U.F.R. S.A.T.

(Unité de formation et de recherché des Sciences Appliquées et Technologie)

DIETEL

(Diplôme d'Ingénieur en Electronique et Télécommunication)

GETI (Génie en Electronique, Télécommunication et Informatique)

INTRODUCTION :

JavaScript est un langage de scripts qui incorporé aux balises Html permet d'améliorer la présentation et l'interactivité des pages Web. Les programmes JavaScript s'intègre dans le code HTML d'une page WEB. L'intérêt d'un langage comme JavaScript est de pouvoir contrôler dynamiquement le comportement d'une page Web : on peut par exemple vérifier que le code postal saisi dans la page est correct, faire afficher des menus spéciaux quand la souris approche

une zone donnée, afficher des bandeaux publicitaires animés, orienter automatiquement le visiteur sur une autre page, en gros JavaScript offre tout une panoplie pour manipuler dynamiquement une page web.

JavaScript est le langage C rendu dynamique et orienté-objet :

Pour les programmeurs expérimentés en C, C++, Java ou PHP ce résumé devrait être suffisant pour maîtriser le langage JavaScript et l'utiliser pour construire des pages Web dynamiques. Les types primaries sont: *boolean, number, string, array, object, function, undefined, Infinity, NaN*. Ils peuvent avoir des attributs et des méthodes. Javascript n'est pas sensible à la casse (différentie les majuscules des minuscules : java est différent de Java).

Les variables et les constantes en JavaScript :

Les variables sont dynamiques en JavaScript, ce sont des noms associés à des valeurs et que l'on peut réaffecter dans le programme à des valeurs de type différent: nombre, chaîne de caractères, tableau, etc.

Le nom d'une variable est une suite de lettres ou de chiffres, qui commence par une lettre ou le symbole de soulignement, ou le symbole $.
Par exemple:

```
_x
nom
nom2
$nom
```

Le symbol $ seul représente une variable. Il est utilisé par jQuery (bibliothèque de javascript)

Déclaration avec le mot réservé var *ou* const :

Une variable se déclare par utilisation du mot réservé *var*:

```
var x;
```

Dans ce cas ci-dessus il faudra qu'elle soit assignée avant que l'on ne l'utilise.
Elle se déclare aussi en assignant une valeur à un nom:

```
x = 24;
```

Une déclaration complète serait plutôt:

```
var x = 24;
```

A l'intérieur du corps d'une fonction, le mot-clé *var* est obligatoire pour créer une variable locale à cette fonction. Il ne l'est pas dans l'espace global ni pour les arguments des fonctions. Si la variable n'a pas de valeur assignée, son contenu est *undefined*, ce que l'on peut tester par une instruction de comparaison:

```
var y;
if(y == undefined)
{
   y = 0;
}
```

Les constantes se déclarent avec le mot réservé *const* (à la place de *var*) et sont assignées lors de la déclaration, puis il est par définition impossible de les modifier ultérieurement.

```
const x = 24;
```

Trois primitives et des objets :

Les primitives du langage sont:

- boolean
- string
- number

Ce sont des mots réservés et d'autres mots-clés ont été réservés pour l'avenir: byte, float, int, short, etc...

Pour connaître le type d'un variable, qui est définit par la valeur qui lui est assignée, on utilise *typeof*. Exemple:

```
document.write(typeof Boolean(true));
var x = "text";
document.write(typeof x);
```

```
boolean
string
```

Il existe des objets prédéfinis de même nom que ces primitives, mais capitalisés:

- Boolean
- Number
- String

Ils sont initialisés par un argument. Exemple:

```
var x = new Number(50);
var y = new String("texte");
```

Si l'on associe une propriété de l'objet correspondant à une primitive,, par exemple *length* à une chaîne, celle-ci sera transformée dynamiquement en objet pour cette instruction. Cela ne change pas le type de la variable au-delà de l'instruction.

Démonstration:

```
var a = "texte";
document.write(typeof a);
document.write(a.length);
a = new String("demo");
document.write(typeof a);
string
5
object
```

La visibilité est locale à une fonction ou globale :

Une variable est considérée comme globale si elle est déclarée hors d'une définition de fonction ou d'une structure. Elle est alors visible dans le corps des fonctions et dans toutes structures de l'espace global ou contenues dans des fonctions.

Une variable définie dans une fonction est visible dans cette fonction et dans le corps de toute structure contenue dans cette fonction.

Mais si elle est créée sans le mot clé *var*, elle fait alors partie de l'espace global, même si elle n'a pas été définie hors de la fonction.

```
function()
{
 var x = 1;
 y = 2;
}
```

La variable x est local tandis qu'y est globale.

Une variable définie globalement dans une fenêtre, peut être utilisée dans une autre si on l'associe au nom de la fenêtre, par exemple, x est défini dans une fenêtre avec le nom win2:

```
win2.x;
```

Des valeurs sont prédéfinies pour les tests conditionnels :

Certaines valeurs font partie du langage:

true

vrai, une valeur booléenne.

false

faux, la valeur booléenne opposée.

Undefined

variable non assignée. Ne correspond à aucun type possible.

NaN

Not A Number. Valeur indiquant que la variable ne contient pas un nombre. On peut assigner cette valeur, x = NaN, et la tester avec la fonction isNaN().

null

N'a aucune valeur assignée. C'est le null du langage C

L'objet Boolean et les valeurs booléennes en JavaScript :

L'objet Boolean peut contenir deux valeurs, *true* et *false*. Par ailleurs il existe en JavaScript les mots clés *true* et *false* qui peuvent être comparés au résultat d'une expression conditionnelle. Ce sont sémantiquement des choses différentes bien qu'elles aient les mêmes effets dans les opérations.

