 - Types booléen, nombre et chaîne

Le type booléen (boolean) ne peut prendre que deux valeurs : true ou false.
Le type nombre (number) est utilisé pour tous les types de nombres. On ne distingue donc pas les entiers, les réels et les octets. La syntaxe utilisée est celle de C. Pour les entiers, plusieurs bases sont possibles : 10, 16 (0x10 ==16, 0X14 ==20) ou 8 (010 ==8, 01 ==1). Pour les réels, la notation exponentielle est possible. Les

nombres réels suivants sont bien construits : 3.141592654, -32.5 et 6.25E-3.
Le type chaîne (string) est utilisé pour les chaînes de caractères. Il n'existe pas de type pour représenter un seul caractère. comme en C, une chaîne de caractères peut commencer et finir par ". Mais on peut aussi utiliser une simple apostrophe ('). Pour déspécialiser les caractères " et ' on utilise le caractère backslash (\). Les caractères spéciaux de C restent disponibles (\n pour le retour chariot, \t pour une tabulation ...). Voici quelques exemples de chaînes correctement construites : "Essai", ">>\"<<" et 'A\tB'.

1.4.2 - Nombres particuliers, types null et indéfini

JavaScript reconnaît trois nombres particuliers dont les valeurs Infinity, -Infinity et NaN. De même, il existe deux types particuliers appelés Null et Undefined.

1.5 - Les tableaux

```
var totaux_mensuels = new Array(12);
totaux_mensuels[3] = 1999.95;
totaux_mensuels[4] = 'Pas de ventes';

var a=4;
document.write(totaux_mensuels[a]);
document.write(totaux_mensuels.length);
document.write(totaux_mensuels);

var jours = new Array("Dim","Lun","Mar","Mer","Jeu","Ven","Sam");
```

Pour déclarer un tableaux, on utilise une instanciation de l'objet Array. La première ligne déclare donc un tableau de 12 éléments. La dernière ligne déclare un tableau de sept éléments. Comme en C, on accède au différents éléments du tableau en faisant suivre au nom de la variable(sans espace), l'indice de l'élément entouré par des crochets. Notons l'existence de la possibilité de connaître à tous moments, le nombres d'éléments constitutifs du tableau. Soit tab une variable de nature tableau. Alors l'expressions tab.length vous rend le nombres d'éléments de ce dernier.
Dans un élément de tableau, on peut aussi stocker un autre tableau. Un tableau de tableau peut être considéré comme un tableau multidimensionnel, qui trouve de nombreuses applications notamment en mathématique :

```
var matrice = new Array(2);
matrice[0] = new Array(2);
matrice[1] = new Array(2);

matrice[0][0] = 1;
matrice[1][0] = 0;
matrice[0][1] = 0;
matrice[1][1] = 1;
```

1.6 - Commentaires et fin d'instructions

Une fois les données stockées, il faut être capable de leur appliquer des traitements.

1.6.1 - Les commentaires

Commenter un programme, c'est lui adjoindre des annotations non significatives pour son exécution. Malgré tout, ils ont une utilités certaines lorsque, après quelques temps vous relisez votre programme. En effet un programme, c'est bien plus que des mots mis côte à côte : chaque mot possède une sémantique (un sens). Il en résulte un comportement précis lors de l'exécution, et il n'est pas forcement aisé de saisir ce comportement dès la première lecture du code source. Les commentaires sont donc là pour vous faciliter cette tâche. Pour ce qui est de la syntaxe, vous avez deux possibilités héritées des langages

. En voici une brève description.

Mettre en commentaire la fin d'une ligne

Cette solution, la plus simple, consiste à placer deux fois consécutivement le caractère divise ('/'), souvent nommé slash. Tout ce qui suit, sur la ligne, ces deux caractères est alors considéré comme un commentaire. Faites attention à ne pas vouloir placer un commentaire alors que vous êtes en train de définir une chaîne de caractères : votre annotation serait alors incluse dans la chaîne.

```
var maVariable = 3 + 2 // un petit commentaire
// Pas de commentaires dans la ligne suivante
var taVariable = "toto // tata";
Mettre un commentaire sur plusieurs lignes
```

Cette deuxième solution devient utile quand le commentaire à placer devient important : il est alors difficile de le mettre sur une seule ligne. Pour marquez le début du commentaire, placer les deux caractères suivants (consécutivement) '/*'. Inscrivez ensuite votre annotation, puis replacez les deux caractères précédent dans l'ordre inverse ('*/') pour terminer votre commentaire.

```
/***********************************
*** Ceci est aussi un commentaire ***
```

```
**********************************/
```

1.6.2 - Fin d'une instruction

La fin d'une instruction est, comme en C, indiquée par un point-virgule. Cependant, JavaScript laisse le programmeur libre d'omettre ces points-virgules si une instruction semble naturellement être terminée :

```
var result;
result = 15    // pour ce test, donner à result la valeur 15
result = 5; result = 20;      // on change la valeur de result
                              // juste pour voir...

result
= 19;                 // la valeur finale de result est 19.

if ( result == 15 )
{
    /* Cet texte ne sera pas affiché puisque result ne
    vaut pas 15 */
    Clib.puts('La valeur de result est ')
    Clib.puts('encore 15')
}
else
{
    /* En fait, ce texte sera affiché puisque result ne
    vaut pas 15 ! */
    Clib.puts('La valeur de result a été ');
    Clib.puts('modifiée et ne vaut plus 15');
}
/* fin de l'exemple */
;
```

1.7 - Expressions et tests conditionnels

Une expression est soit une valeur (15, 16.25, mais aussi "chaîne", ...), soit le contenu d'une variable, soit le résultat d'une opération. Dans les deux premiers cas, rien de plus n'est à rajouter. Dans le dernier cas, le calcul peut être assez complexe : une opération étant constituée d'un opérateur qui prend en compte un certain nombre de sous-expressions (pouvant elles-mêmes être composées). Tous ces opérateurs admettent un ordre de priorité assurant une évaluation correcte de l'expression. Cependant si vous voulez être sûr de l'ordre d'évaluation des composantes de cette expression, ou si vous voulez fixer un autre l'ordre, il vous est possible des parenthèser les sous-expressions composants l'expression globale. Nous allons donc faire un petit tour des opérateurs qui sont à votre disposition.

1.7.1 - Les opérateurs numériques

Les opérateurs numériques prennent deux expressions numériques et rendent une valeur de type numérique. On y trouve l'addition (+), la soustraction (-), la multiplication (*), la division (/), et la fonction modulo (%). On trouve aussi des opérateurs qui agissent bit à bit : le ET (&), le OU (|), le XOR (^), le décalage à gauche (<<), vers la droite (>>). On trouve aussi des opérateurs numériques acceptant uniquement comme opérande, une variable numérique. On trouve notamment la fonction d'incrémentation (++) et celle de décrémentation (--) : dans les deux cas, l'opérateur peut être soit préfixé (l'incrémentation de la variable se fait avant l'évaluation de l'expression), soit postfixé (dans ce cas c'est un peu plus fin, on évalue d'abord, l'expression globale, puis ensuite on incrémentera la variable). Voici une expression numérique dont toutes les sous-expressions sont de type numérique : on remarquera l'utilisation des parenthèses pour forcer l'ordre d'évaluation des sous-expressions.

```
// une expression numérique
(b*b+9*p/q)/(4*c)
```

1.7.2 - Les opérateurs booléens

Une première catégorie d'opérateurs booléens permet de réaliser des comparaisons entre deux variables (deux opérandes), de même nature : deux nombres, deux chaînes ou deux booléens. Le résultat est alors un booléen (vrai ou faux). On en dénombre six : l'opérateur d'égalité "==", d'inégalité "!=", d'infériorité stricte "<", d'infériorité "<=", de supériorité stricte ">" et de supériorité ">=". Une autre catégorie d'opérateurs permet de réaliser des opérations entre booléens : la conjonction "&&" (les deux opérandes doivent être vraies), la disjonction "||" (une au moins, des deux opérandes, doit être vraie, sinon le résultat est faux), la négation "!" (c'est un opérateur unaire qui effectue la négation de l'opérande, forcément de type booléenne).

```
// une expression booléenne dont on voit pas trop le sens
(b==3 && c!=5) || (a==(b*b+9*p/q)/(4*c) || z<=10)
```

1.7.3 - Les opérateurs sur chaînes de caractères

La seule opération définie sur les chaînes de caractères est la concaténation et elle est dénotée par le symbole +. L'opérateur résultant prend deux opérandes de nature chaîne de caractères et rend une chaîne de caractères. Il est de plus possible de passer, comme seconde opérande, une valeur de type numérique ou booléenne. Dans ce dernier cas, c'est la chaîne de caractères décrivant la valeur qui est concaténée : ex "34" pour le nombre 34 ou "true" pour une valeur booléenne qui serait vraie.

1.7.4 - L'opérateur conditionnel

L'opérateur que nous allons maintenant décrire sert à retourner une expression ou une autre selon qu'une troisième (servant d'expression de test) soit vraie ou fausse. Cet opérateur est donc ternaire (il accepte trois opérandes, ici trois expressions). Il est donc logique d'avoir choisi deux caractères de séparation : '?' pour séparer la première de la seconde et ':' pour séparer la seconde de la dernière. Pour vous éclairer un petit peu plus, regardez les exemples qui suivent.

```
// Exemple 1 : si a vaut 5 alors on retourne l'expression
// de type booléen valant vrai, sinon on retourne une
// expression numérique complexe.
(a==5)?true:(a+b)*4

// Exemple 2 : on retourne le numéro d'une des quatre
premières
// lettres de l'alphabet contenues dans la chaîne toto
(toto=="a")?1:(toto=="b")?2:(toto=="c")?3:(toto=="d")?4:false
```

1.7.5 - Les opérateurs d'affectation

Nous allons maintenant voir une série d'opérateurs infixes un peu particuliers : ce sont les opérateurs d'affectation. En effet même si certains langages considèrent l'affectation comme une instruction, les langages issus de C eux la perçoivent comme une expression. La principale conséquence étant que l'affectation retourne un résultat. Ce dernier est en fait la valeur, et donc le type, qui est affectée. Plus généralement, une opération d'affectation recopie la valeur de l'opérande droite, et ce à un calcul près, dans la variable de l'opérande gauche (il ne peut s'agir que d'une variable, car elle seule pourra recevoir un résultat) et retourne cette valeur. L'opérateur le plus général est donc l'affectation classique (symbolisée par le caractère '='). L'opérande droite est simplement recopiée dans la variable nommée par l'opérande gauche. On voit de suite que l'on peut écrire a=b=c=10 : dans ce cas c reçoit 10, puis b reçoit le contenu de c (soit 10), puis a reçoit le contenu de b (soit toujours 10). Ce que l'on peut trouver de plus dans les langages issus de C, ce sont les affectations combinées. C'est-à-dire qu'on effectue d'abord un calcul entre le contenu de la variable, décrite par l'opérande gauche, et celle de droite, puis on affecte le résultat du calcul à la variable. On a notamment les opérateurs suivants : += -= *= /= %= &= |= ^= <<= >>=. Pour mieux comprendre ce qui se passe regardez les équivalences suivantes.

```
a+=10        <=>        a=a+10
b<<=10       <=>        b=b<<10
// Vous trouverez de suite les autres équivalences.
```

Dernière remarque, l'opérateur '+=' est bien entendu aussi défini sur les chaînes de caractères.

1.7.6 - L' opérateur de séquencement

Il peut être utile de diviser le calcul d'une expression en plusieurs sous-expressions. Mais supposons qu'on ne puisse exécuter qu'une seule expression à l'endroit clé (par exemple la sous-expression test d'une expression conditionnelle). Dans ce cas vous pouvez utiliser l'opérateur de séparation d'expression. La valeur de l'expression globale est celle de l'évaluation de la dernière expression. Regardez l'exemple suivant.

```
var a;
var b;
var Result = (a+=x,b=y*z,a*b==3?variable1:variable2);
```

1.7.7 - Précédence des opérateurs

Enfin, je pense qu'il est nécessaire de signaler qu'il n'est pas utile de parenthéser vos expressions à tous vents. Des règles de priorité existent : utilisez-les! Pour information, voici un petit tableau rappelant la priorité des opérateurs : ceux qui sont sur une même ligne sont de priorité identique, ceux du haut étant plus prioritaires que ceux du bas.

```
[ ] .
! ++ --
* / %
+ -
<< >>
== != < <= > >=
&
^
```

```
|
&&
||
?:
= += -= *= /= %= <<= >>= &= ^= |=
,
```

1.7.8 - L'opérateur Typeof

Typeof est utilisé pour identifier le type d'un élément. Etant donnée une variable ou une expression, il renvoie une chaîne de caractères décrivant son type :

```
var a = 'n_importe_quoi';
var b;
document.write(typeof(a));          // affiche "string"
document.write(typeof(b));          // affiche "undefined"
document.write(typeof(c));          // affiche "undefined"
```

1.8 - Contrôle de flux

Pour ceux qui connaissent déjà les langages, cette partie risque de ne pas être des plus intéressantes (la syntaxe et la sémantique restent très similaires). Quant aux autres, ils vont pouvoir s'apercevoir que ce n'est pas si compliqué qu'il pourrait y paraître.
La première règle : de syntaxe (que nous avons déjà évoquée) est très simple : toute instruction se termine par un point-virgule (;), et ce même s'il s'agit de la dernière instruction d'un bloc (certains autres langages permettent à ce moment là de l'omettre). Le point-virgule joue le rôle d'opérateur de séquencement d'instructions.
Seconde règle : toute expression est aussi une instruction. Ainsi l'instruction suivante est valide et calcul une valeur qui est finalement oubliée : "3*a+4*b;". Si vous cherchez une raison à ce choix, pensez qu'il vous faudra bien affecter des valeurs à des variables (or l'affectations est une expression ...), et n'oubliez pas qu'un programme s'écrit avec des instructions.

1.8.1 - Les blocs d'instructions : { instruction ... instruction }

Les blocs d'instructions sont très utiles : en effet on peut être amené à certains moments à faire un traitement qui nécessite plusieurs instructions. Or la syntaxe de certaines instructions de contrôle ne permettrait pas de faire la distinction entre ce qui rentre dans le cadre du traitement et ce qui succède le traitement. L'introduction des blocs d'instructions solutionne le problème. La syntaxe utilisée est relativement simple : on introduit un bloc par une accolade ouvrante, on fait suivre toutes les instructions qui sont nécessaires (sans oublier un point-virgule à la fin de chacune d'entre elles) puis on termine le bloc par une accolade fermante. Pour la suite de ce chapitre, nous définissons un Statement (mot anglais) comme étant soit une instruction de base, soit un bloc d'instructions.

```
// Ce qui suit est un bloc de trois instructions
{
  a=2*i+4;                                  // première
  for (var i=0;i<a;i++) fonction(i); // seconde (composée)
  while(i>0) { fonction2(); i--; }   // troisième
}
```

1.8.2 - L'instruction conditionnelle : if (condition) Statement1 else Statement2

Cette instruction, qui se décompose en trois parties essentielles, permet de lancer un traitement selon qu'une condition soit remplie ou non. La première partie de l'instruction, qui doit obligatoirement être parenthèsée, est la condition. En fonction que le résultat de cette condition soit vrai ou faux, un et un seul des deux Statements possibles sera lancé : le premier si le test est vrai, le second dans l'autre cas.

```
if (var_de_test == calcul(3)) {
  // premier statement : c'est un petit bloc
  b++;
  c--;
} else a+=d;  // second statement
```

Une petite remarque : le else et le statement qui suit sont facultatifs. Ils peuvent donc ne pas apparaître. Dans ce cas là, si la condition n'est pas remplie, aucune action ne sera générée, et l'exécution du programme passera à l'instruction qui suit le if :

```
if ( x == 5 )                 // premier exemple
    y = 6;

if ( name == "fred")          // second exemple
    y = 6;
else
    y = 7;

if ( x == y && a == b )              // troisième exemple
```

```
{
     x++;
     y++;
}
else
{
    a++;
    b++;
}
```

1.8.3 - Boucle For : for(initialisation; rebouclage; mise_a_jour) statement

Cette instruction sert à réaliser un traitement dit itératif, c'est-à-dire que le statement sera traité un certain nombre de fois. Pour réaliser cette boucle, l'instruction admet trois parties, ou expressions, (toutes facultatives) qui sont l'initialisation, le rebouclage et la mise_a_jour. La première sert à initialiser les variables qui permettent la réalisation de la boucle : son code sera donc exécuté une et une seule fois. La seconde sert à déterminer si l'on doit encore faire un passage dans le traitement du statement ou non. Bien naturellement, cette condition est réévaluée à chaque tour de boucle. La dernière, mise_a_jour, permet la mise à jour des variables utilisées pour réaliser la boucle.

```
// Voici un petit exemple d'utilisation du for
var b=0;
for(var i=1;i<10;i++) {
  // Step 0 : on initialise i à 1
  // Step 1 : si i>9 alors on sort du if
  // Step 2 : on ajoute i à b
  // Step 3 : on incremente i d'une unité
  // Step 4 : on repart en Step 1
  b+=i;
}
// la somme des 9 premiers entiers vaut : b
```

```
var fruit = Array(3);
fruit[0] = 'pomme';
fruit[1] = 'poire';
fruit[2] = 'orange';
for (count=0; count <=2; count++)           // affiche tous les
fruits.
    document.write(fruit[count]+ ' ');

var new_line = '<BR>';
for (loop1=0; loop1 < 5; loop1++)           // dessine un triangle
de T
{
    for (loop2=0; loop2 < loop1; loop2++)
    {
          document.write('T');
    }
    document.write(new_line);
}

for (;;)                                    // boucle indéfiniment, sans
rien faire.
       ;
```