On peut créer un objet Boolean avec la syntaxe suivante:

```
var b = new Boolean(valeur)
```

La valeur étant 1 ou *true*, ou 0 ou *false*.

Quelques exemples d'utilisation de nombres booléens, avec le résultat affiché dans cette page sous le code...

```
var a = new Boolean(1);
document.write(a);
```

true

```
var a = new Boolean(0);
document.write(a);
```

false

Tout nombre différent de 0 vaut *true*.

```
var a = new Boolean(5);
document.write(a);
```

true

Une chaîne de caractère vaut *true*.

```
var a = new Boolean("texte");
document.write(a);
```

true

Même si elle est vide...

```
var a = new Boolean("");
document.write(a);
```

true

Le mot-clé *true* a la valeur booléenne *true*.

```
var a = new Boolean(true);
document.write(a);
```

true

Le mot-clé *false* a aussi la valeur booléenne *false*.

```
var a = new Boolean(false);
document.write(a);
```

false

JavaScript considère aussi comme *false* les valeurs des mots-clés suivants:

- null
- NaN
- undefined.

Exercices :

Jeux de déclaration et d'affichage avec la méthode de l'objet document : write().

Déclarer des variables de tous les types vus et afficher les types ainsi que les valeurs.

Correction :

Fichier code_jacascript.js

```
  nom="fall";

document.writeln(nom);

document.write("<br>");

document.writeln(typeof nom,"<br>");

nom=3;

nom+=4;

document.writeln(nom,"<br>");

document.writeln(typeof nom,"<br>");

nom= new String("name");

document.write(typeof nom,"<br>");

document.write(nom,"<br>");

var name;

document.write(typeof name,"<br>");

name="zeus";

if (isNaN(name))

   document.write(" ce n est pas un nombre ");
```

```
alert(" fin de l exercice ");
```

la page html sera :

```
<html>
<head>
<title> EXOS JAVASCRIPT </title>
<script type="text/javascript" src="code_javascript.js" >
</script>
</head>
<body>
</body>
</html>
```

On peut aussi inclure directement le code javascript dans les balises script :

```
<script type="text/javascript" >
  nom="fall";
document.writeln(nom);
document.write("<br>");
document.writeln(typeof nom,"<br>");
nom=3;
nom+=4;
document.writeln(nom,"<br>");
document.writeln(typeof nom,"<br>");
nom= new String("name");
document.write(typeof nom,"<br>");
document.write(nom,"<br>");
```

```
var name;

document.write(typeof name,"<br>");

name="zeus";

if (isNaN(name))

  document.write(" ce n est pas un nombre ");

  alert(" fin de l exercice ");

</script>
```

Les opérateurs de JavaScript et outil de test :

Le langage C a transmis comme à beaucoup d'autres langages ses opérateurs à JavaScript. Les principaux symboles que contient un clavier sont mis à contribution pour former l'éventail des opérations possibles dans une expression.

Opérateurs arithmétiques :

Ce sont des opérateurs binaires, sauf la négation, et ils ont donc la forme:

x = a + b
+ addition
- soustraction
* multiplication
/ division
% modulo. Retourne la reste de la division de deux nombres.
++ incrémentation, simplifie l'addition de 1.
-- décrémentation, simplifie la soustraction de 1.
- négation, génère le négatif d'un nombre. Sous la forme x = -y

Noter que la division de deux entiers peut produire un entier s'il n'y a pas de reste, ou un nombre réel dans le cas contraire. Dans les langages typés, cela est différent.

Les chaînes de caractères ont deux opérateurs:

+ concaténation de deux chaînes: a = str1 + str2
+= concaténation d'une autre chaîne: a += str2

Les opérateurs + et - permettent de convertir une chaîne de caractères en nombre de la même façon que la méthode parseInt. Contrairement à l'opérateur +, - change en plus le signe du nombre.
Le code suivant illustre la mise en œuvre des opérateurs + et - :

```
var chaine = "10"; // chaine est de type string
var nombre = +chaine; // nombre est de type number
var autreNombre = -chaine; // autreNombre est de type number et l'opposé de nombre
```

Opérateurs de comparaison :

Ce sont des opérateurs binaires qui s'emploient dans les tests de comparaison.

Ils comparent des nombres ou des expressions qui retournent *true* si la comparaison réussit ou *false* si elle échoue. Quand les opérandes sont de types différents, ils sont convertis avant comparaison.

```
if(x == 5) ...
==   égal
<    inférieur
>    supérieur
<=   inférieur ou égal
>=   supérieur ou égal
!=   différent
```

Deux chaînes sont égales si elles ont la même taille et les mêmes caractères aux mêmes positions.
Deux variables correspondant à des objets sont égales s'il s'agit du même objet assigné à deux variables différentes.

Les opérateurs stricts :

Lorsque les opérandes d'une comparaison stricte sont de types différents, le résultat est toujours *false*, quel que puissent être les valeurs.

```
===  égalité stricte
!==  différence stricte
```

Les codes *null* et *undefined* ont une égalité simple, mais pas stricte.

Opérateurs logiques :

Ils s'emploient dans les tests de condition, comme les opérateurs de comparaison:

if(a && b) ...
&& et logique. Vrai si les deux opérandes sont vrais.
|| Ou logique. Vrai si l'un des deux opérandes est vraie.
! Négation logique. Vrai si l'opérande est faux.

Opérateurs d'assignement :

L'assignement simple s'effectue avec l'opérateur =

a = 5

Mais on peut combiner l'assignement avec d'autres opérations lorsque la variable a assigner est un des deux opérandes, par exemple:

a += b équivaut à a = a + b
+= ajoute une valeur.
-= soustrait.
*= multiplie par l'opérande. Ex: a *= 3, si a valait 2, elle vaudra 6.
/= divise par l'opérande.
%=
<<= décale à gauche en mettant des zéros à droite. Ex: a <<= 2. si a valait 10, elle vaudra 40 après deux décalages
>>= décale à droite.
>>>= décale à droite, ignore le bit de signe.
&= effectue un et binaire avec l'opérande.
|= effectue un ou binaire.
^= effectue un ou exclusif binaire. Ex: a ^=2, si a valait 4, elle vaudra 16.

Opérateurs sur les bits :

Ils permettent d'effectuer des opérations sur la représentation binaire des nombres. Ainsi 0110 & 0010 retourne 0010.

& et binaire
| ou binaire
^ ou exclusif
~ complément
a << b décalage à gauche selon b. Ex: x = a << b. Si a vaut 0001 et b vaut 2, le résultat sera 0100.
a >> b décalage à droite avec préservation du bit le plus à gauche. C'est l'opération inverse.
a >>> b décalage à droite avec remplissage par des zéros.

La différence entre >> et >>> est que le bit le plus à gauche, qui est le bit de signe est préservé dans le premier cas et non dans le second. Cela permet de préserver le signe dans le

premier cas, tandis que dans le second, la variable est supposée être utilisée comme stockage de bits et le bit le plus à gauche n'a pas rôle de signe.

Autres opérateurs :

Ils ont des rôles divers dans le langage selon le contexte.

. le point associe un membre à un objet
[ ] les crochets enclosent les indices.
() les parenthèses regroupent les expressions.

, la virgule est un séparateur.
? : cette construction signifie pour exp ? action1 : action2, si exp vaut true, exécuter action1 sinon action2.
new précède un constructeur. Crée un objet.
this désigne un variable interne à un objet.
in appartenance. Dans x in y, la condition est vraie si x fait partie des éléments de y.
delete supprime un objet, une variable.
instanceof x instanceof y retourne vrai si l'objet x est une instance de y.
typeof typeof x affiche une chaîne qui indique le type de x.

Méthodes de détection de types et de validité des variables :

isArray Détermine si le paramètre est un tableau.

isBoolean Détermine si le paramètre est un booléen.

isEmpty Détermine si un tableau est vide.

isFinite Détermine si le paramètre correspond à un nombre fini.

isFunction Détermine si le paramètre est une fonction.

isNaN Détermine si la valeur du paramètre correspond à NaN (Not a Number).

isNull Détermine si le paramètre est null.

isNumber Détermine si le paramètre est un nombre.

isObject Détermine si le paramètre est un objet.

isString Détermine si le paramètre est une chaîne de caractères.

isUndefined Détermine si le paramètre est indéfini, c'est-à-dire une référence non initialisée.

Les méthodes escape et unescape offrent respectivement la possibilité d'encoder et de décoder des chaînes afin qu'elles puissent être utilisées dans des pages HTML. Le code suivant donne un exemple de leur utilisation :

```
var chaine = "Ceci est une chaîne de caractère!";

var chaineEncodee = escape(chaine);

     /* chaineEncodee contient « Ceci%20est%20une%20cha%EEne%20de%20caract%E8re%21 » */

var chaineDecodee = unescape(chaineEncodee);

    // chaineDecodee contient la même chose que la variable chaine
```

Exemple de conversion de type :

```
var booleen = true;

var variable1 = booleen.toString();

      // variable1 contient la chaîne de caractère « true »

var nombreEntier = 10;

var variable2 = nomEntier.toString();

       // variable2 contient la chaîne de caractère « 10 »

var nombreReel = 10.5;

var variable2 = nombreReel.toString();

        // variable2 contient la chaîne de caractère « 10.5 »

var nombreEntier = 15;

var variable1 = nombreEntier.toString();

              // variable1 contient la chaîne de caractère « 10 »

var variable2 = nombreEntier.toString(2);

              // variable2 contient la chaîne de caractère « 1111 »

var variable4 = nombreEntier.toString(16);

            // variable4 contient la chaîne de caractère « f »
```

```
var entier1 = parseInt("15");

                    // entier1 contient le nombre 15

var entier2 = parseInt("0xf");

                    // entier2 contient le nombre 15

var entier3 = parseInt("f", 16);

                    // entier3 contient le nombre 15

ar reel = parseFloat("15.5");

                    // reel contient le nombre réel 15,5
```

Un exemple de code HTML :

```
<html>

<head>

<title> EXOS JAVASCRIPT </title>

<script type="text/javascript">

 a=prompt(" donner l entier a ");

 b=prompt(" donner l entier b ");

 a=parseInt(a);

  b=parseInt(b);

 alert(" la somme est "+(a+b));

 a=prompt(" donner le nom ");

 b=prompt(" donner le prenom ");

 alert(" la somme est "+(a+b));

 a=100;

 a+=2;

  c=a.toString();    // bug qui peut intervenir : une chaine n est jamais converti en une
chaine. c est logique non!

 alert("en chaine "+c);

  c=a.toString(2);

 alert(" en base binaire  "+c);

 c=a.toString(16);

 alert(" en base hexadecimal  "+c);

 c=a.toString(8);
```

```
  alert(" en base octale  "+c);
 </script>
</head>
<body>
</body>
</html>
```

Précédence des opérateurs :

On peut se dispenser des parenthèses dans une expression si la précédence permet de determiner quels opérandes sont concernés par un opérateur. Ainsi (x * y) + z est équivalent à x * y + z, par ce que * à la précédence sur +.

Dans l'ordre de précédence:

. [] le point et les crochets sont prioritaires sur tous autres et ensuite
() new
! ~ - + ++ -- typeof void delete: tous les opérateurs unaires
* / %
+ -
<< >> >>>
< > <= >= in instanceof
== != === !==
&
^
|
&&
| |
?:
= et autres opérateurs d'assignement
, la virgule a la priorité la plus faible.