1.8.4 - Boucle while : while(condition) statement

Cette instruction réalise aussi une boucle. En fait, la boucle while peut être vue comme une boucle for à laquelle on aurait enlevé les expressions initialisation et mise_a_jour (et réciproquement). Seule l'expression dite de condition, reste utilisable. La boucle while et donc plus générale que celle obtenue par l'instruction for. L'exemple qui suit réalise la même chose que l'exemple précédent afin de vous faire sentir les différentes subtilités qu'il peut y avoir entre les deux versions de la boucle en JavaScript.

```
// Voici un petit exemple d'utilisation du while
var b=0;
var i=1;       // On crée un variable utile pour la boucle
while(i<10) {  // tant que celle-ci est inferieure à 10
  b+=i;        // on fait une partie du calcul
  i++;         // on augmente la variable de boucle d'une unité
}
// la somme des 9 premiers entiers vaut : b
```

```
var num = 134 - 1;    // exemple 1. Trouver le plus grand
// diviseur de 134.
var finished = false;

while ( finished == false )
{
    if ( 134 % num == 0 )
    {
          document.write('Le plus grand diviseur de 134 est ' + num);
          finished = true;
    }
    num--;
}

num = 1;       // exemple 2. Affiche tous les nombres (boucle
infinie).
while (true)
{
    document.write(++num + ' ');
}
```

1.8.5 - L'instruction continue

Cette instruction s'utilise uniquement à l'intérieur d'une structure de boucle et permet de sortir du statement, et ce sans terminer le traitement en cours, et de reprendre un nouveau tour de boucle. Ceci peut être utile, par exemple, pour avorter un traitement selon qu'une condition soit remplie ou non. L'exemple qui suit sera peut être plus explicite que mes explications.

```
for(var i=1;i<11;i++){
  if (i==6) continue;   // Si i=6 alors on ne fait pas de traitement
                        // mais on continue les autres étapes !!!
  AppelFonction();      // On appelle une fonction
}

// il résulte que cette boucle fait seulement 9 appels à la
// fonction, car le sixième a été oublié
```

```
var loop = 0;  // Exemple 1. Affiche les premiers multiples de 7.

while ( ++loop < 200 )
{
    if ( loop % 7 != 0 )
         continue;
    document.write(loop + ' ');
}
```

1.8.6 - L'instruction break

Comme l'instruction précédente, celle-ci s'utilise dans le statement d'une boucle. De même, elle permet d'interrompre le traitement en cours, mais à l'inverse de la précédente, l'exécution ne se poursuit pas avec la prochaine itération, mais directement à l'instruction qui suit celle de la boucle. Cela permet d'interrompre brutalement une boucle, si c'est nécessaire. L'exemple suivant donne une autre version pour le calcul de la somme des 9 premiers entiers.

```
var b=0;
var i=1;       // On crée un variable utile pour la boucle
while(true) {  // on boucle à l'infini car toujours vrai
  if (i==10) break; // si i=10 on arrête tout !
  b+=i;        // on fait une partie du calcul
  i++;         // on augmente la variable de boucle d'une unité
}
// la somme des 9 premiers entiers vaut : b
```

```
var loop = 0;         // Exemple 2. Trouve le premier nombre plus
// grand que 1000 et multiple de 99

for (loop=1000; loop < 1099; loop++)
{
    if (loop % 99 == 0)
         break;
}
document.write(loop);
```

1.8.7 - Les fonctions : function nom(paramètres){corps}

Comme dans tous langages informatiques dignes de ce nom, il vous est possible de déclarer des fonctions, et de faire des appels à celles-ci. L'intérêt d'une telle possibilité est très simple : si votre programme a besoin, et ce à plusieurs reprises, d'une même portion de code (aux valeurs des variables près), vous allez pouvoir localiser, en un seul point du code, cette section et y faire appel, là où cela est nécessaire, avec des valeurs quelconques (elles seront alors nommées paramètres de la fonction). L'avantage majeur étant que désormais vous n'aurez qu'une seule modification à y apporter si vous avez commis une erreur (ce qui est fort agréable).
Pour introduire une définition de fonction, on utilise le mot clé function. Le nom que vous donnez à la fonction est très important : il sera utile lorsque vous voudrez faire appel à celle-ci (il sera donc judicieux de trouver des noms appropriés pour vos fonctions, histoire que vos programmes soient simples à relire et à comprendre). Ensuite, il vous faut spécifier les paramètres utilisés par la fonction. Il peut bien sûr ne pas y en avoir (dans ce cas, refermez directement la parenthèse). Sinon il vous suffit juste de donner les noms de ces paramètres séparés les uns des autres par une virgule. Faites tout de même attention : l'ordre que vous avez donné à ces paramètres doit être respecté lorsque vous passerez les valeurs requises lors de l'appel de la fonction. Autre petite chose très utile : une fonction peut rendre un résultat. Dans ce cas, la valeur d'un appel à la fonction sera la valeur retournée. Pour ce faire, vous avez à votre disposition le mot clé return. S'il n'est pas suivi d'une expression, alors il indique que l'exécution de la fonction s'arrête à sa position. Sinon, le programme s'arrête aussi, mais rend la valeur de l'expression spécifiée. Notez bien que dans tous les cas, si un return est rencontré, l'exécution de la fonction s'arrête et le programme se poursuit sur l'instruction qui suit l'appel à la fonction.
Pour y voir plus clair, regardez l'exemple qui suit. Celui-ci part de l'hypothèse que le bout de code qui calcule la somme de nos 9 premiers entiers est utilisé à plusieurs reprises. Vous en faites donc une fonction qui ne prend aucun paramètre

(toutes les données relatives à la solution du problème sont connues d'avance).

```
// On définit notre fonction très utile dans le programme :
function Sigma9(){
  var b=0;
  var i=1;
  while(i<10) b+=i, i++;
  return b;
}

...  // Plein de lignes de codes

// la somme des 9 premiers entiers vaut : Sigma9()
```

Considérons maintenant que l'on veuille généraliser la fonction pour qu'elle puisse calculer la somme des i premiers entiers (ou i est une valeur que vous fixez à l'appel de la fonction selon vos besoins). Dans ce cas la valeur i doit être passée en paramètres. Voici le code résultant.

```
// Voici notre petite fonction
function Sigma(i){
  var b=0;
  while(i>0) b+=i, i--;
  return b;
}
...  // Plein de lignes de codes
// la somme des 10 premiers entiers vaut : Sigma(10)
```

Pour information, s'il vous prenait l'idée de vouloir définir une fonction à l'intérieur d'une fonction, sachez que la spécification actuelle de JavaScript ne le permet pas. Malgré tout, l'interprète de Netscape accepte ce genre de choses. Dernière petite remarque, de manière générale, les définitions de fonctions se font dans le couple de

marques HTML <HEAD> ... </HEAD>
, par l'utilisation des marques

<SCRIPT> ... </SCRIPT>
. Mais ceci n'est qu'une convention, vous pouvez très bien ne pas la respecter.

Un autre exemple :

```
function dire_bonjour()
{
    document.write('Bonjour. ');
    document.write('Bienvenue dans la fonction dire_bonjour(). ');
    return;
}

function ajouter(premier, second)
{
    var resultat = premier + second;
    return resultat;
}

function ajouter_tous()
{
    var boucle=0, somme=0;
    for ( boucle = arguments.length-1; boucle >=0; boucle--)
        somme += arguments[boucle];
    return somme;
}

// Maintenant, servons-nous de ces fonctions

dire_bonjour();
var reponse = ajouter(5, 3);
document.write( reponse + ' ' + ajouter(10, 4) );
document.write( ' ' + ajouter_tous(1, 3, 5, 7, 9) );

Qui produira l'affichage :
Bonjour. Bienvenue dans la fonction Dire_Bonjour(). 8 14 25
```

Et encore un :

```
var truc = 'dehors';

function magie()
{
    var truc = 'dedans';
    document.write(truc);
}

document.write(truc);                  // affiche 'dehors'
magie();                                       // affiche 'dedans'
```

1.8.8 - Quelques notions sur la récursivité

Comme tout bon langage qui se respecte, JavaScript permet de programmer des fonctions dites récursives. Derrière ce mot barbare se cache une méthode de programmation qui, dans bien des cas, permet de résoudre plus simplement le problème (ceci sous-entend que vous ayez bien assimilé le processus).
On dit qu'il y a un appel récursif dans une fonction, si celle-ci se fait appel pour

calculer une sous-partie du problème. Il y a aussi récursivité si la fonction s'appelle indirectement (c'est-à-dire qu'elle fait appel à une fonction qui fera, peut-être elle-aussi indirectement, appel à la fonction de départ). Dans tous les cas, lors de l'exécution de la fonction à un niveau de profondeur donné d'appel, il n'y aura pas écrasement des valeurs des variables (car le passage des paramètres se fait par recopie des valeurs).
Pour y voir plus clair regardez le programme suivant. Je pense que vous aurez deviné qu'il va faire la somme des i premiers entiers (comment avez-vous deviné ???).

```
// Voici notre petite fonction
// si i vaut 1 alors la somme du premier entier vaut 1
// sinon elle vaut la somme de i + la somme des i-1 premiers entiers
// Très simple et très efficace !!!
function Sigma(i){
  return i==1?1:i+Sigma(i-1);
}

...  // Plein de lignes de codes

// la somme des 10 premiers entiers vaut : Sigma(10)


// Une fonction qui calcule un factoriel !!!
function Factoriel(i){
  return i<=1?1:i*Factoriel(i-1);
}

...  // Plein de lignes de codes

// le factoriel de 10 vaut : Factoriel(10)
```

1.9 - Les objets

Certains programmeurs sont plus effrayés par la notion d'objet que par tout autre. Heureusement, les objets JavaScript sont simples à comprendre.

1.9.1 - Qu'est-ce qu'un objet ?

D'une manière générale, il s'agit du regroupement d'un certain nombre de variables et fonctions à l'intérieur d'un ensemble organisé qui pourra facilement être réutilisé.
Les défenseurs d'autres langages orientés objets risquent de ne pas apprécier cette définition simpliste des objets, mais comme JavaScript n'impose pas une grande rigueur dans l'écriture des scripts, il s'adapte parfaitement à la définition d'objets simples.
Voici un exemple d'objet JavaScript :

```
var chose = new Object;

chose.name  = 'Une vieille chose';
chose.age   = 59;
chose.hobby = 'évoluant le plus lentement possible.';

chose.chose2 = new Object;
chose.chose2.name = "une chose qui appartient à notre vieille chose";

delete chose;
```

La première ligne permet de créer un objet nommé chose. Pour rendre cet objet utile, il faut lui adjoindre des propriétés. Les lignes 3 à 5 montrent à quel point il est simple de le faire :
Nom_de_variable_objet.nom_de_propriété = valeur;

Les objets peuvent posséder autant de propriété qu'on veut du moment que l'on est un nom différent pour chacune.

1.9.2 - Les méthodes

Les habitués d'autres langages de programmation peuvent déjà être familier des concepts de structures, d'enregistrement et de formulaires. Un objet qui ne possède que des propriétés est comparable à ce type de structure : ce ne sont que des données regroupées sous une même désignation.
Cependant, les objets peuvent aussi posséder des méthodes, comme dans l'exemple suivant :

```
function affichage_de_bonjour_standard() {
document.write("Bonjour."); }

function affichage_special_du_nom()
{
     document.write("Mon nom est : " + this.nom);
}
var chose = new Object;
chose.nom = "Fred";
chose.dire_bonjour = affichage_de_bonjour_standard;
chose.afficher_nom = affichage_special_du_nom;

chose.dire_bonjour();
chose.afficher_nom();
```

Les deux fonctions définies dans l'exemple sont apparemment normales, même si

la deuxième fait appel à une variable mystérieuse nommée this. Vers la fin du script, deux propriétés de l'objet chose référencent les fonctions définies auparavant.

```
chose.dire_bonjour = affichage_de_bonjour_standard;
chose.afficher_nom = affichage_special_du_nom;
```

Ainsi, tout à la fin du script, les propriétés de l'objet commencent à ressembler à des fonctions :

```
chose.dire_bonjour();
chose.afficher_nom();
```

Une méthode est simplement une fonction contenue dans un objet. On arrive à ce résultat en utilisant l'une des propriétés de l'objet pour référencer une fonction du script, de la même façon qu'une variable peut garder la trace d'un objet complet.
Lorsqu'une méthode est appelée, la variable spéciale this est définie à l'intérieur de la méthode. this est en fait une référence vers l'objet qui a invoqué la méthode et permet d'accéder directement aux autres propriétés de cet objet.
Ainsi, les objets sont réellement des regroupements de propriétés et de méthodes.

1.9.3 - Prototypes et constructeurs

Soit l'exemple suivant :

```
function afficher()
{
    document.write(this.nom);
}

function Employe(nom, age)
{
    this.nom = nom;
    this.age = age;
    this.status = 'Plein temps';
}

Employe.prototype.adresse = 'JS Industries, Station St.';
Employe.prototype.contact = '012 345 6789';
Employe.prototype.job = "Semble indispensable";
Employe.prototype.affiche = afficher;

var personne1 = new Employe("Doe, John", 47);
var personne2 = new Employe("Doe, Jane", 45);

personne1.affiche() + document.write(" " + personne2.contact);
```

Les variables personne1 et personne2 sont créées à partir du constructeur d'objet Employe. En fait, un constructeur est une fonction qui permettra d'initialiser des propriétés et des méthodes d'un objet à l'aide de paramètre passés au constructeur lors de la création. Dans l'exemple, la fonction constructeur nommée Employe demande deux arguments pour initialiser les propriétés nom et age de l'objet en création. La propriété status reçoit une valeur fixe, elle aurait pu être mise en prototype.
La propriété prototype associée aux fonctions spéciales constructeurs permet d'initialiser des propriétés et des méthodes avec des valeurs communes à tous les objets lors de leur création. Dans l'exemple précédent, tout objet créé avec le constructeur Employe, se verra attribué trois propriétés (adresse, contact, job) et une méthode (affiche).

1.9.4 - L'instruction For … in …

Il arrive que l'on ne connaisse pas le contenu d'un objet. Pour pouvoir parcourir le contenu d'un objet sans pour autant le connaître, nous disposons de l'instruction suivante :

```
for (variable in varaible_objet)
      instruction ou bloc
```

Un exemple :

```
for (prop in objet_quelconque)
{
  if (typeof(objet_quelconque[prop])== « chaine »)
    document.write(« Propriété : »+prop+« Valeur : »
                                          +objet_quelconque[prop]) ;
}
```

Cet exemple parcours toutes les propriétés d'un objet quelconque et affiche uniquement les propriétés contenant une valeur de type chaîne.

1.9.5 - L'instruction With …

Nous avons vu que chaque variable faisait partie d'un objet que nous pourrions appeler contexte courant, objet courant ou portée courante. Parfois, pour faciliter l'écriture d'un script, nous pouvons essayer de changer la portée courante. L'utilisation de with le permet. Sa syntaxe est la suivante :

```
with (objet)
       instruction ou bloc d’instructions
```

Un exemple :

```
var nom_incroyablement_long = new Object;
nom_incroyablement_long.seule_propriete = new Object;
nom_incroyablement_long.seule_propriete.un = "A vos marques";
nom_incroyablement_long.seule_propriete.deux = "Prets";
nom_incroyablement_long.seule_propriete.trois = "Partez";

with ( nom_incroyablement_long.seule_propriete )
{
    document.write( un + ', ' + deux + ', ' + trois +'!');
               // A vos marques, prets, partez !
}
```

2.1 - Ajouter du code JavaScript à un document HTML

JavaScript peut être implanté dans une page HTML de deux façons :
par la balise SCRIPT directement, ou en mettant le code dans un fichier séparé, en utilisant les événements.

```
<HTML>
<HEAD>

<SCRIPT>
  var step=1;

// exemple 1.
</SCRIPT>

// exemple 2.
<SCRIPT LANGUAGE="JavaScript"> step=2; </SCRIPT>

// exemple 3.
<SCRIPT LANGUAGE="JavaScript1.1"> step=3; </SCRIPT>

// exemple 4.
<SCRIPT LANGUAGE="JavaScript1.2"> step=4; </SCRIPT>

// exemple 5.
<SCRIPT SRC="mesfonctions.js"> </SCRIPT>

</HEAD>
</HTML>
```

2.1.1 - En utilisant la balise <SCRIPT>

```
L'exemple 1 montre qu'en fait, le langage JavaScript est le langage
par défaut des scripts. Cependant, les exemples 2, 3 et 4, nous
montre que JavaScript n'est pas le seul langage de script.

<SCRIPT LANGAGE="JavaScript">
  code
</SCRIPT>
```

On le voit, le choix du langage ne se limite pas à JavaScript et reste ouvert à d'autres langages comme Visual Basic, que supporte Internet Explorer.
Le code placé à l'intérieur d'un couple de balises SCRIPT est évalué après le

chargement de la page.
Les fonctions sont définies dans les couples de balises SCRIPT dans la partie entête (HEAD) de la page HTML ou même dans le corps de la page, ce qui n'est cependant pas conseillé.
La balise SCRIPT peut prendre un argument : LANGAGE="JavaScript1.1" qui indique que le code JavaScript est de version 1.1, actuellement compris par les versions de Netscape 3.x et supérieure et par les versions Microsoft Internet Explorer 4 et supérieures.
L'argument LANGAGE="JavaScript1.2" indique que le code JAVASCRIPT est de version 1.2, actuellement compris par les versions de Netscape 4.x et supérieure.
Pour utiliser dans une page un code JavaScript à la fois compatible 1.0 et 1.1 il suffit de déclarer les procédures deux fois comme le montre l'exemple suivant :

```
<SCRIPT LANGUAGE="JavaScript">
  code
</SCRIPT>
<SCRIPT LANGUAGE="JavaScript1.1">
</SCRIPT>
Ceci peut être obtenu également par l'astuce suivante

<SCRIPT LANGUAGE="JavaScript1.1">
location.replace("URL de la page HTML version 1.1")
code compatible 1.0
</SCRIPT>
```

Ceci peut se décliner par des variantes du style

```
<SCRIPT LANGUAGE="JavaScript">
  if (navigator.userAgent.indexOf("3.0") != -1)
    jsVersion = "1.1" else jsVersion = "1.0"
</SCRIPT>
```

La variable jsVersion servira alors tout au long des programmes à établir un code compatible avec la version du navigateur.