JavaScript et C: Héritage et différences :

En JavaScript:

1. Les identificateurs sont sensibles à la casse.
2. Les mots-clés doivent être en minuscules.
3. Le point-virgule est optionnel.
4. Les fonctions peuvent être des objets.
5. Une fonction peut contenir d'autres fonctions (elles servent de méthodes).
6. Les variables ont des types dynamiques.
7. Les tableaux sont dynamiques et associatifs (ceci ressemble à PHP mais vient de AWK).
8. Les expressions régulières font partie du langage ce qui vient de Perl.
9. On crée des prototypes pour gérer l'héritage d'objets.
10. *with* est ajouté à JavaScript.

Les différences sont faites pour rendre le langage plus dynamique et faciliter la création de scripts.

Les points importants de la syntaxe JavaScript :

Instructions et séparateurs, le cas du point virgule :

Les instructions ne sont pas limitées par la fin de ligne (sauf le commentaire simple) et ont pour unique séparateur le point-virgule. Cependant quand une instruction est considérée comme complète et que l'interpréteur rencontre un espace, une fin de ligne, une tabulation, il insère virtuellement un point-virgule. Il vaut mieux toutefois ne pas compter là-dessus pour ne pas rendre le code illisible.

La virgule est un séparateur interne aux instructions, par exemple dans l'en-tête d'une fonction elle sépare les arguments.

Les parenthèses () regroupent des listes ou isolent une expression.
Les crochets [] servent à indicer un tableau.

La syntaxe des commentaires et blocs d'instructions est celle de C :

Un commentaire simple débute par une double barre inclinée // et se poursuit jusqu'en fin de ligne.

Un commentaire fermé commence par le couple de symboles /* et se termine par */ sans que les fins de lignes n'aient de rôle à jouer autre que la mise en forme.
Il peut comporter une seule ligne ou plusieurs.

```
... code ...    /* commentaire */
```

Un ensemble d'instructions est isolé par les opérateurs "{" pour le début du bloc et "}" pour la fin. Cet ensemble est généralement associé à une commande comme if(condition), ou par une instruction de boucle.
Il peut être utilisé tout simplement, sans grande utilité, mais sans que ce soit une erreur de syntaxe.

Un bloc d'instructions est considéré comme une instruction, ainsi:

```
if(x == 5) document.write("ok");
```

est équivalent à:

```
if(x == 5)
{
   document.write("ok");
}
```

Avec la différence que dans le second cas on peut regrouper un ensemble d'instructions.

La clause* with *a été ajoutée à JavaScript :

C'est un exemple de différence de syntaxe avec le langage C.

Son rôle est de définir un sous ensemble dans lequel on utilise les méthodes et attributs d'un objet sans mentionner cet objet.

Exemple:

```
var x = new String("--> hello");
with(x)
{
   document.write(toUpperCase());
}
```

```
--> HELLO
```

Dans cet exemple, la méthode *toUpperCase*() de l'objet String est appliquée implicitement à l'instance x désignée par *with.*

Un exemple plus utile serait de placer document en *with*, on peut alors faire un code plus simple en utilisant *write* ou d'autres méthodes comme ***getElementById*** en série.

```
with(document)
{
  write("--> hello");
}
```

--> hello

Les tableaux en JavaScript sont dynamiques :

Comme tous les langages de script, JavaScript a des tableaux dynamiques: leur taille n'est pas prédéterminée, ni le type des données contenues.

On les crée avec un littéral ou un constructeur

Avec le mot réservé *new*.

```
var x = ["a", "b", "c"];   // littéral avec initialisation

var x = new Array();      //  constructeur
var y = new Array("a", "b", "c");   // constructeur avec éléments
```

Noter que si le constructeur Array() contient en argument un seul élément numérique, cet élément correspond à la taille du tableau et n'est pas un élément du tableau.

```
var x = new Array(5);    // tableau vide prévu pour 5 éléments.
var x = new Array(1, 2); // tableau contenant deux éléments, les nombres 1 et 2.
```

Cela peut sembler peu cohérent mais cela n'a en fait pas d'importance, car si on veut créer un tableau de quelques éléments, on utilisera plutôt un littéral. Exemple:

```
var x = [5];           // tableau contenant un seul élément, le nombre 5.

var tableau1 = []; // Initialisation d'un tableau vide
```

On peut le vérifier avec cet exemple...

On construit le tableau Array(5). On affiche le tableau, qui est vide, puis la taille.
Ensuite on crée un tableau avec le constructeur Array(1,2), donc contenant deux éléments.

```
var x = new Array(5);
document.writeln(x);
document.writeln(x.length);


x = new Array(1, 2);
document.writeln(x);
```

```
,,,,
5
1,2
```

La valeur littérale d'un tableau peut contenir des variables :

La liste des éléments se place entre crochets et contient tout type de valeurs, numériques, chaînes, objets, séparées par une virgule.

Les virgules servent à spécifier le nombre d'éléments, elles peuvent contenir des blancs:

```
x = [ "a", , , , "e"]
```

La lettre "e" sera en position 4 dans le tableau. Les virgules terminales sont ignorées.

Une variable peut être donnée comme élément d'un tableau littéral, c'est alors son contenu qui sera assigné dynamiquement au tableau.

Pour un tableau littéral à deux dimensions, on écrira:

```
x = [[ "a", "b", "c"] , [ 1,2,3]]
```

L'indexation d'un tableau peut lui donner sa taille :

Le premier élément a la position 0. On assigne ou lit le contenu d'un tableau avec l'indice de position entre crochets.

```
var x = new Array();
alert(x[0]);        // lire
x[0] = "car";       // assigner
```

Si on assigne à une autre position du tableau, par exemple en indice 5, le tableau sera automatiquement redimensionné à la taille de 6 éléments au minimum, si sa taille antérieure était nulle ou inférieure à 6.

On assigne des tableaux comme éléments pour ajouter une dimension :

On construit un tableau multi-dimensionnel en donnant à un tableau des éléments qui sont d'autres tableaux.

```
var a = new Array( new Array(1, 2, 3), new Array(4, 5, 6));
document.write(a);
```

Résultat: 1,2,3,4,5,6

```
var a = [ [ 1, 2, 3], [ 4, 5, 6] ];
document.write(a);
```

Résultat: 1,2,3,4,5,6

```
a[0][0] = 1;
a[0][1] = 2;
a[0][2] = 3;
a[1][0] = 4;
a[1][1] = 5;
a[1][2] = 6;
document.write(a);
```

Résultat: 1,2,3,4,5,6

On peut aussi accéder aux éléments du tableau avec les indices en chaînes, par exemple:

```
document.write(a[1][2]);
```

6

Essayer interactivement les méthodes pour accéder aux éléments ou transformer le tableau :

Comme pour tout objet, on peut invoquer des méthodes associées au nom d'une instance de Array.

```
var x = new Array();
x.push("a");
```

Description de toutes les méthodes

void push(x)

Ajoute un élément à la fin du tableau.

```
var queue = [];
//Ajout d'éléments à la queue
pile.push("element1");
pile.push("element2");
//Récupération du premier élément de la queue
var elementQueue = pile.shift();
//elementQueue contient la valeur « element1 »
```

var y = x.pop()

Retourne le dernier élément et l'enlève du tableau.

```
var pile = [];
//Ajout d'éléments à la pile
pile.push("element1");
pile.push("element2");
//Récupération du premier élément de la pile
var elementPile = pile.pop();
//elementPile contient la valeur « element2 »
```

String s = x.join([sep])

Crée une chaîne de caractères à partir des éléments d'un tableau. Ne modifie pas le tableau.

[sep] est un séparateur optionnel en paramètre.

```
var tableau = [ "element1", "element2", "element3", "element4" ];
var chaine = tableau.join(",");
//chaine contient « element1,element2,element3,element4 »
```

Array z = x.concat(y)

Concatène le contenu du tableau x avec le tableau y et retourne un nouveau tableau z.

Le tableau de départ x n'est pas modifié.

```
var tableau = [ "element1", "element2" ];
var tableauAAjouter = [ "element3", "element4" ];
//Retourne un tableau correspondant à la concaténation des deux
var tableauResultat = tableau.concat(tableauAAjouter);
```

void sort(fonction)

Classe les éléments selon un ordre qui applique la fonction donnée en argument.

```
var tableau = [ "trois", "quatre", "cinq", "six" ];
var resultat = tableau.sort();
/*resultat correspond au tableau [ "cinq", "quatre", "six", "trois" ] */
```

Array x = slice(int début [, int fin])

Retourne un sous-ensemble du tableau qui reste inchangé.

```
var tableau = [ "element1", "element2", "element3", "element4" ];
var resultat = tableau.slice(1,3);
/*resultat correspond au tableau [ "element2", "element3" ] */
```

void splice(int pos, int nombre, element0, element1, ...)

A partir de la position pos, supprime le nombre d'éléments et insère les éléments de la liste. Si on supprime 0 éléments il faut au moins ajouter un élément.
Avec splice, le contenu du tableau lui-même change.

```
var tableau = [ "element1", "element2", "element3", "element4" ];
//Remplacement de deux éléments dans le tableau
var tableauResultat = tableau.splice(1,2,"nouveau element2","nouveau element3");
/* tableauResultat est équivalent à [ "element1", "nouveau element2",
➥"nouveau element3", "element4" ] */
//Suppression des deux éléments ajoutés précédemment
tableauResultat = tableau.splice(1,2);
// tableauResultat est équivalent à [ "element1", "element4" ]
```

var x =shift()

Retourne le premier élément et l'enlève du tableau puis décale les éléments vers la gauche

void unshift(x)

Insère un élément au début du tableau en décalant le contenu d'une position.

String toString()

Fournit une représentation en chaîne de caractère du tableau. Comme join() mais avec virgule comme séparateur obligé.

void reverse()

Inverse l'ordre des éléments contenus.

```
var tableau = [ "element1", "element2", "element3", "element4" ];
var resultat = tableau.reverse();
/*resultat correspond au tableau [ "element4", "element3", "element2", "element1" ] */
```

Tableau associatif en JavaScript :

Les tableaux associatifs sont des objets dynamiques que l'utilisateur redéfinit selon ses besoins. Quand on assigne des valeurs à des clés dans une variable de type Array, le tableau se transforme en objet, et il perd les attributs et méthodes de Array. L'attribut *length* n'est plus disponible car la variable n'a plus le type *Array*.

Nous allons faire la démonstration de tout cela et aussi montrer comment ajouter une méthode essentielle à un objet pour avoir le nombre d'éléments contenus quand il devient un tableau associatif.

Un tableau associatif est déclaré ou créé dynamiquement :

On peut le créer en assignant un littéral à une variable.

```
var x = { "un" : 1, "deux" : 2, "trois": 3 };
```

On a en fait *implicitement* créé une variable de type *Object*.