Une autre astuce, pour connaître le numéro de version de JavaScript est donnée par le programme suivant :

```
<HTML><HEAD>
<SCRIPT LANGUAGE="JavaScript">var v = 1.0</SCRIPT>
<SCRIPT LANGUAGE="JavaScript1.1">v = 1.1</SCRIPT>
<SCRIPT LANGUAGE="JavaScript1.2">v = 1.2</SCRIPT>
</HEAD><BODY><SCRIPT>
document.write(v);
</SCRIPT></BODY></HTML>
```

Le couple de balises <NOSCRIPT> </NOSCRIPT> permettent d'encadrer le texte qui sera affiché si JavaScript n'est pas compris par le navigateur.

```
<HTML><BODY><SCRIPT>
  alert("Ouvrir la page JavaScript ?");
  document.location.href = 'js_top.htm';
</SCRIPT>
<NOSCRIPT>Offrez-vous un vrai navigateur !</NOSCRIPT>
</BODY></HTML>
```

Pour cacher le contenu d'une section SCRIPT pour un navigateur ancien ou non compatible, on utilise la mise en commentaire suivante :

```
<HTML>
 <HEAD>
  <TITLE>Notre premier script</TITLE>
  <SCRIPT LANGUAGE="JavaScript">
   <!--masque le script aux navigateurs anciens
   // Nous n'avons pas de fonctions ou de variables à déclarer,
   // mais il aurait fallu les placer ici...
   // fin du masque destiné aux navigateurs anciens -->
  </SCRIPT>
 </HEAD>
 <BODY>
  <SCRIPT LANGUAGE="JavaScript">
   <!--masque le script aux navigateurs anciens
   document.write('Bonjour le monde !<BR>');
   // fin du masque destiné aux navigateurs anciens -->
  </SCRIPT>
 </BODY>
</HTML>
```

2.1.2 - Le code dans un fichier source

Comme le montre l'exemple 5 du début de chapitre, à partir de la version 3 de Netscape, il est possible de mettre le code des programmes JavaScript dans un fichier annexe, en utilisant la balise SCRIPT comme suit :

```
 <SCRIPT LANGUAGE=JavaScript SRC=source.js> </SCRIPT>
```

source.js peut être un fichier dans le répertoire courant ou bien une URL pointant vers un fichier distant.
Du code inséré avant la balise </SCRIPT> ne sera exécuté que si le fichier source.js n'a pu être chargé.
Mais cette option, outre le fait qu'elle n'est supportée que par les version 3 de Netscape nécessite que le serveur HTTP, soit configuré avec un type MIME "application/x-JavaScript" en regard des fichiers d'extension js.

2.1.3 - En utilisant les événements

Les événements sont les résultats d'une action de l'utilisateur, comme par exemple un clic sur l'un des boutons de la souris.
La syntaxe de ces événements est :

```
<balise eventHandler="code JavaScript">
```

où balise est le nom d'une balise et eventHandler est le nom d'un événement.
Par exemple, pour utiliser la fonction exemple quand un utilisateur appuie sur un bouton, l'instruction suivante

```
<INPUT TYPE=BUTTON VALUE=Essai onClick="exemple(this.form)">
```

Une fonction ou bien un code JavaScript peut être inséré comme valeur de l'argument exemple
Bien évidemment, il est plus intéressant d'utiliser une procédure, lorsque le code employé est utilisé plusieurs fois.
Voici la liste des événements disponibles :
Chaque événement est codé sous une des deux formes :

2.1.4 - Avec formulaire

```
<FORM>
<INPUT VALUE=événement
événement=alert("événement")>
</FORM>
```

2.1.5 - Avec hypertexte

```
<A HREF=#EX  événement=alert("événement")>événement</A>
```

onBlur : l'utilisateur après avoir cliqué sur une zone clique à l'extérieur de la zone active.

onClick : l'utilisateur clique sur une zone hypertexte.

onChange : l'utilisateur après avoir cliqué sur une zone, la quitte après avoir changé le texte.

onFocus : l'utilisateur clique sur une zone.

onLoad : l'utilisateur charge la page dans le navigateur

onMouseOver l'utilisateur passe la souris sur la zone

onSelect : l'utilisateur sélectionne un élément d'un formulaire

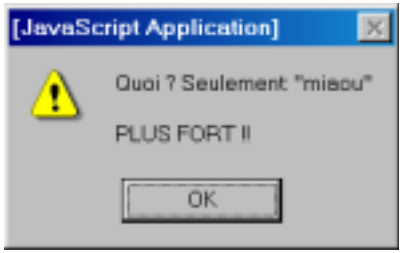
onSubmit : l'utilisateur soumet un formulaire

onUnload : l'utilisateur quitte la page

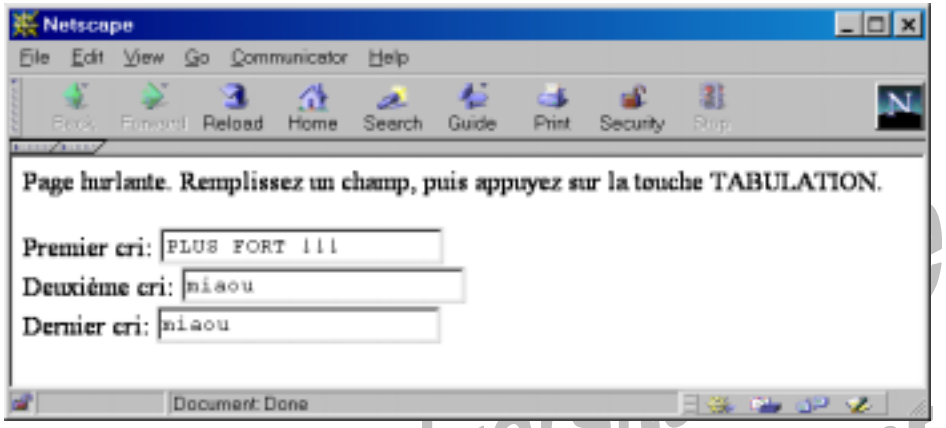
Exemple :

```
<HTML><HEAD><SCRIPT>
var cris = 0;
function crier(obj)
{
    alert('Quoi ? Seulement: "' + this.value + '"\n\nPLUS FORT !!');
    cris++;
}
</SCRIPT></HEAD>
<BODY>
Page hurlante. Remplissez un champ, puis appuyez sur la touche
TABULATION.<BR>

<FORM NAME="crieur"> Premier cri:
<INPUT TYPE="text" NAME="un" ONCHANGE="this.value='PLUS FORT
!!!!';return true;">
<BR> Deuxième cri:
<INPUT TYPE="text">
<BR> Dernier cri:
<INPUT TYPE="text" NAME="trois">
</FORM>

<SCRIPT>
document.crieur[1].onchange=crier;
document.crieur.trois.onchange=crier;
</SCRIPT></BODY></HTML>
```

L'exemple précédent donne l'affichage suivant après le remplissage du deuxieme cri :



L'exemple précédent donne l'affichage suivant après quelques utilisation.



En fait, on peut remarquer que le premier INPUT prend la valeur « PLUS FORT ! ! ! »dèq que l'on quitte l'objet et que son contenu à changé.
Ensuite, pour les deux autres INPUT, leur événement onChange fait que lorsque l'on quitte l'objet après avoir changer sont contenu, une boite de message apparaît avec comme texte le contenu de l'objet INPUT que l'on vient de quitter :

2.2 - Les événements temporels :

La fonction JavaScript setTimeout() permet d'exécuter un script au bout d'un certain temps. L'exemple suivant montre l'utilisation de cette fonction afin d'afficher une boit d'alerte toute les deux secondes :

```
<HTML><HEAD><SCRIPT>
function reveiller()
{
    // Essayons de réveiller l'utilisateur.
    alert('Incendie, Inondation, Famine !');
    setTimeout('reveiller()', 2000);
}
// 2000 millisecondes dans le futur, on réveille
setTimeout('reveiller()', 2000);
</SCRIPT></HEAD></HTML>
```

2.3 - Des scripts dans des documents HTML

Voici deux exemples de scripts JavaScript :

```
<HTML>
  <HEAD>
    <SCRIPT>
      function meteo()
      {
        if ( !Math.random )          // n'existe pas dans Navigator
2.0
        {
          document.write('<PRE> -- le temps est perturbé par la
pluie --</PRE>');
        }
       else if ( Math.floor((Math.random()*2)) == 0 )
       {
         document.write("<STRONG>C'est simplement
épouvantable.</STRONG>\n");
       }
       else
       {
          document.write("<EM> C'est merveilleux ! </EM>\n");
       }
      }
    </SCRIPT>
  </HEAD>
  <BODY>
    <P>Prévisions météo pour la journée:</P>
    <SCRIPT>
      meteo();                       // ajout de choses spéciales
    </SCRIPT>
    <P>Fin des prévisions.</P>
  </BODY>
</HTML>
```

Dans cet exemple, une fonction meteo est déclarée. Ensuite, un appel à cette fonction est fait dans le déroulement de la page HTML.
Dans l'exemple suivant, on crée une page complète, à l'aide d'un script JavaScript :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<SCRIPT>
   with ( document )
   {
     write('<HTML><HEAD>\n');
```

```
    write('<TITLE>Pratiquement tout en JavaScript !</TITLE>\n');
    write('</HEAD><BODY>\n');
    write('Qui a besoin d'HTML statique ?\n');
    write('<BR><HR></BODY></HTML>');
  }
</SCRIPT>
```

Ceci n'est pas très orthodoxe, mais tout à fait supporté par les navigateurs. D'une manière générale, il est préférable d'utiliser les balises SCRIPT dans la partie HEAD d'un document HTML pour toutes les déclarations.

2.4 - Les entités JavaScript

Un nom entité, en HTML, est un élément syntaxique de la forme &NomEntité; , qui permet de représenter des caractères difficiles à symboliser. C'est ainsi que l'on représente un espace insécable par , le caractère « < » par &lt, et le caractère accentué « é
par é .Les entités JavaScript sont des extensions de cette syntaxe, qui se présente sous la forme &{VariableJavaScript}; ou &{ExpressionJavaScript}; .Par exemple :

```
<HTML><HEAD>
<SCRIPT>
var pixels_par_unite = 25.4;
var total_unites = 8;
</SCRIPT>
</HEAD><BODY>
```

Graphique sous forme de barre représentant le nombre d'unités :

```
<HR ALIGN=LEFT SIZE=10 WIDTH="&{pixels_par_unite *
total_unites};"><BR>
```

Graphique sous forme de barre représentant le nombre d'unités sous forme d'un pourcentage de la largeur totale de la fenêtre du navigateur (100% pour 10 unités) :

```
<HR ALIGN=LEFT SIZE=10 WIDTH="&{total_unites * 10};%">
</BODY></HTML>
```

2.5 - Les URL JavaScript

Les URL JavaScript fonctionne comme tout autre type d'URL, de ce fait, on précise le type de lien : javascript:script .En voici un exemple :

```
<HTML><BODY>
<SCRIPT>
var resultat_test = "Le résultat de votre test est : " + 23 + "%";

function ameliore_resultat()
{
    resultat_test = "Résultat parfait !";
}

var petit = "#define x_width 1\n#define x_height 1\nstatic char
x_bits[] = {0x00};";

</SCRIPT>

<A HREF="javascript:resultat_test">Cliquez ici pour voir vos
résultats</A><BR>
<A HREF="javascript:ameliore_resultat()">Cliquez ici pour améliorer
vos résultats</A>
<IMG SRC="javascript:petit" NAME='Image bitmap simple'>

</BODY></HTML>
```

Ainsi

```
<A HREF="javascript:history.go(0)">cliquer pour rafraîchir</A>
placé dans le fichier « javascri4.htm », aura la même action que :
<A HREF="javascri4.htm">
LIENHYPERTEXTE "javascr4.htm"
cliquer pour rafraîchir
</A>
sans pour autant forcer à spécifier le nom de la page.
Ceci peut être étendu à une syntaxe du type :
<A HREF="javascript:history.go(-1)">aller à la page précédente</A>
```

2.6 - Les feuilles de styles JavaScript (JSSS)

Exemple :

```
<HTML><HEAD>
<STYLE TYPE="text/JavaScript">
  tags.P.borderWidth = 100;
  tags.P.borderStyle = 'none';// sinon on a une très large bordure
  tags.B.color = "red";
</STYLE>

<LINK REL=STYLESHEET TYPE="text/JavaScript" HREF="fancystyle"
TITLE="Fancy">
```

```
</HEAD><BODY>
<P> Paragraphe aéré ! </P>
<P><B>Paragraphe aéré agressif !</B></P>
</BODY></HTML>
```

L'utilisation des balises <STYLE> montre que la création de feuilles de styles dans JavaScript consiste à créer des propriétés correctement nommées et de leur affecter des valeurs réalistes. Les lettres en majuscules P et B correspondent respectivement au marqueurs <P> et <B>.
La balise <LINK> dans l'en-tête du document HTML a pour but d'identifier de manière générale le relation de dépendance de ce document avec d'autres documents. L'une de ces dépendances possibles est une feuille de style JavaScript, si bien que cette balise procure une alternative à l'incorporation d'une feuille de styles directement dans le document.
Il existe quelques fonctions JavaScript destinées spécifiquement aux JSSS :

```
<HTML><HEAD>
<STYLE TYPE="text/JavaScript">
  tags.H1.color = 'pink';
  // positionne les marges haute, droite, bas, gauche.
  tags.H1.margins(1,2,3,4);
  tags.H1.rgb(50,50,50);

  contextual(tags.H1, tags.B).fontStyle = "italic";