On peut *explicitement* créer un objet et lui assigner des clés et des valeurs:

```
var o = new Object();
o["un"] = 1;
o["deux"] = 2;
o["trois"] = 3;
```

Le propre des objets en JavaScript est que *les attributs sont aussi des clés* comme on va le vérifier dans la démonstration.
Ainsi, le même tableau peut être créé plus simplement:

```
var oa = new Object();
oa.un = 1;
oa.deux = 2;
oa.trois = 3;
```

On accéde au contenu par les clés :

Quelle que soit la méthode de création utilisée, on accède au contenu soit par les clés.

```
var y = o["un"];
```

Pour obtenir le contenu, élément par élément, on utilise une boucle *for*:

```
for(var i in tab)
{
```

```
    document.writeln(i + "=" + tab[i]);
}
```

Les clés sont assignées à la variable i, laquelle permet d'obtenir la valeur correspondante.

Pour connaître le nombre d'éléments, il faut créer une fonction :

Puisque nous ne disposons plus de l'attribut *length* de l'objet *Array*, il reste à ajouter une méthode à *Object* qui retourne la taille de la liste:

```
Object.size = function(tab)
{
   var size = 0;
   for (var key in tab)
   {
      if (tab.hasOwnProperty(key)) size++;
   }
   return size;
};
```

On obtient le nombre d'élément ainsi:

```
var s = Object.size(x);
```

On peut aussi définir une simple fonction qui prend l'objet en argument:

```
function size(arr)
{
   var size = 0;
   for (var key in arr)
   {
      if (arr.hasOwnProperty(key)) size++;
   }
   return size;
};
```

On obtient le nombre d'élément ainsi:

```
var s = size(x);
```

Démonstrations de tableaux associatifs :

Création de tableaux associatifs selon différentes méthodes et accès aux éléments.

Créer un tableau associatif avec un littéral :

```
var x = { "un" : 1, "deux" : 2, "trois": 3 };
for(var i in x)
{
    document.writeln(i + "=" + x[i]);
}
un=1 deux=2 trois=3
```

Créer le tableau avec un objet :

```
var o = new Object();
o["un"] = 1;
o["deux"] = 2;
o["trois"] = 3;
for(var i in o)
{
    document.writeln(i + "=" + o[i]);
}
un=1 deux=2 trois=3
```

Les attributs d'un objet JavaScript sont aussi des clés :

```
var oa = new Object();
oa.un = 1;
oa.deux = 2;
oa.trois = 3;
for(var i in oa)
{
    document.writeln(i + "=" + x[i]);
}
un=1 deux=2 trois=3
```

L'attribut *length* n'a aucune valeur.

```
x.length: undefined
x instanceOf Array: false
```

Maintenant on ajoute une méthode *size* comme ci-dessus:

```
Object.size(x): 3
```

Ou une simple fonction:

```
size(x): 3
```

Parcourir un tableau en JavaScript :

Démonstration et comparaison des trois méthodes pour parcourir le contenu d'un tableau. Nous allons calculer le temps d'exécution de for(), for in et forEach.

for ( )

Une boucle simple qui incrémente un indice et accède aux éléments successifs par cet indice.

```
var x = ["un", "deux", "trois" ];
for(var i= 0; i < x.length; i++)
{
    document.write(x[i]);
}
```

C'est la méthode la plus rapide.

for in

Une boucle for in est l'équivalent du foreach de PHP et autres langages, elle assigne directement les éléments du tableau à une variable.

```
for(var i in x)
{
    document.write(i);
}
```

Cette méthode est la moins performante.

Firefox offre une syntaxe particulière.

```
for each(var i in x)
{
    document.write(i);
}
```

Mais ce n'est pas compatible avec d'autres navigateur et donc il est inutile de s'y attarder.

La méthode forEach

Elle a été ajoutée à la version 1.6 de JavaScript et est supportée par Firefox et Webkit mais pas par Internet Explorer.

```
x.forEach(function(y)
```

```
{
  document.write(y);
}
);
```

On peut associer la méthode directement à un tableau litéral:

```
["un", "deux", "trois"].forEach(function(y) { document.write(y); });
```

Si l'on aime le code compact, cela convient bien, mais c'est moins performant que la boucle for simple.

D'autres méthodes sont encore possibles mais peu lisibles.

Les structures de contrôle en JavaScript :

Les boucles et conditions de JavaScript sont celles du langage C mais elles sont étendues et plus flexibles. Par exemple, le *switch case* accepte tout type de valeurs.

if else
for
for in
while
break et continue
do while
switch case

if ... else

La syntaxe pour l'exécution conditionnelle de code est:

```
if(condition) { }
```

ou

```
if(condition) { } else { };
```

Les accolades sont optionnelles s'il y a une seule instruction tandis que les parenthèses sont toujours obligatoires.
Quand l'évaluation de la condition retourne *true*, la ou les instructions sont exécutées sinon la partie *else* le sera si elle est présente.

Exemple:

```
if(a == 5)
{
   document.write("a vaut 5");
}
```

Il serait possible comme en C d'assigner une variable à l'intérieur de la condition, une pratique à éviter.

for

Pour exécuter en boucle une série d'instruction, la syntaxe est:

```
for(var = initialiseur; condition; incrémentation)
{
 ...instructions...
}
```

Exemple:

```
for(var i = 0; i < 10; i++)
{
   document.write(i + "<br>");
}
```

C'est la formulation de base. C'est aussi la plus rapide, comme le montrent les benchmarks sur les diverses formes de boucle en JavaScript.
Mais il existe d'autres formulations, plus simples à écrire.

for in

Cette structure permet de parser le contenu d'un objet pour accéder à la liste de ses propriétés et leurs valeurs. S'il s'agit d'un tableau, les propriétés sont les indices du tableau.