  function changer()
  {
    if ( color == 'pink' ) fontStyle = "medium";
  }

  tags.I.apply = changer();
</STYLE></HEAD></HTML>
```

La méthode margins() permet de définir les marges d'un seul coup.
La fonction contextual(), permet à un style de dépendre d'un autre. Ici, si un élément <B> est imbriqué dans un élément <H1>, alors il doit être affiché en italique.
Enfin, la propriété apply de la balise <I> est affecté sur le fonction changer().

3 - Fenêtres et Frames

Les fenêtres et frames du navigateurs sont structurés autour de l'objet navigateur et de l'objet document.

3.1 - Les fenêtres Navigateur

Les fenêtres principales du navigateur sont représentées par un objet *window*. La méthode *open()* de l'objet *window* permet d'ouvrir une autre fenêtre de navigateur.avec la syntaxe suivante :

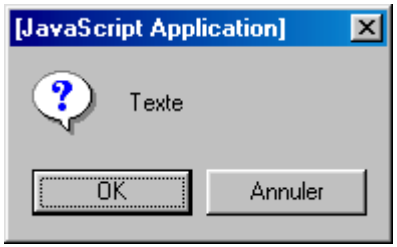
```
Nom_Variable = open(« URL », « Nom fenetre », « option=valeur,
option=valeur… ») ;
```

3.2 - Les fenêtres popup

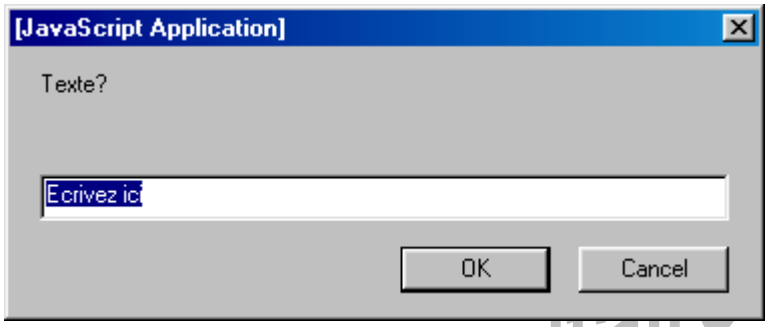
Trois types de fenêtres popup sont utilisables par JavaScript. Elles peuvent êtres instanciées par les méthodes de l'objet *window* et ont pour nom : *alert*, *confirm* et *prompt*.

3.2.1 - Une boite d'alerte :

3.2.2 - Une boite de confirmation :



3.2.3 - Une boite de saisie :



Un exemple :

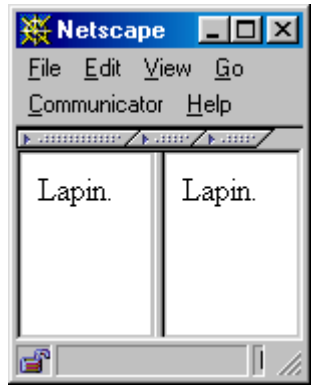
```
<HTML><BODY><SCRIPT>
var accord = false, notes = '';
accord = confirm('Vous aimez le chocolat?');
if ( accord )
    notes = prompt("est-ce bon pour la santé ?",'Ecrivez ici');
else
    notes = prompt("Mauvais ?",'Ecrivez ici');
alert(notes + '\nVous pouvez etre fier de vous.');
</SCRIPT></BODY></HTML>
```

La fonction *confirm()* renvoie un résultat booléen et la fonction *prompt()* une chaîne de caractères entrée par l'utilisateur ; la fonction *alert()* renvoie *undefined*. Les fenêtres ont toujours pour titre [Javascript Application].

3.3 - Les frames

Chaque fenêtre possède un objet *window* et chaque objet *window* possède deux propriétés importantes : *document* et *frames*. La propriété *document* référence l'objet document correspondant à l'URL de la fenêtre. La propriété *frames* est un tableau (éventuellement vide) d'objets *window*, représentant les frames. Le tableau de frames a une propriété *length*.
Examinez les quatre documents HTML suivants :

Source HTML	Affichage	Propriété *length*
`<!-- lapin.htm : trois lapins -->` `<HTML><BODY>Lapin.</BODY></HTML>`		0

`<!-- vsplit.htm : deux lapins -->`
`<HTML><FRAMESET COLS="*,*">`
`<FRAME SRC="lapin.htm" NAME="lapin1">`
`<FRAME SRC="lapin.htm" NAME="lapin2">`
`</FRAMESET></HTML>`
		2

Source HTML	Affichage	Propriété *length*
`<!-- trois.htm : trois lapins --> <HTML><FRAMESET ROWS="*,*"> <FRAME SRC="lapin.htm" NAME="lapin3"> <FRAMESET COLS="*,*"> <FRAME SRC="lapin.htm" NAME="lapin4"> <FRAME SRC="lapin.htm" NAME="lapin5"> </FRAMESET></FRAMESET></HTML>`		3
`<!-- complexe.htm : plusieurs lapins --> <HTML><FRAMESET ROWS="*,*"> <FRAME SRC="lapin.htm" NAME="lapin6"> <FRAME SRC="vsplit.htm" NAME="lapin7"> </FRAMESET></HTML>`		2

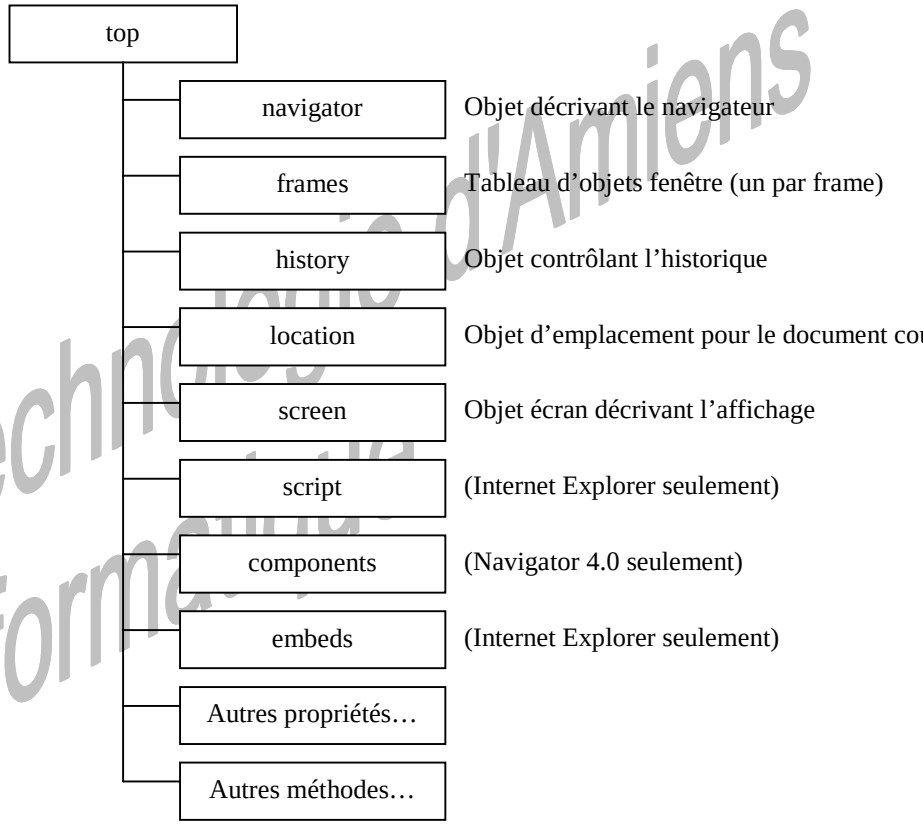
Dans le quatrième exemple, pour accéder au document (objet) en bas dans la partie gauche du frame initial, le jeu de frames parent (*top*) de JavaScript doit se repérer à travers la hiérarchie des objets de la façon suivante :

Top.frames[1].frames[0].document ;

Ou alors :

Top.lapin7.lapin1.document ;

La structure de ces quatre fenêtres peut être illustrée ainsi :



3.4 - L'Historique

Nous avons vu que les fenêtres et frames disposaient chacun d'un objet document pour leur URL. Ils possèdent également un objet historique, appelé *history*. Cet objet mémorise les URL de tous les documents chargés dans cette fenêtre ou ce frame, permettant de se référer à ces documents par la suite. L'utilisation la plus courante des historiques est illustrée par l'exemple ci-dessous :

```
<A HREF="#" ONCLICK="window.history.go(-1)">
  Revient un pas en arrière </A>
<A HREF="#" ONCLICK="window.location.reload()">
  Ne va nulle part </A>
<A HREF="#" ONCLICK="window.location.reload(true)">
  Tourne toujours en rond, mais avec plus de rigueur </A>
<A HREF="#" ONCLICK="window.location.replace(url)">
  Oublie le passe</A>
```

4 - Les Formulaires

4.1 - Description

JavaScript interagit avec les formulaires HTML de nombreuses manières. Il peut :

- Créer des balises pour formulaires contenant du JavaScript, de la même façon qu'il sait le faire pour d'autres éléments HTML,
- Agir comme un gestionnaire d'événements, par le biais d'attributs de balises ou de propriétés d'objets hôtes,
- Lire et modifier les valeurs des éléments du formulaire,
- Dans une certaine limite, construire lui-même des éléments de formulaires.

En allant au-delà de ces tâches simples, JavaScript apporte une réelle plus-value aux formulaires HTML. Voici certaines fonctions de plus haut niveau que JavaScript peut remplir :

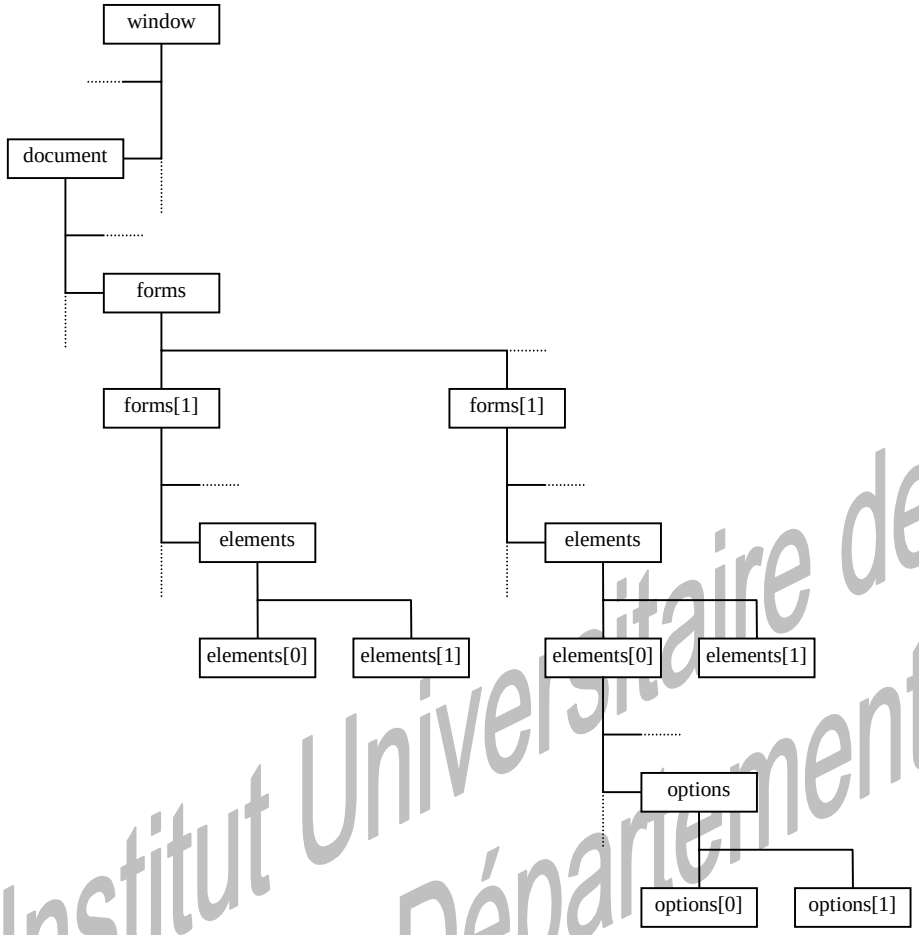
- Valider ou corriger les entrées utilisateur,
- Exécuter des calculs ou autres traitements sur les données entrées par l'utilisateur,
- Stocker et réacheminer les données entrées par l'utilisateur,
- Contrôler la navigation de l'utilisateur dans un formulaire d'après ses entrées.

Toutes ces possibilités ajoutent à la fluidité de traitement des formulaires, puisque ces tâches peuvent être effectuées au sein du navigateur, sans échanges fastidieux sur le réseau. Sans JavaScript, la moindre vérification nécessiterait une requête vers le serveur web, ce qui ralentirait considérablement les choses.

4.2 - Objets formulaires

Les formulaires HTML font partie des documents HTML. Il est donc naturel que les balises correspondantes soient reflétées par des propriétés de l'objet *document* dans JavaScript.

Le diagramme ci-après illustre l'organisation des objets relatifs aux formulaires par rapport à ceux du document.



4.2.1 - Présentation des objets formulaires

Un document HTML pouvant contenir plusieurs formulaires, il est naturel que la propriété *forms*, proche du sommet de la hiérarchie, soit un tableau et contienne une entrée pour chaque formulaire.
Chaque objet formulaire a une propriété *elements*, qui est un tableau. Chacun des éléments de ce tableau est l'un des contrôles du formulaire. Les contrôles du formulaires sont srockés dans le tableau en suivant l'ordre du document HTML.

Les balises `<INPUT>`, `<BUTTON>`, `<TEXTAREA>` et `<SELECT>` sont stockées comme éléments du tableau *forms*.

Ces objets ont tous en commun les propriétés *type*, *value*, *name* et *form* :

- La propriété *type* permet de reconnaître, depuis JavaScript, quelle balise a été utilisée pour créer l'élément ;
- La propriété *value* contient la valeur courante du contrôle ;
- La propriété *name* est une reprise de l'attribut `NAME` de la balise ;
- La propriété *form* fait référence au tableau *forms* associé au formulaire.

Voici un exemple de formulaire illustrant l'accès aux éléments de formulaires et à leurs propriétés :

```
<HTML><BODY><FORM NAME="patient">
Symptômes de folie. Entrez le numéro de la semaine: <INPUT
NAME="semaines" TYPE="text">
Entendez-vous des voix? <INPUT NAME="symptome" TYPE="checkbox">
Avez-vous des visions? <INPUT NAME="symptome" TYPE="checkbox">
Votre personnage préféré:
<SELECT NAME="regime">
<OPTION VALUE="napoleon"> Napoléon Bonaparte
<OPTION VALUE="folamour"> Dr. Folamour
</SELECT>
<INPUT TYPE="submit" value="Soumettre la requête">
</FORM></BODY></HTML>
```

Ces propiétés sont toutes valables pour le document :

```
with ( top.document )
{
  forms[0].name;                    // = 'patient'
  forms[0].method;                  // = 'GET' (méthode par défaut)
  forms[0].elements.length;   // = 5
  forms[0].elements[0].value; // = '', L'utilisateur peut le
modifier.
  forms[0].elements[0].onchange; // null, pas de gestionnaire.
  forms[0].elements[1].name;  // = 'symptome'
  forms[0].elements[2].checked;      // = 'false', l'utilisateur peut
le modifier.
  forms[0].elements[3].options.length; // = 2
  forms[0].elements[3].options[1].value; // 'dates'
}
```

Remarquez que la propriété *value* d'un éléments texte (*text*) ets toujours une chaîne de caractères, même si, comme dans l'exemple, on demande à l'utilisateur de taper un nombre.

4.2.2 - Nommer les éléments de formulaire

Si l'on dispose de plus d'un formulaire, ou si on utilise de nombreux éléments de formulaires, utiliser les tableaux à indices proposés par le langage peux s'avérer déroutant, comme le montrent les quelques lignes de l'exemple précédent. Fort heureusement, il est possible d'accéder aux formulaires et champs de formulaires en utilisant leurs noms. C'est possible car JavaScript perçoit objets et tableaux presque de la même façon. Voici donc un exemple illustrant ce principe :

```
with ( top.document )
{
patient.name;          // inutile
patient.method;
patient.elements.length;      // inutile si on connait tous les noms
patient.semaines.value;
patient.semaines.onchange;
patient.symptome[0].name;     // pour cette ligne et la suivante:
traitement spécial
patient.symptome[1].checked;
patient.regime.options.length;
patient.regime.options[1].value;
}
```

4.2.3 - Evénements associés aux formulaires

Le modèle d'événements interne à JavaScript offre le moyen le plus normal et habituel pour intercepter et traiter les saisies dans un formulaire. Si l'utilisateur désactive JavaScript dans son navigateur, tous les éléments d'un script suivront leur comportement par défaut. Voici un exemple qui illustre le gain en interactivité et en efficacité apporté par la gestion des événements :

```
<HTML><HEAD><SCRIPT>
function gagne()
{
  var reponse = "C'est la bonne reponse !\n" +
              "L'homme rampe quand il est jeune, marche quand il est
adulte et utilise une canne quand il vieillit\n" +
               "\nSi vous rencontriez un Sphinx egyptien aujourd'hui,
vous seriez parfaitement en securite !";
  alert(reponse);
}
</SCRIPT></HEAD>
<BODY><FORM>
<H1>Devinettes</H1>
Qu'est ce qui marche à quatre pattes le matin, à deux le midi et
trois le soir ?
<BR> <INPUT TYPE="radio" ONCLICK="alert('Vous semblez vous aussi en
pleine confusion')"> Une table en pleine confusion
<BR> <INPUT TYPE="radio" ONCLICK="alert('Vous avez pris un coup de
soleil, peut-etre ?')"> Le soleil
<BR> <INPUT TYPE="radio" ONCLICK="gagne()"> L'homme
</FORM></BODY></HTML>
```

Dans cet exemple simple, le gestionnaire d'événements donne à l'utilisateur des informations supplémentaires, en fonction de sa réponse. Ces informations ne pourraient être affichées par un document HTML seul.

Les événements *onReset* et *onSubmit* sont les seuls à s'appliquer au formulaire dans son ensemble. Ils correspondent aux actions par défaut des balises `<INPUT TYPE=« submit »>` et `<INPUT TYPE=« reset »>`.

L'événement *onClick* est en général le plus utilisé. On l'emploie pour détecter la sélection de boutons radio ou de cases à cocher, pour exécuter des actions et traitements comme ouvrir de nouvelles fenêtres, etc.

L'événement *onChange* est, lui, déclenché quand le pointeur de la souris de l'utilisateur quitte une zone qui vient de changer de valeur.

Les gestionnaires *onBlur* et *onFocus* sont activés quand l'utilisateur se déplace dans un formulaire d'un champ à l'autre (ou d'un frame à l'autre) dans une page. Leur principale utilisation est donc de contrôler la façon dont l'utilisateur navigue entre les champs.

En plus de ces événements, d'autres son disponibles à partir de la version 4.0 de HTML. Ces événements, tels que *onMouseMove* et *onKeyPress* sont encore peu utilisés dans les formulaires HTML.

4.3 - Les différents types de formulaires

Les éléments de formulaires peuvent être regroupé de différentes façons et pour différentes raisons. Il est utile de bien identifier le but général d'un formulaire, car cela permet de simplifier les techniques employées par la suite.

4.3.1 - Formulaires de contrôle et de calcul

Ce sont les types de formulaires les plus simples. Ils peuvent se résumer à un seul bouton intitulé « Fais ceci ». Une application classique consiste à fournir à l'utilisateur une barre de boutons, pour faciliter sa navigation entre les pages d'un

site. Cette utilisation des formulaires permet d'améliorer la présentation visuelle d'un document, qui sinon ne contiendrait que du texte et des liens classiques.

Une autre utilisation des formulaires consiste à utiliser l'interpréteur JavaScript pour exécuter des traitements mathématiques ou logiques, suivant les entrées de l'utilisateur comme, par exemple, un convertisseur d'unité.

4.4 - Envoyez vos données et oubliez les

Les formulaires les plus courant sont utilisés pour simplement récupérer des données relatives aux utilisateurs. Les données de ces formulaires sont envoyées vers un serveur et rien d'autre ne se passe : l'utilisateur n'aura sans doute jamais plus d'informations sur les données qu'il a saisies. De tels formulaires sont souvent des formulaires d'inscription ou de demande d'information. Ils correspondent à l'utilisation d'origine de la balise <FORM>, qui a été introduite avec le standard HTML 2.0.

4.5 - Le « panier à provisions »

Quand un formulaire sert à gérer plusieurs pages HTML, on peut alors utiliser la technique du « panier à provisions » pour surmonter certaines limitations de HTML.

Une utilisation classique du panier à provisions est de permettre à l'utilisateur de choisir et de commander des objets dont la description se trouve répartie sur plusieurs pages HTML : au fur et à mesure de la navigation sur le site, l'utilisateur remplit son panier comme il le ferait dans un supermarché, et le formulaire garde la mémoire des choix déjà effectués. Une fois sont choix fait, l'utilisateur peut d'un seul clic commander la liste complète des articles sélectionnés. Pour se rappeler cette liste, le formulaire a donc besoin d'un espace de stockage temporaire.

Les formulaires précédents, du type « envoyer et oublier » suffiraient si ne se posaient pas trois problèmes :

- La taille du document et sa présentation : la page ne serait pas lisible et serait longue à charger si la liste complète des articles était faite sur une seule page ;
- HTML ne propose aucun espace pour stocker temporairement des données : il faudrait donc que l'utilisateur commande article par article ;
- HTML ne permet pas de découper un formulaire unique sur plusieurs pages différentes.

Avec les navigateurs récents, JavaScript vient à la rescousse du programmeur. Les traitements nécessaires sont les suivants :

- Créer un espace de stockage provisoire pour le panier à provisions ;
- Créer des formulaires dans chacune des pages du catalogues ;
- Organiser ces formulaires pour que leurs données puissent être récupérées par le panier à provisions ;
- Envoyer les données du panier à provisions si le client décide de confirmer ses achats.

Le panier à provision sera donc un objet JavaScript disponible pour toutes les pages du catalogues (du site). Pour y parvenir, le meilleur moyen, voire l'unique, est d'utiliser les frames. Ainsi, le panier ayant été déclaré lors du chargement de la page contenant le frame de contrôle, le panier sera accessible à toutes les pages. De plus, l'utilisation des frames permettra d'avoir une table des matière, un menu qui rendra le parcours du client plus simple et direct.

4.5.1 - Un exemple

Tout d'abord, on crée le panier à provision. Ce panier sera stocké dans un document à base de frames, l'un des frames étant utilisé pour contrôler le menu tandis qu'un autre servira à explorer les pages du catalogue.

```
<!—catalogue.htm-->
<HTML>
<HEAD>
<TITLE> Catalogue </TITLE>
<SCRIPT>
var panier = new Object;
var length = 0;

function achats(nom, prix, quantite)
{
  if (quantite == 0)
  {
    for (choix in panier)
  {
    if (panier[choix].nom == nom)
    {
      delete panier[choix];
          length--;
    }
  }
  }
  else
  {
    this.nom = nom;
    this.prix = prix;
    this.quantite = quantite;
```

```
    length++;
  }
}

function eff_panier()
{
  length = 0;
  for (i in panier)
  {
    delete panier[i];
  }
}
</SCRIPT>
</HEAD>
<FRAMESET COLS="300,*">
  <FRAME NAME="menu" SRC="menu.htm">
  <FRAME NAME="page" SRC="bienvenue.htm">
</FRAMESET>>
</HTML>
```

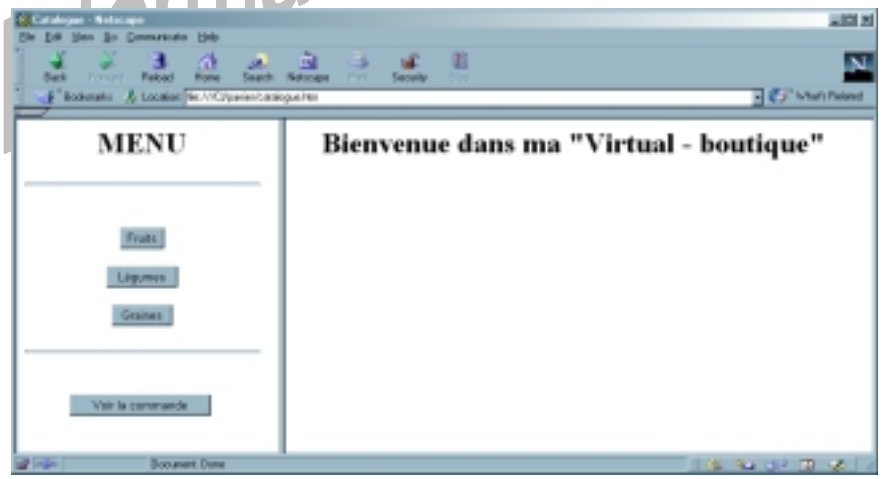
La frame de contrôle sert de menu :

```
<HTML>
<HEAD>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<SCRIPT>
  if (top == self)
  {
    self.location.href = "catalogue.htm";
  }
</SCRIPT>
<CENTER>
<H1>MENU</H1>
<HR><BR><BR>
<FORM>
<INPUT  TYPE="button"  ONCLICK='top.page.location="fruits.htm"'
VALUE="Fruits"><BR><BR>
<INPUT  TYPE="button"  ONCLICK='top.page.location="legumes.htm"'
VALUE="Légumes"><BR><BR>
<INPUT  TYPE="button"  ONCLICK='top.page.location="graines.htm"'
VALUE="Graines"><BR><BR>
<HR><BR><BR>
<INPUT  TYPE="button"  ONCLICK='top.page.location="commande.htm"'
VALUE="Voir la commande">
</FORM>
</CENTER>
</BODY>
</HTML>
```

Notez cependant que la page teste si elle est ouverte seule, auquel cas elle demande à charger la page contenant les frames du catalogues.
La page de garde sert de présentation pour l'utilisateur :

```
<HTML>
<HEAD>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<SCRIPT>
  if (top == self)
  {
    self.location.href = "catalogue.htm";
  }
</SCRIPT>
<CENTER>
<H1>Bienvenue dans ma "Virtual - boutique"</H1>
</CENTER>
</BODY>
</HTML>
```

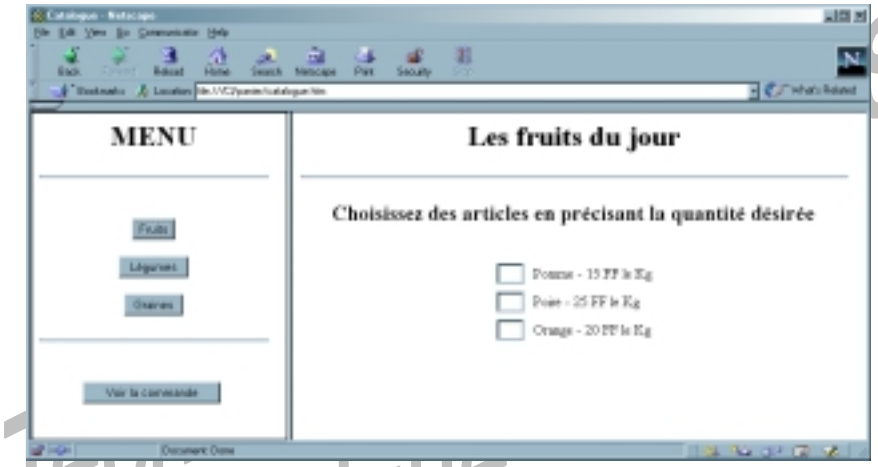
Voici l'image de la première page qui accueille l'utilisateur :



Toutes les pages du catalogue suivent le même schéma. Voici la première :

```
<HTML>
<HEAD>
<SCRIPT>
  function rappel_comm()
  {
    if (top.panier['PO'])
    {
document.form1.t1.value = top.panier['PO'].quantite;
    }
    if (top.panier['PR'])
    {
      document.form1.t2.value = top.panier['PR'].quantite;
    }
    if (top.panier['OR'])
    {
      document.form1.t3.value = top.panier['OR'].quantite;
    }
  }
</SCRIPT>
</HEAD>
<BODY BGCOLOR="#FFFFFF" ONLOAD="rappel_comm()">
<SCRIPT>
  if (top == self)
  {
    self.location.href = "catalogue.htm";
  }
</SCRIPT>
<CENTER>
<H1>Les fruits du jour</H1>
<HR>
<H2>Choisissez   des   articles   en   précisant   la   quantité
désirée</H2>
<BR>
<TABLE>
<FORM NAME="form1">
<TR><TD><INPUT   TYPE="text"   NAME="t1"   MAXLENGTH=2   SIZE=3
ONCHANGE="top.panier['PO']   =   new   top.achats('Pomme',15,
this.value)">
</TD><TD>Pomme - 15 FF le Kg<BR></TD></TR>
<TR><TD><INPUT   TYPE="text"   NAME="t2"   MAXLENGTH=2   SIZE=3
ONCHANGE="top.panier['PR']   =   new   top.achats('Poire',25,
this.value)">
</TD><TD>Poire - 25 FF le Kg<BR></TD></TR>
<TR><TD><INPUT   TYPE="text"   NAME="t3"   MAXLENGTH=2   SIZE=3
ONCHANGE="top.panier['OR']   =   new   top.achats('Orange',20,
this.value)">
</TD><TD>Orange - 20 FF le Kg<BR></TD></TR>
</FORM>
</TABLE>
</CENTER>
</BODY>
</HTML>
```

Si l'utilisateur charge cette page, elle apparaît comme suit :



Dans cet exemple, on identifie chacun des articles par deux lettres de code (PO, PR, OR), mais on pourrait utiliser n'importe quelle autre convention. La fonction rappel_comm() illustre comment on peut charger un nouveau frame tout en préservant les choix déjà effectués par l'utilisteur. Cela permet à la page d'être abandonnée et visitée à nouveau sans qu'aucune informations ne se perde.

La dernière page est celle qui permet de passer commande des objets du panier à provisions :

```
<HTML>
<HEAD>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<SCRIPT>
  var choix, total, totaligne;

  if (self == top)
  {
    self.location.href = "catalogue.htm";
  }
  else
  {
    if (top.length >0)
    {
    document.write('<H1>Vous avez choisi les articles suivants :
</H1><BR><BR><UL>');

    total = 0;
    for (choix in top.panier)
    {
```

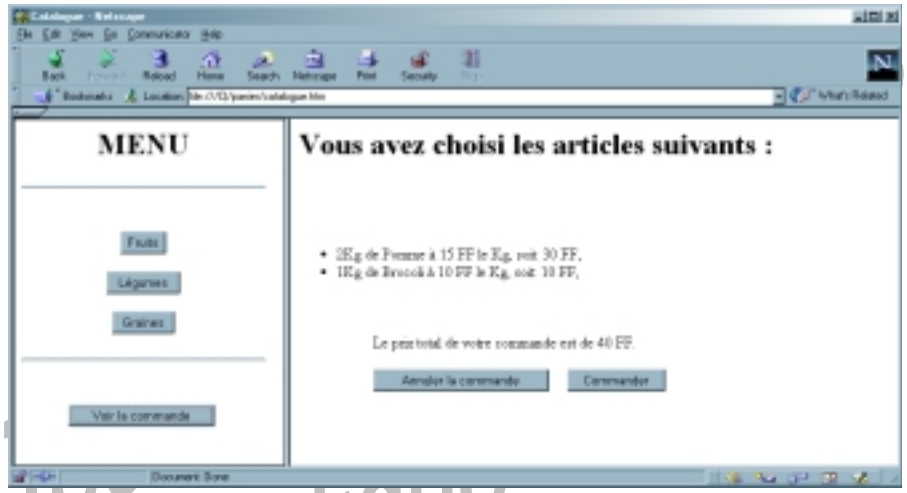
```
        with (top.panier[choix])
        {
          totaligne = quantite * prix;
          document.write('<LI>'+quantite);
          document.write('Kg de '+nom);
          document.write(' à '+prix);
            document.write('  FF  le  Kg,  soit:  '+totaligne+'
FF,');
          total += totaligne;
         }
   }
   document.write('<FORM METHOD="POST">');
    document.write('<UL><BR><BR>Le prix total de votre commande
est de '+total+' FF.<BR><BR>');
   for (choix in top.panier)
   {
        document.write('<INPUT  TYPE="hidden"  NAME="'+choix+'"
VALUE="'+top.panier[choix].quantite+'">');
        document.write('<INPUT TYPE="button" VALUE="Annuler la
commande"
ONCLICK=\'top.eff_panier();top.page.location.href="bienvenue.htm
"\'>');
        document.write('     ');
        document.write('<INPUT                    TYPE="submit"
VALUE="Commander">');
        document.write('</FORM>');
        document.write('</UL></UL>');
   }
  }
  else
  {
     document.write("<H2>Faite  d'abord  votre  choix  parmis  les
articles proposés.</H2>");
  }
  }
</SCRIPT>
</BODY>
</HTML>
```

Le contenu du panier à provisions est examiné deux fois :

- une fois pour que l'utilisateur puisse vérifier sa commande,
- une deuxième fois pour créer des champs cachés (qui pourraient ne pas l'être) dans le formulaire à envoyer.

Ce formulaire consiste en une liste de paires nom/prix, qui rappellent les codes des produits et les quantités demandées.

Cette page ressemblera par exemple à ceci :



5.1 - Les MAP et ISMAP en JavaScript

La balise MAP décrite au chapitre HTML 3.2 permet de déterminer les régions sensibles d'une image. Il est possible de positionner l'URL javascript dans cette balise comme le montre l'exemple suivant :

```
<MAP NAME="essai">
<AREA SHAPE="RECT" COORDS="0,0,100,100"
        HREF ="javascript:top.close(); window.location=page.html">
<AREA SHAPE="RECT" COORDS="101,101,200,200"
        HREF="essai.html" target="javascript:alert(`chargement') ;
      top.location=essai.html">
</MAP>
```

De même l'attribut ISMAP peut être accompagné d'un script avec appel de la méthode **search** :

```
<A HREF=javascript:history.go(0)><IMG SRC=france.gif BORDER=0
ISMAP></A>
<SCRIPT>
str = location.search
if (str == "")
        document.write("<P>Pas de coordonnées mais si vous cliquez
sur la carte ....")
else {
        positionseparateur = str.indexOf(",")
        document.write("<P> x est " + str.substring(1,
positionseparateur ))
        document.write("et y est " +
str.substring(positionseparateur +1, str.length))
}
</SCRIPT>
```

Pas de coordonnées mais si vous cliquez sur la carte

5.2 - Les cookies en JavaScript

Les cookies sont explicités dans le chapitre sur les serveurs HTTP, les notions doivent être assimilées pour comprendre la suite de ce chapitre.
JavaScript permet de positionner une variable cookie, avec une date d'expiration, ou d'obtenir la valeur de cette variable. Ainsi pour positionner une variable il suffit d'utiliser la fonction `setCookie` comme le montre l'exemple suivant :

```
 function setCookie(nom, valeur, expiration) {
        document.cookie = nom + "=" + escape(valeur)
        + ((expiration == null) ? "" : ("; expires=" +
expiration.toGMTString()))
}
```

de même la fonction suivante permet d'obtenir la variable en cookie :

```
function getCookie(nom) {
        var recherche = nom+ "="
        if (document.cookie.length > 0) {
                offset = document.cookie.indexOf(recherche)
                if (offset != -1) { // si le cookie existe
                        offset += recherche.length
                        // position du début
                        fin = document.cookie.indexOf(";",
offset)
                        // position de la fin
                        if (fin == -1)
                                fin=
document.cookie.length
                        return
unescape(document.cookie.substring(offset, fin))
                }
        }
}
```

Escape et **unescape** sont indispensable pour coder et décoder les valeurs.

5.3 - Teinte des propriétés

Les teintes de données, ou données teintées permettent d'assurer une fonction de sécurité importante en JavaScript, cette fonctionnalité n'étant valable qu'avec Netscape 3.0. Lorsque une données est dite teintée, un script JavScript ne peut pas l'envoyer sur une autre fenêtre, quel que soit d'ailleurs le serveur qui a envoyé la fenêtre en question. Si une variable est teintée le script ne peut pas envoyer d'informations à l'extérieur de la fenêtre active.
Pour permettre de teinter des informations, une variable d'environnement est appelée **NS_ENABLE_TAINT.** Cette variable est globale est doit être positionnée sur le client par une variable d'environnement (sur UNIX setenv ou set; sur Windows set dans l'autoexec.bat, sur NT variable système, sur Macintosh, ressource Envi). La variable doit être mise à 1 ou ne pas exister.
On le voit le mécanisme est basé sur un acte volontaire de la part de l'utilisateur du navigateur. Et si le navigateur ne permet pas la gestion des données teintées, une fenêtre viendra avertir l'utilisateur de l'impossibilité du traitement. Ainsi le code

suivant permet de tester l'état du navigateur :

```
if (navigator.taintEnabled()) {
   fonction1()
}
else fonction2()
```

Regardons les valeurs par défaut des différents objets manipulés par JavaScript.

Objet	Propriétés teintées
document	cookie, domain, forms, lastModified, links, referrer, title, URL
form	Action
éléments de formulaire	checked, defaultChecked, defaultValue, name, checked, defaultChecked, defaultValue, name, checked, defaultChecked, defaultValue, name,
history	current, next, previous, toString
Select	defaultSelected, selected, text, value
location et **Link**	hash, host, hostname, href, pathname, port, protocol, search, toString
window	defaultStatus, status

Il est ainsi possible d'interdire la communication des éléments teintés ou des éléments dérivés à travers le réseau. On appelle élément dérivés les retours de fonctions quand les paramètres des fonctions sont teintés, ou des sous-chaînes de chaîne de caractères teintées.
Pour teinter un élément dans un script il suffit d'utiliser la fonction **`taint`** et pour enlever la propriété d'utiliser **`untaint.`**

Exemple :

```
untaintedStatus=untaint(window.defaultStatus)
```

5.4 - Les types MIME

Un tableau contient tous les types MIME supportés par le client, aussi bien côté applications helpers que plug-ins. La syntaxe est la suivante :

```
navigator.mimeTypes[extension].type|description|suffixes|enabledPlugi
n
```

Le tableau suivant montre les cas possible pour le traitement de fichier GIF :

Expression	Valeur
navigator.mimeTypes["image/gif"].**type**	image/gif
navigator.mimeTypes["image/gif"].**description**	GIF Image
navigator.mimeTypes["image/gif"].**suffixes**	gif
navigator.mimeTypes["image/gif"].**enabledPlugin**	null

Ainsi pour savoir si le navigateur peut exécuter une séquence vidéo quicktime in effectura le test suivant :