Syntaxe:

```
var arr = ["a", "b", "c"];
for(x in arr)
{
  document.write(arr[x]);
}
```

Voici des exemples d'utilisation de la structure de controle *for in* :

1) for in avec un tableau

La structure *for in* assigne les indices du tableau à une variable et l'on utilise celle-ci pour indexer le tableau.

```
var a = new Array("un","deux","trois","quatre","cinq");
a[5]="six";
for(x in a)
{
  document.write(x + ") " + a[x] +  "<br>");
}
```

0) un
1) deux
2) trois
3) quatre
4) cinq
5) six

2) for in avec un objet

La structure *for in* assigne le nom des propriétés d'un objet à une variable et l'on utilise celui-ci pour retrouver la valeur de la propriété de l'objet.

```
function voiture()
{
 this.passagers = 5;
 this.vitesse = 250;
 this.roues = 4;
 this.prix = 10000;
}
var v = new voiture();
for (x in v)
{
  document.write(x + ":" + v[x] + "<br>");
}
```

passagers:5
vitesse:250
roues:4
prix:10000

Autre exemple, *for in* dans une fonction avec un objet...

```
function scanObject(obj, nom)
{
  var str = "";
  for(i in obj)
  {
      str += nom + "." + i + " = " + obj[i] + "
";
  }
  return str;
}
v = new voiture();
v.prix = 5000;
document.write(scanObject(v, "voiture"));
```

```
voiture.passagers = 5
voiture.vitesse = 250
voiture.roues = 4
voiture.prix = 5000
```

3) for each (non standard)

For each obtient directement le contenu de l'objet et fonctionne comme la fonction foreach de PHP. Cette structure à été ajoutée à JavaScript 1.6 et puisqu'Internet Explorer 7 ne reconnaît que la version 1.5, elle ne fonctionnera pas avec ce navigateur. A ne pas utiliser sur un site public, donc.
Syntaxe avec toujours le même tableau:

```
for each(x in arr)
{
  document.write(x);
}
```

while

Pour une boucle qui exécute tant que la condition donnée est vraie, ce qui implique que l'expression conditionnelle contienne une variable qui est modifée dans le corps de la boucle:

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

Exemple:

```
var i = 0;
while(i < 3)
{
  document.write(arr[i]);
  i++;
}
```

un
deux
trois

Il est facile d'oublier d'incrémenter la variable de la condition ce qui provoque une boucle sans fin et le blocage du navigateur. A utiliser donc avec précautions. La boucle *for* de l'exemple suivant aura le même résultat et est donc préférable de même que *for in*.

```
for(i = 0; i < 3; i++)
{
   document.write(arr[i]);
}
```

break et continue

break permet de sortir de la boucle, tandis que *continue* passe à l'itération suivante.
Dans l'exemple, on crée une boucle sans fin avec une condition *true* qui sera évidemment toujours vraie, et l'on compte sur la commande *break* pour sortir de la boucle au moment choisi:

```
var arr = ["a", "b", "c", "d", "e"];
var x = 0;
while(true)
{
 if (x == 2) { x++;  continue; }
 if (x == arr.length) break;
 document.write(arr[x]);
 x++;
}
```

La chaine "c" n'est pas affichée car on continue quand on arrive à l'indice 2. La boucle s'arrête quand la taille du tableau est atteinte, grâce à l'instruction *break*:

a
b

d
e

Si une boucle *while* peut facilement tourner à l'infini, les choses s'aggravent encore avec l'utilisation de la commande *continue* car celle-ci peut passer l'instruction d'incrémentation de la variable de la condition qui permet de sortir de la boucle, ici avec l'instruction *break*.

La possibilité d'associer une étiquette à *continue* ne nous aide pas à éviter cet inconvénient car l'étiquette doit être déclarée avant l'utilisation de *continue*.

```
label:
...instructions...
continue label;
```

Ce que rend l'option de peu d'intérêt.

do ... while

do while équivaut à la structure *while* avec la condition reportée en fin de boucle. Le contenu de la boucle sera toujours exécuté au moins une fois.

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

Exemple:

```
var i = 0;
do
{
  document.write(arr[i]);
  i++;
} while(i < 3);
```

a
b
c

Dans le cas présent, il n'y a pas de différence avec *while*. La différence n'intervient que si la condition n'est jamais respectée. Par exemple (i > 4) n'affiche rien avec *while* et affiche "a" avec *do while*.

switch case

Exécute un traitement au choix en fonction de la valeur d'une expression conditionnelle.

Syntaxe:

```
switch(expression)
{
  case valeur:
     ... instructions ...
     break;
  case valeur:
     ...instructions...
     break;
  default:
  ....
}
```

Les cas sont comparés successivement, c'est la première valeur qui corresponde à l'expression qui est retenue, et le code associé est exécuté. L'instruction *break* marque la fin du code pour le cas, donc si *break* est omis, le code du cas suivant est exécuté à son tour. Et le mot réservé *default* permet d'activer une procédure de secours quand aucun cas n'est vérifié.

On peut assigner des variables et des expressions dans les cas, exemple:

```
var x = 10;
var y = 8;
switch(x)
{
  case 5: break;
  case y + 2:
    document.write("x = y + 2");
    break;
  default:
    document.write("Inconnu");
}
```

x = y + 2

En gros, JavaScript hérite de la syntaxe et des structures peu sûres du langage C, ce qui inclut l'aberration de l'assignement à l'intérieur d'une condition, avec des effets amplifiés en environnement client-serveur. Il faut être attentif aux boucles infinies éventuelles et pour les performances privilégier *for* ou *for .. in*.
Mai s il offre une liberté supérieure grâce à sa nature dynamique.

Exercices :

- saisir un tableau de n éléments et l'afficher avec une boucle.
- réaliser une page web capable de saisir une matrice 3x3(avec la fonction valeur=prompt(« donner valeur ») ;), calcule le déterminant, affiche les éléments avec une boucle ainsi que la valeur du déterminant. La matrice sera un tableau associatif où les clés seront les indices de case de la matrice comme Mij=valeur.