```
var typemime = navigator.mimeTypes["video/quicktime"]
if (typemime)
   document.writeln("Cliquer <A HREF='fichier.qt'>ici</A> pour voir
")
else
   document.writeln("quicktime n'est pas supporté")
```

5.5 - Les Plug-Ins installés sur le navigateur

Comme nous le voyons dans le chapitre sur la technologie Netscape One, il est possible de contrôler les *plug-ins* installés sur le navigateur. Nous allons étudier ici comment tout cela peut être fait en JavaScript.
Le tableau *plugins* cointient tous les plug-ins installés sur le navigateur. La syntaxe est la suivante :

```
navigator.plugins['mom du plug-in'].name|description|filename|length
```

Ainsi pour Live Audio le tableau suivant montre les syntaxes possibles

Expression	Valeur
navigator.plugins['LiveAudio'].**name**	LiveAudio
navigator.plugins['LiveAudio'].**description**	LiveAudio - Netscape
navigator sound playing component	d:\nettools\netscape\nav30\
navigator.plugins['LiveAudio'].**filename**	Program\plugins\NPAUDIO.DLL
navigator.plugins['LiveAudio'].**length**	7

La longueur explicite que le plug-in live audio a un tableau mime contenant 7 éléments

Ainsi pour tester l'existance d'un plug-in Shockwave la programmation suivante

sera utilisée :

```
var Plugin = navigator.plugins["Shockwave"]
if (Plugin)
   document.writeln("<EMBED SRC='sequence.dir' HEIGHT=250
WIDTH=120>")
else
   document.writeln("Shockwave n'est pas installé!")
```

5.6 - Les préférences du navigateur

La méthode preference associée à l'objet navigator permet au développeur de connaître ou de positionner une préférence du navigateur. Cette méthode a été introduite avec la version 1.2 du langage.
Pour lire une préférence, la syntaxe suivante est utilisée :

```
navigator.preference(Nompreference),
```

la méthode retrourne alors la valeur de la préférence.
Pour positionner une préférence la syntaxe suivante est utilsiée :

```
navigator.preference(Nompreference,Valeurpreference)
```

Dans ce deuxième cas, la méthode retourne la valeur donnée à la préférence.

Dénomination	Syntaxe	Valeur	Description
Chargement des images	**general.always_load_images**	true ou false	permet de gérer le réglage du navigateur quant au chargement des images
Autorisation JAVA	**security.enable_java**	true ou false	permet de gérer le réglage du navigateur quant au chargement des applets JAVA
Gestion des feuilles de style	**browser.enable_style_sheets**	true ou false	permet de gérer le réglage du navigateur quant à ses capacités de lectures des feuilles de style
Gestion des installations automatiques	**autoupdate.enabled**	true ou false	permet de gérer le réglage du navigateur quant à sa capacité d'installer automatiquement des composants
Gestion des cookies	**network.cookie.cookieBehavior**	0 si le navigateur accepte tous les cookies 1 si le navigateur accepte uniquement les cookies qui proviennent du serveur d'origine 2 si le navigateur n'accepte pas la cookies	permet de gérer le réglage du navigateur quant à ses capacités de gestion des cookies
Gestion des cookies	**Network.cookie.warnAboutCookies**	true ou false	permet de gérer le réglage du navigateur quant aux messages d'alerte sur les Cookies

Exemple :

```
navigator.preference ("general.always_load_images", false);
```

6.1 - Les objets de base

Dans une page HTML, un certain nombre d'objets pré-définis sont accessibles sans que vous ayez à les créer; ce sont :

window : l'objet de plus haut niveau, c'est la fenêtre que vous voyez dans le navigateur.
child windows : ce sont les fenêtres filles que l'on retrouve dans chacune des frames. On verra leur appel dans les déclarations des frames
location : c'est l'URL courante et ses propriétés
history : ce sont les URL précédemment visitées.
document : ce sont les propriétés du document courant, comme le titre, les couleurs de fonds, les formulaires
navigator : ce sont le nom et la version du navigateur ainsi que les types MIME et les PLUG-INS supportés par le client.

Ceci représente un avantage extrêmement intéressant : vous allez pouvoir définir en JavaScript une nouvelle valeur pour le fond de votre document ou définir vos formulaires de façon dynamique.
Le schéma suivant, que l'on retrouve souvent dans la littérature, illustre la hiérarchie de toutes les classes présentes dans le Navigateur :

```
window
  |
  + --parent, frames, self, top
  |
  + --location
  |
  + --historique
  |
  + --document
       |
       + --formulaire (FORM)
       |    |
       |  éléments (text fields, textarea, checkbox, password
       |            radio, select, button, submit, reset)
       + --links
       |
       + -- URL
```

Prenons à titre d'exemple, une forme définie de la façon suivante :

```
<FORM NAME="nom">
<INPUT TYPE="text" NAME="essai" onChange="..."> ;
</FORM>
```

La valeur du texte entré est accessible par la valeur `document.nom.essai.value`
Le formulaire dans un document est stocké dans un tableau appelé `forms`, le premier formulaire est noté `forms[0]`, la deuxième est `forms[1]` etc..
Ainsi la valeur du texte entré qui était `document.nom.essai.value` peut être aussi appelé `document.form[0].essai.value`.
Ce mécanisme se répète pour tous les éléments comme les radios boutons, les champs texte etc .. L'objet window est un objet important dans le langage JavaScript, il permet l'ouverture et la fermeture de pages dans un nouveau navigateur, d'émettre des fenêtres Popup ou des fenêtres de confirmation.
A l'instar des formulaires rangés en tableau `form[]`, les frames sont rangées dans un `tableau frames[0]`, ... `frames[n]`. Ainsi la première frame d'une page composée est `window.frame[0]`.

6.2 - Les tableaux prédéfinis

Les principaux objets d'une page HTML sont définis par des tableaux que l'on retrouve dans la liste suivante :

Tableau	Description
anchors	comprend toutes les balises `<A>` qui contiennent un argument `NAME`
applets	comprend toutes les balises `<APPLET>` du document
arguments	comprend tous les arguments d'une fonction
elements	comprend toutes les balises `<FORM>` dans l'ordre de leur définition
embeds	comprend toutes les balises `<EMBED>` dans l'ordre de leur définition
forms	comprend toutes les balises `<FORM>` du document
frames	comprend toutes les balises `<FRAME>` contenant un FRAMESET dans l'ordre de leur définition
history	comprend tous l'historique du navigateur
images	comprend toutes les balises `<IMG>` dans l'ordre de leur définition
links	comprend tous les liens `<AREA HREF="...">`, `<A HREF="">` et les objets `Link` créés par la méthode linkwith the `link`
mimeTypes	Comprend tous les types MIME supportés par le navigateur (helpers ou plug-ins)
options	comprend tous les éléments `OPTION` d'une balise `SELECT`
plugins	comprend tous les plug-ins installés sur votre navigateur

6.3 - Les Objets

6.3.1 - Anchor

Nom	Définition	Voir aussi
Anchor	Les anchors sont les destinations (ou les signets) des liens hypertextes	Link
Syntaxe	`<A HREF="URL" NAME="Nom" TARGET="fenetre"> Texte </A>`	Définition
Tableau	document.anchors[0] .. document.anchors[document.anchors.length - 1]	-
Propriété	length	-
Méthode	eval – toString - valueOf	-
Se rapporte à	Document	-
Evénement	-	-

Exemple	`<HTML>`
	`<HEAD>`
	`<TITLE>UNGI &copy; Links et Anchors</TITLE>`
	`</HEAD>`
	
	`<BODY>`
	`<SCRIPT>`
	`window2=open("suscribe.htm","Abonnement","scrol lbars=yes,width=250, height=400")`
	
	`function AllerFenetre(num) {`
	`  if (window2.document.anchors.length > num)`
	`  window2.location.hash=num`
	`  else`
	`  alert("Erreur")`
	`}`
	`</SCRIPT>`
	
	`<B>Links et Anchors</B>`
	`<FORM>`
	`Cliquer sur le bouton pour aller vers l'URL<P>`
	`<INPUT TYPE="button" VALUE="Abonnement"`
	`NAME="bouton_abonnement"`
	`onClick="AllerFenetre(this.value)">`
	`</FORM>`
	`</BODY>`
	`</HTML>`	

Remarque	-	-

6.3.2 - Applet

Nom	Définition	Voir aussi
applet	permet d'accéder aux paramètres d'une applet L'applet doit être appelée avec un paramètre MAYSCRIPT	applet
Syntaxe	document.Applet[numero] `<APPLET CODE="monapplet.class" WIDTH50 HEIGHT=25 NAME="momApp" MAYSCRIPT>`	definition
Tableau	document.applets[]	-
Propriété	length (nombre d'applets dans le doc)	-
Méthode	eval – toString - valueOf	-
Se rapporte à	Document	-
Evénement	-	-

Exemple	`<SCRIPT>`
	`Document.write('Ce document contient ',document.applets.length,' applets')`
	`</SCRIPT>`	

Remarque	Version 1.1	-

6.3.3 - Area

Nom	Définition	Voir aussi
Area	permet la gestion des AREA des MAPS	Link

Syntaxe	`Voir exemple AREA`
	`<MAP NAME="nomImage">`
	`<AREA>`
	`</MAP>`
		MAP

Tableau	-	-
Propriété	hash - host – hostname – href – pathname - port - protocol - search - target	-
Méthode	-	-
Se rapporte à	document	-
Evénement	onMouseOut onMouseOver	-
Exemple	-	
Remarque	Version 1.1	-

6.3.4 - Array

Nom	Définition	Voir aussi
Array	permet la déclaration de tableaux	Link
Syntaxe	monTableau = new Array (*taille*) monTableau[0]= »Gilles »	MAP
Tableau	-	-
Propriété	length – prototype	-
Méthode	join – reverse – sort concat – pop – push – shift (version 1.2)	-
Se rapporte à		-
Evénement	-	-
Exemple	-	-
Remarque	Version 1.1	-

6.3.5 - Booléen

Nom	Définition	Voir
Boolean	permet la création /l'utilisation de types booléns	
Syntaxe	Déclaration : objetboolen = new Boolean(value) Utilisation : objetbooleen.propertyName	
Tableau	-	-
Propriété	prototype	-
Méthode	eval – toString – valueOf	-
Se rapporte à		-
Evénement	-	-
Exemple	bNoParam = new Boolean() bZero = new Boolean(0) bNull = new Boolean(null) bEmptyString = new Boolean("") bfalse = new Boolean(false)	-
Remarque	Version 1.1	-

6.3.6 - Boutons

Nom	Définition	Voir
Button	Bouton dans une page HTML	form – reset – submit
Syntaxe	`<INPUT TYPE= »button » NAME= »Nom »` `VALUE= »texte » onClick= »appelFonction »>`	definition INPUT
Tableau	-	-
Propriété	name – value – type	-
Méthode	click – blur – focus (blur et focus Netscape 3)	-
Se rapporte à	form	-
Evénement	onClick – onBlur – onFocus	-

Exemple	`<FORM><INPUT TYPE="button" VALUE="Abonnement"`
	`NAME="bouton_abonnement" onClick="window.open`
	`('suscribe.htm', 'Abonnement',`
	`'scrollbars=yes,status=yes,width=500,height=300`
	`')">`
	`</FORM>`	

Remarque	Button a été ajouté comme possibilité de la balise FORM avec le champ onClick, évitant le bouton *submit*	-

6.3.7 - Checkbox

Nom	Définition	Voir
checkbox	Les boîtes à cocher permettent la sélection d'un objet à 0 ou à 1	form – radio

Syntaxe	`<INPUT TYPE= »checkbox » NAME= »Nom »`
	`VALUE= »Valeur » CHECKED`
	`onClick= »appelFontion »>`
		definition INPUT

Tableau	-	-
Propriété	checked - defaultChecked - name - value - type	-
Méthode	click – blur – focus (blur et focus Netscape 3)	-
Se rapporte à	Form	-
Evénement	onClick – onBlur – onFocus	-

Exemple	`<FORM><INPUT TYPE="checkbox" VALUE="aide"`
	`NAME="aide"`
	`onClick="alert ('faire attention aux quotes`
	`dans l\'appel d\'alert') ">`
	`</FORM>`	

Remarque	La valeur onClick a été ajoutée au type *checkbox* existant	-

6.3.8 - Date

Nom	Définition	Voir
date	L'objet date permet d'utiliser la date et l'heure	-
Syntaxe	new Date() ,new Date("month day, year hours:minutes:seconds"), new Date(year, month, day), new Date(year, month, day, hours, minutes, seconds)	-
Tableau	-	-
Propriété	-	-
Méthode	getDate - getDay - getMinutes- getMonth - getSeconds - getTime - getTimeZoneoffset - getYear - parse - setDate - setHours - setMinutes - setMonth - setSeconds - setTime - setYear - toGMTString - toLocaleString - UTC – prototype	-
Se rapporte à	-	-
Evénement	-	-

Exemple	`<SCRIPT>`
	`<!-- a mettre dans l'entete`
	`function AfficheHeure () {`
	`Today = new Date ()`
	`document.write (Today.getHours() + "h" +`
	`Today.getMinutes () )`
	`}`
	`</SCRIPT>`
	
	`<SCRIPT>`
	`<! a mettre dans le corps`
	`AfficheHeure ()`
	`</SCRIPT>`	

Remarque	Il est normal que l'heure de s'affiche pas si vous imprimez le document, car le code généré par JavaScript ne s'imprime pas!	-

6.3.9 - document

Nom	Définition	Voir
document	L'objet document contient les propriétés énumérées ci-après	window – frame definition BODY

Syntaxe	`<BODY BACKGROUND= »backgroundImage »`
	`BGCOLOR= »backgroundColor »`
	`TEXT= »foregroundColor »`
	`LINK= »unfollowedLinkColor »`
	`ALINK= »activatedLinkColor »`
	`VLINK= »followedLinkColor »`
	`onLoad= »handlerText »`
	`onUnload= »handlerText »>`	

Tableau	-	-
Propriété	alinkColor - anchors - bgColor - cookie - fgColor - forms - lastModified - linkColor - links - location - referrer - title – vlink	-
Méthode	clear - close - open - write -writeln - getselection	-
Se rapporte à	window	-
Evénement	-	-
Exemple	`document.write ("La couleur de fond est ",` `document.fgColor,` `" La couleur du texte est` `",document.bgColor)` `</SCRIPT>`	
Remarque	-	-

6.3.10 - Element array

Nom	Définition	Voir aussi
element array	les tableaux *éléments* permettent d'indexer les éléments d'un formulaire par ordre d'apparition (check box, text area..)	Form
Syntaxe	formName.elements[index] – formName.elements.length	-
Tableau	document.anchors[0] .. document.anchors[document.anchors.length - 1]	-
Propriété	length	-
Méthode	-	-
Se rapporte à	form	-
Evénement	-	-
Exemple	-	
Remarque	-	-

6.3.11 - Event

Nom	Définition	Voir aussi
event	L'objet event contient le dernier événement, comme par exemple la dernière touche du clavier appuyée.	form
Syntaxe	event.	-
Tableau		-
Propriété	type - target - layerX - layerY - PageX - PageY - screenX - screenY - which - data - modifiers	-
Méthode	Abort - KeyDown - MouseUp - Blur - KeyPress - Move - Click - KeyUp - Reset – Change – Load - Resize – DblClick - MouseDown - Select – DragDrop – MouseMove - Submit - Error – MouseOut - Unload - Focus - MouseOver	-
Se rapporte à		-
Evénement	-	-
Exemple	`<SCRIPT LANGUAGE= »JavaScript1.2 »> function exemple(even) { alert (« Evenement : « + even.type) ; alert (« x position « + even.layerX) ; alert (« y position « + even.layerY) ; return true ; } document.onmousedown = exemple ; </SCRIPT>`	
Remarque	Version 1.2	-

6.3.12 - FileUpload

Nom	Définition	Voir aussi
FileUpload	l'attribut file Upload permet d'envoyer un fichier vers le serveur à partir du navigateur.	form
Syntaxe	<INPUT TYPE= »file » NAME= »fileUploadName » [onBlur= »texte »] fileUploadName.propertyName fileUploadName.methodName	-
Tableau	document.anchors[0] .. document.anchors[document.anchors.length – 1]	-
Propriété	name - type - value – formproperties	-
Méthode	blur - eval - focus - toString – valueOf	-
Se rapporte à	form	-
Evénement	onBlur – onChange - onFocus	-
Exemple	`<FORM NAME= »form1 »>` `Fichier à envoyer : <INPUT TYPE= »file »` `NAME= »myUploadObject »>` `<P>Get properties ` `<INPUT TYPE= »button » VALUE= »name »` `onClick= »alert('name : ' +` `document.form1.myUploadObject.name) »>` `<INPUT TYPE= »button » VALUE= »value »` `onClick= »alert('value : ' +` `document.form1.myUploadObject.value) »> ` `</FORM>`	
Remarque	FileUploadName est bien la valeur à renseigner pour accéder à l'objet FileUpload, ce n'est pas le nom du fichier à envoyer. Nescape 3.0	-

6.3.13 - Form

Nom	Définition	Voir aussi
form	les formulaires permettent de gérer les entrées claviers (zones textes, boutons radio etc.)	button - checkbox - hidden - password - radio - reset - select - submit - text – textarea
Syntaxe	`<FORM NAME="nom" TARGET="fenetre" ACTION="action" METHOD=GET | POST ENCTYPE="encodingType" onSubmit="appelFonction"> </FORM>`	definition FORM
Tableau	document.forms[index]- document.forms.length	-
Propriété	action - elements - encoding - length - method – target	-
Méthode	submit – reset (version 3)	-
Se rapporte à	document	-
Evénement	onSubmit – onReset	-

Exemple	`<SCRIPT>`
	`<!-- a mettre dans la partie entete>`
	`function testInput (){`
	`if (document.form1.quatrecar.value.length != 4)`
	`{`
	`alert("Entrez exactement 4 caractères " + document.form1.quatrecar.value + " n'est pas valide")`
	`}`
	`}`
	`</SCRIPT>`
	
	`<!-- dans le corps du texte>`
	
	`<FORM NAME=form1 onSubmit="testInput()" >`
	`<INPUT TYPE="text" VALUE="1" SIZE=4 NAME=quatrecar>`
	`</FORM>`	

Remarque	On le voit, l'événement onSubmit est un moyen efficace de tester les champs entrés dans un formulaire	-

6.3.14 - Frames

Nom	Définition	Voir aussi
frames	les frames sont les sous fenêtres indépendantes d'une page HTML	window – document
Syntaxe	`<FRAMESET ROWS= »hauteur » COLS= »largeur » onLoad= »appelfonction » onUnload= »appelfonction »> <FRAME SRC= »URL » NAME= »nom »> </FRAMESET>`	definition FRAME
Tableau	[frameReference.]frames[index] – [frameReference.]frames.length – [windowReference.]frames[index] – [windowReference.]frames.length	-
Propriété	length – parent – self – window	-
Méthode	clearTimeout – setTimeout – blur – focus	-
Se rapporte à	frame – window	-
Evénement	onBlur – onFocus	-
Exemple	`<SCRIPT>document.write ("Nb frames : " + window.frames.length)</SCRIPT>`	
Remarques	-	-

6.3.15 - Fonction

Nom	Définition	Voir aussi
Function	spécifie une séquence à compiler sous la forme d'une fonction	
Syntaxe	Nomfonction= new Function ([arg1, arg2, ... argn], corps de la fonction)>	
Tableau	-	-
Propriété	arguments - array - caller - prototype	-
Méthode	eval – toString - valueOf	-
Se rapporte à		-
Evénement		-
Exemple	var setBGColor = new Function("document.bgColor='antiquewhite'")	-
Remarques	Netscape 3.0 - Moins efficace qu'un appel de fonction non compilé.	-

6.3.16 - Hidden

Nom	Définition	Voir aussi
hidden	l'argument hidden permet de soumettre une requête de formulaire au serveur sans afficher la commande ainsi masquée	cookie
Syntaxe	`<INPUT TYPE= »hidden » NAME= »nom » VALUE= »texte »>`	definition INPUT
Tableau	-	-
Propriété	name – value – type	-
Méthode	eval – toString – valueOf	-
Se rapporte à	form	-
Evénement	-	-

Exemple	`<FORM NAME=form3>`
	`<INPUT TYPE="hidden" NAME="quelconque" VALUE="Caché">`
	`<INPUT TYPE="texte" NAME="surprise">`
	`<INPUT TYPE="button" Value="voir champ caché" NAME="bouton"`
	
	`onClick="document.form3.surprise.value=document.form3.quelconque.value">`
	`</FORM>`	

Remarque	-	-

6.3.17 - History

Nom	Définition	Voir aussi
history	Permet de récupérer la liste des précédentes pages visitées par le navigateur	-
Syntaxe	history.propertyName - history.methodName(parameters)	-
Tableau	history (3)	-
Propriété	length	-
Méthode	back – forward – go current – next – previous (Version 3)	-
Se rapporte à	document	-
Evénement	-	-

Exemple	`<SCRIPT>`
	`<--! partie a insérer dans l'en-tête`
	`function Reload () {`
	`history.go(0)`
	`}`
	`</SCRIPT>`
	
	`<INPUT TYPE="image" BORDER=0 SRC="gif/reload.gif"  onClick="Reload()"`	

Remarque	History permet de simuler également les autres touches de navigation de votre logiciel.	-

6.3.18 - Image

Nom	Définition	Voir aussi
Image	L'objet image peut être manipulé en JavaScript comme avec la balise	images
Syntaxe	imageName = new Image([width, height]) imageName.propertyName	-
Tableau	document.image[numero|length]	-
Propriété	border - complete - height - hspace - lowsrc - name - prototype - src - vspace - width	-
Méthode	eval – toString – valueOf	-
Se rapporte à	document	-
Evénement	onAbort onError onLoad	-

Exemple	`<SCRIPT>`
	`delai = 100`
	`Numero = 0`
	`MonImages = new Array()`
	`for(i = 0; i < 10; i++) {`
	`  MonImages[i] = new Image()`
	`  MonImages[i].src = "gif/" + i + ".gif"`
	`}`
	`function animate() {`
	`  document.animation.src = MonImages[Numero].src`
	`  Numero++`
	`  if(Numero>= 10) {`
	`  Numero = 0`
	`  }`
	`}`
	`</SCRIPT>`
	`<BODY BGCOLOR="white">`
	`<IMG NAME="animation" SRC="gif/0.gif""`
	`  onLoad="setTimeout('animate()', delai)">`	

Remarque	Version 1.1	-

6.3.19 - Link

Nom	Définition	Voir aussi
link	les liens hypertextes	anchor
Syntaxe	`<A HREF=URL NAME= »nom du signet » TARGET= »fenetre » onClick= »appelfonction » onMouseOver= »appelFonction »> texte</A>`	definition URL
Tableau	document.anchors[index] –document.anchors.length	-
Propriété	hash - host - hostname - href - pathname - port - protocol - search - target – length	-
Méthode	-	-
Se rapporte à	document	-
Evénement	onClick – onMouseOver – onMouseOut	-
Exemple	`<!-- dans le corps du texte> <A HREF=annonce.htm onMouseOver ="alert('N\'oubliez pas de regarder les annonces Ungi')">annonce.htrm</A>`	
Remarque	Bien entendu, l'événement *onMouseOver* masque l'événement *onClick*	-

6.3.20 - Location

Nom	Définition	Voir aussi
location	Permet d'extraire toutes les informations relatives à une URL	-
Syntaxe	[windowReference.]location.*nom*	history – location
Tableau	-	-
Propriété	hash - host - hostname - href - pathname - port - protocol - search – target	-
Méthode	reload – replace	-
Se rapporte à	document	-
Evénement	-	-
Exemple	`<!-- dans le corps du texte> <SCRIPT> document.write("<FONT COLOR=Green>Le protocole est :</FONT> " + window.location.protocol) </SCRIPT>`	
Remarque	hash désigne le nom du signet ou de l'ancre	-

6.3.21 - Math

Nom	Définition	Voir aussi
Math	ensemble de fonctions et valeurs mathématiques	-
Syntaxe	Math.propertyName – Math.methodName(parameters)	-
Tableau	-	-
Propriété	E - LN2 - LN10 - LOG2E - LOG10E - PI - SQRT1_2 - SQRT2	-
Méthode	abs - acos - asin - atan - ceil - cos - exp - floor - log - max - min - pow - random - round - sin - sqrt – tan	-
Se rapporte à	-	-
Evénement	-	-
Exemple	`<FORM> <INPUT TYPE="button" VALUE="racine" onClick="form.result.value = Math.sqrt(form.nombre.value)"> <INPUT TYPE="text" NAME="nombre" SIZE=6><INPUT TYPE="text" NAME="result"> </FORM>`	
Remarque	-	-

6.3.22 - Navigator

Nom	Définition	Voir aussi
navigator	information sur la version de navigateur utilisée	-
Syntaxe	navigator.propertyName	-
Tableau	-	-
Propriété	appCodeName - appName - appVersion - userAgent + mimeTypes plugins (V3)	-
Méthode	javaEnabled (V1.1) preference (V1.2)	-
Se rapporte à	-	-
Evénement	-	-
Exemple	`<SCRIPT> document.write ( "<FONT COLOR=green>Votre navigateur est</FONT> " + navigator.appName + navigator.appVersion ) </SCRIPT>`	
Remarque	-	-

6.3.23 - Number

Nom	Définition	Voir aussi
Number	Objet permettant de manipuler des valeurs numériques	-
Syntaxe	numberObjectName = new Number() numberObjectName.propertyName	-
Tableau	-	-
Propriété	MAX_VALUE - MIN_VALUE – NaN - NEGATIVE_INFINITY - POSITIVE_INFINITY prototype	-
Méthode	reval – toString – valueOf	-
Se rapporte à	document	-
Evénement	-	-
Exemple	biggestNum = Number.MAX_VALUE	-
Remarque	Version 1.1	-

6.3.24 - Option

Nom	Définition	Voir aussi
option	Objet d'un champs SELECT	-
Syntaxe	selectName.options selectName.options[index] selectName.options.length	-
Tableau	-	-
Propriété	-	-
Méthode	-	-
Se rapporte à	-	-
Evénement	-	-

Exemple	`<FORM Name=form4>`
	`<SELECT NAME="choix"`
	`onChange="form4.resultat.value+=form4.choix.opt`
	`ions[form4.choix.selectedIndex].value">`
	`<OPTION VALUE="Aller "> Aller`
	`<OPTION VALUE="Retour "> Retour`
	`<OPTION VALUE="Séjour ">Séjour`
	`</SELECT>`
	`<INPUT TYPE=TEXTE SIZE=15 NAME="resultat">`
	`</FORM>`	

Remarque	-	-

6.3.25 - Password

Nom	Définition	Voir aussi
password	champs d'un formulaire remplis avec des caractères *	form – text
Syntaxe	`<INPUT TYPE= »password » NAME= »nom »` `VALUE= »texte » SIZE=nombre>`	-
Tableau	-	-
Propriété	defaultValue – name – value – type	-
Méthode	focus – blur – select	-
Se rapporte à	form	-
Evénement	onBlur onFocus	-

Exemple	`<FORM>`
	`<INPUT TYPE="password" NAME="passwd"`
	`onBlur="alert('Votre mot de passe était ' +`
	`this.form.passwd.value)">`
	`</FORM>`	

Remarque	-	-

6.3.26 - Plugin

Nom	Définition	Voir aussi
Plugin	donne la liste des plugins installé sur le navigateur	-
Syntaxe	navigator.plugins[index].propertyName navigator.plugins[pluginIndex][mimeTypeIndex].mimeTypePropertyName	-
Tableau	navigator.plugins[index]	-
Propriété	description – filename – length - name	-
Méthode	eval – toString – valueOf	-
Se rapporte à	navigator	-
Evénement	-	-

Exemple	`<SCRIPT>`
	`for (i=0; i < navigator.plugins.length; i++) {`
	
	`document.writeln(navigator.plugins[i].name,`
	`navigator.plugins[i].description,`
	`navigator.plugins[i].length)`
	`}`
	`</SCRIPT>`	

Remarque	Version 1.1	-

6.3.27 - Radio

Nom	Définition	Voir aussi
Radio	bouton radio dans un formulaire (choix exclusifs)	form – checkbox – select
Syntaxe	`<INPUT TYPE= « radio » NAME= "nom » VALUE= « valeur » [CHECKED] [onClick= « handlerText »] >`	-
Tableau	-	-
Propriété	checked - defaultChecked - length - name - value + type (v3)	-
Méthode	click – blur – focus	-
Se rapporte à	form	-
Evénement	onClick	-

Exemple	`<FORM NAME=form5>`
	`<INPUT TYPE="text" Name="ch1" SIZE=4>`
	`<INPUT TYPE="radio" NAME="ch2" value=un onClick="this.form.ch1.value=this.value"><INPUT TYPE="radio" NAME="ch2" value=deux onClick="this.form.ch1.value=this.value">`
	`</FORM>`	

Remarque	-	-

6.3.28 - Reset

Nom	Définition	Voir aussi
reset	bouton de remise à zéro dans les formulaires	form – onSubmit
Syntaxe	`<INPUT TYPE="reset" NAME="nom" VALUE="texte" [onClick="handlerText"] >`	-
Tableau	-	-
Propriété	name – value - type	-
Méthode	click – blur – focus	-
Se rapporte à	form	-
Evénement	onClick – onBlur – onFocus	-

Exemple	`<FORM>`
	`<INPUT TYPE="Reset" onClick="alert('pas question')">`
	`</FORM>`	

Remarque	-	-

6.3.29 - Select

Nom	Définition	Voir aussi
select	menu à choix multiple dans un formulaire	form – checkbox – radio

Syntaxe	`<SELECT NAME= »nom » [SIZE= »integer »] [MULTIPLE] [onBlur= »appelFonction »] [onChange= »appelFonction »] [onFocus= »appelFonction »]>`
	`<OPTION VALUE= »valeur » [SELECTED]> texte`
	`</SELECT>`
		-

Tableau	-	-
Propriété	length - options - selectedIndex - defaultSelected - index - selected - text - value + type (V3)	-
Méthode	blur – focus	-
Se rapporte à	form – select	-
Evénement	onBlur – onChange – onFocus	-

Exemple	`<FORM>`
	`<SELECT NAME="liste" SIZE=1>`
	`<OPTION SELECTED VALUE="">Choisissez`
	`<OPTION VALUE="index.htm">Index`
	`<OPTION VALUE="suscribe.htm">Inscription`
	`<OPTION VALUE="aparaitr.htm">A paraître`
	`</SELECT>`
	`<INPUT TYPE="button" VALUE="Allez" onClick="if (form.liste.selectedIndex != 0)`
	
	`window.open(form.liste.options[form.liste.selectedIndex].value, 'Abonnement', 'scrollbars=yes,status=yes,width=500,height=300');`
	`else alert('Veuillez faire un choix')">`
	`</FORM>`	

Remarque	-	-

6.3.30 - String

Nom	Définition	Voir aussi
string	chaîne de caractères	text – textarea
Syntaxe	*chaine.propriété* - *chaine.méthode*(*paramètres*)	
Tableau	-	-
Propriété	length	-
Méthode	anchor - big - blink - bold - charAt - fixed - fontcolor - fontsize - indexOf - italics - lastIndexOf - link - small - strike - sub - substring - sup - toLowerCase – toUpperCase	-
Se rapporte à	-	-
Evénement	-	-

Exemple	`<FORM>`
	`<INPUT TYPE="button" VALUE="majuscule"`
	`onClick="form.result.value =`
	`form.chaine.value.toUpperCase() ">`
	`<INPUT TYPE="text" NAME="chaine" VALUE=gilles`
	`SIZE=6>`
	`<INPUT TYPE="text" NAME="result" >`
	`</FORM>`	

Remarque	-	-

6.3.31 - Submit

Nom	Définition	Voir aussi
submit	permet l'envoi au serveur http de la requête de formulaire	
Syntaxe	<INPUT TYPE= »submit » NAME= »Nom » VALUE= »Texte » onClick= »handlerText »>	form – reset
Tableau	-	-
Propriété	name – value + type (V3)	-
Méthode	click + blur – focus (V3)	-
Se rapporte à	form	-
Evénement	onClick	-

Exemple	`<SCRIPT>`
	`function Calcul (form)`
	`{`
	`i=eval(form.op1.value) + eval( form.op2.value)`
	`i="Résultat : " + i`
	`alert ( i)`
	`}`
	`</SCRIPT>`
	`<FORM>`
	`<INPUT NAME=op1 VALUE=1>`
	`<B>+</B>`
	`<INPUT NAME=op2 VALUE=2>`
	`<INPUT NAME=result TYPE=SUBMIT VALUE="="`
	`onClick="Calcul(this.form)">`
	
	`</FORM>`	

Remarque	-	-

6.3.32 - Text

Nom	Définition	Voir aussi
text	texte entré dans un formulaire	-
Syntaxe	<INPUT TYPE="text" NAME="Nom" VALUE="Valeur" SIZE=integer onBlur="handlerText" onChange="handlerText" onFocus="handlerText" onSelect="handlerText">	password - form - string – textarea
Tableau	-	-
Propriété	defaultValue - name - value + type (V3)	-
Méthode	focus – blur – select	-
Se rapporte à	form	-
Evénement	onBlur – onChange – onFocus – onSelect	-
Exemple	<FORM NAME="farce"> <INPUT TYPE="text" NAME="champ" SIZE="15" onFocus="this.value='que je tape vite!'"> </FORM>	
Remarque	-	-

6.3.33 - Textarea

Nom	Définition	Voir aussi
textarea	Zone de saisie de texte sur plusieurs lignes dans un formulaire	-
Syntaxe	`<TEXTAREA NAME= »Nom » ROWS= »integer » COLS= »integer » textToDisplay onBlur= »handlerText » onChange= »handlerText » onFocus= »handlerText » onSelect= »handlerText »>texte</TEXTAREA>,`	password – form – string – text
Tableau	-	-
Propriété	-	-
Méthode	focus - blur - select+ type (V3)	-
Se rapporte à	form	-
Evénement	onBlur – onChange – onFocus – onSelect	-

Exemple	`<FORM>`
	`<TEXTAREA NAME="texte" COLS=27 ROWS=4 onFocus="this.value='Cette aide est vraiment\ntrès bien faite,\ndire que je viens d\'acheter\nun bouquin complètement nul'">`
	
	`</TEXTAREA>`
	`</FORM>`	

Remarque	-	-

6.3.34 - Window

Nom	Définition	Voir aussi
window	fenêtre à l'intérieur du navigateur ou fenêtre indépendante	-
Syntaxe	windowVar = window.open("URL", "Nom" [,"options"])	document - frame
Tableau	-	-
Propriété	defaultStatus - frames - length - name - parent - self - status - top – window	-
Méthode	alert - close - confirm - open - prompt - setTimeout – clearTimeout	-
Se rapporte à	-	-
Evénement	onUnload	-
Exemple	`<FORM><INPUT TYPE="button" VALUE="Abonnement" NAME="bouton_abonnement" onClick="window.open ('suscribe.htm', 'Abonnement', 'scrollbars=yes,status=yes,width=500,height=300')">` `</FORM>`	
Remarque	-	-

6.4 - Leurs Propriétés

Nom	Propriété	Syntaxe	Description
action	form	nomdeforme.*action*	détermine l'URL appelée après appui sur le bouton Submit
alinkColor	document	document.alinkColor	détermine la couleur des liens visités
appCodeName	navigator	navigator.appCodeName	donne le nom de code du navigateur (par exemple Mozilla pour Netscape)
appName	navigator	navigator.appName	donne le nom du navigateur (par exemple Netscape)
appVersion	navigator	navigator.appVersion	donne une chaîne de caractères contenant la version, la plate-forme et la langue du navigateur
bgColor	document	document.bgColor	donne la couleur de fond du document
checked	checkbox, radio	nomcheckbox.checked nomradio[index].checked	renvoit la valeur de l'objet True/False
cookie	document	document.cookie	permet d'accéder à des variables internes (dates d'expiration du cache etc)
defaultChecked	checkbox, radio	nomcheckbox.defaultChecked nomradio[index].defaultChecked	indique l'état par défaut de l'objet concerné
defaultSelected	option	nom.options[index].defaultSelected	retoune une valeur indiquant l'état par défaut d'un objet sélectionable
defaultStatus	window	windowReference.defaultStatus	est le message par défaut affiché dans la région de statut de votre navigateur
defaultValue	hidden,password, text, textarea	nomdechamp.defaultValue	chaîne de caractères indiquant la valeur par défaut d'une zone texte, mot de passe ou zone texte
E	Math	Math.E	Constante e (environ 2.718)
encoding	form	nom.encoding	chaîne spécifiant l'encodage MIME dans une forme c'est à dire la valeur ENCTYPE
fgColor	document	document.fgColor	chaîne spécifiant la couleur du texte d'un document
getselection	document	document.getselection	Renvoie le texte sélectionné dans le navigateur
hash	link, location	location.hash	chaîne commençant par le caractère # qui spécifie une ancre dans une URL
host	link, location	location.host	chaîne de caractères déterminant la partie host d'une URL

Nom	Propriété	Syntaxe	Description
Hostname	link , location	location.hostname	chaîne de caractères déterminant la partie hostname d'une URL
href	link , location	Location.href	chaîne de caractères déterminant l'URL
index	option	nom.options[indexValue].index	entier représentant l'index d'une option de l'objet sélectionné
lastModified	document	document.lastModified	la date de dernière modification d'un document
length	frame, history, radio, select, string, window, anchors, elements, forms, frames, links, options	objet.length	longueur de l'objet ou nombre d'objet
linkColor	document	document.linkColor	Spécifie la couleur des liens Hypertextes
LN2	Math	Math.LN2	Logarithme de 2 (0,693)
LN10	Math	Math.LN10	Logarithme de 10 (2,302)
location	document	document.location	l'URL complet du document
LOG2E	Math	Math.LOG2E	Logarithme base 2 de e (1.442)
LOG10E	Math	Math.LOG10E	Logarithme base de 10 de e (0,434)
method	form	nom_de_form.method	spécifie la méthode (POST,GET) utilisée pour envoyer un formulaire sur un serveur

Nom	Propriété	Syntaxe	Description
name	objet	objet.name	window.name est le nom de la fenêtre
parent	window frame	Parent.methodName	c'est la fenêtre contenant les frames
pathname	location link	location.pathname	C'est la partie répertoire d'un URL
preference	navigator	navigator.preference	permet de connaître ou de positionner les préférences du navigateur. Voir éléments avancés de Javascript
PI	Math	Math.PI	le nombre PI (3.14159)
port	location	location.port	le port de l'URL
protocol	location	location.protocol	protocole de l'URL (http, ftp etc..)
referrer	document	document.referrer	l'URL du document appelant le lien cliqué
search	location	location.search	chaîne commençant par un point d'interrogation et correspondant aux informations d'une requête
selected	options	objet.options[index].selected	booléen indiquant si un objet est sélectionné ou non
selectedIndex	selectoptions	nom.selectedIndex, objet.options.selectedIndex	donne le rang de l'objet sélectionné dans une liste
self	window, frame	self.méthodew	désigne la frame ou la fenêtre courante
SQRT1_2	Math	Math.SQRT1_2	racine carrée de 1/2 (0,707)
SQRT2	Math	Math.SQRT2	racine carrée de 2 (1,414)

Nom	Propriété	Syntaxe	Description
Status	window	fenêtre.status	c'est la fenêtre status en bas à droite de Netscape
target	form, link ,location	form.target, lien.target	c'est le nom de la fenêtre qui va afficher le résultat d'un clic sur un URL ou un bouton de soumission
text	options	nom.options[index].text	texte suivant une balise <OPTION> dans une sélection
title	document	document.title	c'est le titre d'un document
top	window	top.propriété	c'est la fenêtre de plus haut niveau, c'est à dire celle initiale
userAgent	navigator	navigator.userAgent	nom d'agent du logiciel de navigation
value	button, checkbox, hidden, password, radio, reset, submit, text, textarea, options	objet.value	valeur de l'objet
vlinkColor	document	docuement.vlinkColor	couleur des liens visité
window	window	window.propriété	c'est la fenêtre courante

6.5 - Leurs événements

- **onBlur** : se produit quand un textarea, un text ou un select perd la main sur les entrées clavier
- **onChange** : se produit quand un textarea, un text ou un select est modifié par l'utilisateur
- **onClick** : se produit quand un button, checkbox, radio, link, reset ou un submit reçoit un click de la souris
- **onFocus** : se produit quand un textarea, un text ou un select prend la main sur les entrées clavier
- **onLoad** : se produit quand le navigateur a fini de charger une fenêtre ou toutes les frames d'un FRAMESET. L'événement
- **onLoad** se positionne dans la balise BODY ou dans la balise FRAMESET
- **onMouseOver** : se produit quand la souris passe sur un link ou un layer (version 1.2)
- **onSelect** textarea, un text : se produit quand un textarea, un text ou un text est sélectionné.
- **onSubmit** : se produit quand une form est soumise au serveur par l'appui du bouton Submit.
- **onUnload** :se produit quand un document est quitté. L'événement onUnload se positionne dans la balise BODY ou dans la balise FRAMESET
- **onAbort** : (version 1.1) se produit quand l'utilisateur avorte le chargment d'une image
- **onError** : (version 1.1) se produit quand le chargement d'une page ou d'une image produit une erreur.
- **onMouseout** : (version 1.1) se produit quand la souris quitte une zone area , un link ou un layer (version 1.2)
- **onReset** : (version 1.1) se produit quand on clique sur le bouton reset d'un formulaire
- **onDblClick** : (version 1.2) se produit quand on produit un double click sur la souris (ne fonctionne pas sur Macintosh ou le double clic a sa signification propre)
- **onDragDrop** : (version 1.2) se produit lorsqu'on fait un glisser lacher vers le navigateur, par exemple pour ouvrir une page HTML depuis son disque dur.
- **onKeyDown** : (version 1.2) se produit quand une touche du clavier est lachée.
- **onKeyPress** : (version 1.2) se produit quand une touche du clavier a été utilisée
- **onKeyUp** : (version 1.2) se produit quand une touche du clavier est appuyée
- **onMouseDown** : (version 1.2) se produit quand une touche de la souris est lachée.
- **onMouseMove** : (version 1.2) se produit quand le curseur de la souris est bougé
- **onMouseUp** : (version 1.2) se produit quand un bouton de la souris est rélaché
- **onMove** : (version 1.2) se produit quand une fenêtre est dépacée
- **onResize** : (version 1.2) se produit quand une fenêtre subit un changement de taille

6.6 - Leurs méthodes

Nom	Dépend de	Syntaxe	Description
abs	Math	Math.abs(nombre)	donne la valeur absolue d'un nombre
acos	Math	Math.acos(nombre)	Donne l'arc cosinus en radian d'un Nombre entre -1 et 1 (0 sinon)
alert	window	alert(message)	Affiche une fenêtre d'alerte et un bouton OK
anchor	string	text.anchor(nom)	Crée un signet dans la page HTML
asin	Math	Math.asin(nombre)	Retourne l'arc-sinus en radian d'un Nombre compris entre -1 et 1 (0 sinon)
atan	Math	Math.atan(nombre)	Retourne l'arc tangente en radian d'un nombre entre -pi/2 et pi/2
back	history	history.back()	revient d'un niveau dans la hiérarchie des pages lues
big	string	chaine.big()	positionne le couple de balises BIG (gros caractères) autour du texte
blink	string	chaine.blink()	positionne le couple de balises BLINK (clignotant) autour du texte
blur	password - select - text -textarea	password.blur() - select.text() - text.blur() - textarea.blu()	Retire le focus de l'objet sélectioné
bold	string	chaine.bold()	positione le couple de balises BOLD (gras)
ceil	Math	Math.ceil(nombre)	donne l'entier le plus proche par valeur supérieure
charAt	string	chaîne.charAt(index)	retourne le caractère index de la chaîne (commençant à 0)
clear	document	document.clear()	vide de contenu de la fenêtre
clearTimeout	frame - window	clearTimeout(timeoutID)	détruit la temporisation mise par setTimeout
click	button - checkbox - radio - reset - submit	élément.click()	simule un click sur l'élément donné
close (document)	document	document.close()	termine l'émission des données, le navigateur affiche alors un message de fin de réception.
close (window)	window	windowReference.close()	ferme ma fenêtre spécifiée
concat	array	tableau1.concat(tableau2)	concatène le tableau 2 à la suite du tableau 1
confirm	window	confirm(message)	demande la confirmation d'une action; renvoit 1 si le bouton OK est sélectionné et 0 si c'est le bouton annuler qui l'est.
cos	Math	Math.cos(nombre)	Retourne le cosinus d'un angle en Radian

Nom	Dépend de	Syntaxe	Description
escape	string	escape(chaine)	retoune en caractères ASCII des caractères ISO Latin-1 (& devient %26)
eval	interne	eval(chaine)	évalue les expressions dans une chaîne de caractères et renvoie la valeur correspondante
exp	Math	Math.exp(nombre)	retourne la valeur exponentielle d'une valeur
fixed	string	chaine.fixed()	positionne le couple de balises TT (fixe) autour du texte
floor	Math	floor(nombre)	donne l'entier le plus proche par valeur inférieure
focus	password - select - text - textarea	objet.focus()	Met le focus sur l'objet concerné
fontcolor	string	chaine.fontcolor(couleur)	donne la couleur donnée à la chaîne de caractères concernée (équivalent à `<FONT COLOR=couleur>`)
fontsize	string	chaine.fontsize(taille)	donne la taille de police donnée à la chaîne de caractères concernée (équivalent à `<FONT size=taille>`)

Nom	Dépend de	Syntaxe	Description
forward	history	history.forward()	donne la page suivante dans la hierarchie des pages lues et gardées en mémoire du navigateur
getDate	Date	Date.getDate()	Retourne le jour du mois de la date Concernée (entier de 1 à 31)
getDay	Date	date.getDay()	retourne le jour de la semaine de la date concernée (0=dimanche)
getHours	Date	date.getHours()	retourne l'heure de la journée de la date concernée (0 à 23)
getMinutes	Date	date.getMinutes()	retourne la minute de l'heure de la date concernée (0 à 59)
getMonth	Date	Date.getMonth()	Retourne le mois de la date concernée (0 à 11)
getSeconds	Date	Date.getSeconds()	Retourne la seconde de la minute de la date concernée (0 à 59)
getTime	Date	Date.getTime()	Retourne l'heure dans la valeur Numérique correspondante
getTimezoneOffset	Date	date.getTimezoneOffset()	donne la différence (en minutes) entre l'heure courante et l'heure GMT
getYear	Date	date.getYear()	donne le nombre d'années écoulées depuis 1900

Nom	Dépend de	Syntaxe	Description
go	history	history.go(delta | location)	recherche la page dans le cache avec des valeurs entières positives (en avant) ou négatives (arrière) ou par leur URL (location)
indexOf	string	chaine1.indexOf(chaine2,[depuis])	retourne la position de chaîne2 recherchée dans chaine1 à partir de la position depuis(-1 si pas trouvé)
isNaN	interne	isNaN(valeur)	sur les plates-formes Unix teste si la valeur est un entier
italics	string	chaine.italics()	positionne le couple de balises I (italique) autour du texte
lastIndexOf	string	chaine1.lastIndexOf(chaine2, [depuis])	retourne la dernière occurrence de chaine2 dans chaine1 à partir de la position depuis
link	string	texte.link(URL)	crée un lien hypertexte vers URL à partir de texte
log	Math	Math.log(nombre)	calcule le log de nombre
max	Math	max(nombre1, nombre2)	retourne le maximum entre deux nombres
min	Math	min(nombre1, nombre2)	retourne le minimum entre deux nombres

Nom	Dépend de	Syntaxe	Description
number	-	Number (objet)	Convertit un objet en un entier (par exemple une date)
open (document)	-	document.open(["mimeType"])	ouvre un stream destiné à être alimenté par les méthodes write et writeln. Le type du stream peut être text/html, text/plain, image/gif, image/jpeg, image/x-bitmap, plugIn
open (window)	window	[windowVar =][fenetre].open(URL, nom, options)	ouvre une nouvelle fenêtre avec les options suivantes (séparées par ,) toolbar[=yes|no]|[=1|0], location[=yes|no]|[=1|0], directories[=yes|no]|[=1|0], status[=yes|no]|[=1|0], menubar[=yes|no]|[=1|0], scrollbars[=yes|no]|[=1|0], resizable[=yes|no]|[=1|0], width=pixels, height=pixels
parse	Date	Date.parse(chainedate)	retourne le nombre de millisecondes depuis le 01/01/1970

Nom	Dépend de	Syntaxe	Description
parseFloat	interne	parseFloat(chaine)	analyse une chaîne de caractères et retourne sa valeur virgule flottante
parseInt	interne	parseInt(chaine [,base])	retourne la valeur entier de la chaîne de caractères dans la base indiquée
pop	array	nomdutableau.pop()	supprime le dernier élément du tableau et retourne sa valeur
Pow	Math	pow(base, exposant)	retourne base puissance exposant
prompt	window	prompt(message, [valeurpardefaut])	affiche une fenêtre de saisie avec un message et la valeur par défaut initialisée
push	array	nomdutableau.push(element1,..,elementn)	ajoute les éléments donnés en argument au tableau
random	Math	Math.random()	Retourne une valeur aléatoire entre 0 et 1
round	Math	round(nombre)	retourne l'entier le plus proche de la valeur donnée en argument
select	password - text - textarea	objet.select()	sélectionne la zone de saisie de l'objet correspondant
setDate	Date	Date.setDate(valeur)	Positionne le jour du mois de la Date concernée (entier de 1 à 31)
setDay	Date	Date.setDay(valeur)	positionne le jour de la semaine de la date concernée (0=dimanche)

Nom	Dépend de	Syntaxe	Description
setHours	Date	Date.setHours(valeur)	Positionne l'heure de la journée de la date concernée (0 à 23)
setMinutes	Date	date.setMinutes(valeur)	positionne la minute de l'heure de la date concernée (0 à 59)
setMonth	Date	date.setMonth(valeur)	positionne le mois de la date concernée (0 à 11)
setSeconds	Date	Date.setSeconds(valeur)	positionne la seconde de la minute de la date concernée (0 à 59)
setTime	Date	Date.setTime(valeur)	Positionne l'heure dans la valeur Numérique correspondante
setTimeout	frame - window	valeur=setTimeout(expression, msec)	évalue expression après un delai d'attente de msec.valeur permet d'arrêter l'évaluation avec la méthode clearTimeout
setYear	Date	date.setYear(valeur)	positionne le nombre d'années écoulées depuis 1900
shift	array	nomdutableau.shift()	retourne le premier élément du tableau et supprime celui-ci du tableau
sin	Math	Math.sin(angle)	retourne le sinus de l'angle donné en radian

Nom	Dépend de	Syntaxe	Description
small	string	chaine.small()	positionne le couple de balises SMALL (petites lettres) autour du Texte
sqrt	Math	Math.sqrt(nombre)	Retourne la racine carrée de Math.sqrt(number)
strike	string	chaine.strike()	positionne le couple de balises STRIKE (biffé) autour du texte
string	-	string (objet)	convertit un objet en chaîne de caractères (par exemple une date)
sub	string	chaine.sub()	positionne le couple de balises SUB(indice) autour du texte
submit	form	formulaire.submit()	provoque la soumition du formulaire au serveur http
substring	string	chaine.substring(position1, position2)	retourne la chaine de caractères commençant à la position1 et finissant à la position2
sup	string	chaine.sup()	positionne le couple de balises SUP (exposant) autour du texte
tan	Math	Math.tan(nombre)	Retourne la tangente du nombre Donné en radian

Nom	Dépend de	Syntaxe	Description
toGMTString	Date	Chainedate.toGMTString()	Convertit une date en chaîne de Caractères en suivant les conventions GMT (Mar, 01 Mar 1996 20:00:00 GMT)
toLocaleString	Date	chainedate.toLocaleString()	convertit une date en chaîne de caractères en suivant les conventions locales (03/01/96 20:00:00)
toLowerCase	string	chaine.toLowerCase()	convertit la chaine concernée en minuscule
toUpperCase	string	chaine.toUpperCase()	convertit la chaine concernée en majuscule
unescape	-	unescape(chaine)	retourne la valeur ASCII d'une valeur donnée en Ox.. (hexadécimal) ou en %.. (décimale)
unshift	array	nomdutableau.unshift(element1,..,element2)	ajoute les éléments donnés en argument au tableau
UTC	Date	Date.UTC(an, mois, jour [, h] [, min] [, sec])	Retourne le nombre de secondes Ecoulées depuis le 01/01/1970 0h0mn
write	document	write(chaine [,chaine], ...[,chaine])	écrit les chaines spécifiées dans le document
writeln	document	writeln(chaine [,chaine], ...[,chaine])	écrit les chaines spécifiées suivies d'un retour ligne dans le document