Correction :

```
<html>

<head>

<title> EXOS JAVASCRIPT  </title>

<script type="text/javascript">

tab = [];

for(i=0;i<=5;i++)

  {  tab[i]= prompt("donner lelement num "+i);  }



for(x in tab)

{

  document.write(x ,") ",tab[x], "<br>");

}

var tab2=tab.join(" ");

var tab3=tab.sort();

with (document)

{
```

```
for(x in tab)

write(x ,") ",tab2[x]);

}

var
b={"c1":{"m11":0,"m12":0,"m13":0},"c2":{"m21":0,"m22":0,"m23":0},"c3":{"m31":0,"m32":0,"m
33":0}};

 for(i in b )

                    {

       for(j in b[i])

                                        {

                                        b[i][j]=prompt("donner l'element  "+j);

                              }

                     }

                     document.write("<br>")

                     for(i in b )

                    {

       for(j in b[i])
```

```
                        {
                        document.write(" ",j," = ",b[i][j],"    ");
                    }
          document.write("<br>")
          }
 </script>
</head>
<body>
</body>
</html>
```

Manipulation des chaînes :

JavaScript gère les chaînes de caractères de manière similaire à d'autres langages tels que java. Ces dernières peuvent cependant être définies littéralement, aussi bien avec des guillemets ou des apostrophes, comme dans le code suivant :

```
var chaine1 = "ma chaîne de caractère";
var chaine2 = 'mon autre chaîne de caractère';
```

Le langage JavaScript introduit la classe String correspondante. Le code suivant illustre la façon de créer une chaîne de caractères par l'intermédiaire de cette classe :

```
var chaine1 = new String("ma chaîne de caractère");
```

Méthodes de manipulation de chaînes de la classe String :

charAt(Index du caractère dans la chaîne) : Retourne le caractère localisé à l'index spécifié en paramètre.

charCodeAt (Index du caractère dans la chaîne) : Retourne le code du caractère localisé à l'index spécifié en paramètre.

concat (Chaîne à concaténer) : Concatène la chaîne en paramètres à la chaîne courante.

fromCharCode (Chaîne de caractères Unicode) : Crée une chaîne de caractères en utilisant une séquence Unicode.

indexOf (Chaîne de caractères) : Recherche la première occurrence de la chaîne passée en paramètre et retourne l'index de cette première occurrence.

```
var uneChaine = "Le début de la chaine. Sa fin.";

var indicePointDebut = uneChaine.indexOf(".");

//La valeur de indicePointDebut est 10

var indicePointFin = uneChaine.lastIndexOf(".");

//La valeur de indicePointFin est 21
```

lastIndexOf (Chaîne de caractères) : Recherche la dernière occurrence de la chaîne passée en paramètre et retourne l'index de cette dernière occurrence.

match(Expression régulière) :Détermine si la chaîne de caractères comporte une ou plusieurs correspondances avec l'expression régulière spécifiée.

replace (Expression régulière ou chaîne de caractères à remplacer puis chaîne de remplacement) : Remplace un bloc de caractères par un autre dans une chaîne de caractères.

```
var uneChaine = "Le début de la chaine. Sa fin.";

var resultat = uneChaine.replace(". "," ; ");

//resultat contient « Le début de la chaine ; Sa fin. »
```

search (Expression régulière de recherche) : Recherche l'indice de la première occurrence correspondant à l'expression régulière spécifiée.

slice (Index dans la chaîne de caractères) : Retourne une sous-chaîne de caractères en commençant à l'index spécifié en paramètre et en finissant à la fin de la chaîne initiale si la methode ne comporte qu'un seul paramètre. Dans le cas contraire, elle se termine à l'index spécifié par le second paramètre.

split (Délimiteur) : Permet de découper une chaîne de caractères en sous-chaînes en se fondant sur un délimiteur

```
var uneChaine = "Le début de la chaine. Sa fin.";

var resultat = uneChaine.split (".");

/* resultat est un tableau contenant les éléments « Le début de la chaine »,

  ➥« Sa fin » et «  » */.
```

substr (Index de début et de fin) :Méthode identique à la méthode slice

substring (Index de début et de fin) :Méthode identique à la précédente

```
var uneChaine = "Le début de la chaine. Sa fin.";

var sousChaine = uneChaine.substring(15,21);

// sousChaine contient la chaîne « chaine »
```

toLowerCase ( ) : Convertit la chaîne de caractères en minuscules.

toString ( ) : Retourne la chaîne de caractère interne sous forme de chaînes decaractères.

toUpperCase ( ) : Convertit la chaîne de caractères en majuscules.

```
var uneChaine = "Une chaine";

var chaineMajuscule = uneChaine.toUpperCase();

// chaineMajuscule contient « UNE CHAINE »

var chaineMinuscule = uneChaine.toLowerCase();

// chaineMajuscule contient « une chaine »
```

valueOf ( ) : Retourne la valeur primitive de l'objet. Est équivalente à la méthodetoString.

Les fonctions JavaScript :

Tout langage de programmation est étendu par la définition de fonctions. En JavaScript elles se déclarent avec le mot-clé *function*, suivi du nom, entre parenthèses une liste d'arguments et enfin le corps de la fonction entre accolades:

```
function(a, b)
{
  var x = a + b;
  return x;
}
```

Noter que les arguments ne sont pas précédés du mot-clé *var*, inutile, contrairement aux les variables déclarées dans le corps de la fonction.
Les paramètres sont passés par valeurs pour toutes les variables mais pas non lorsqu'il s'agit d'objets.
Ainsi si on change le contenu d'un objet dans le corps d'une fonction, ces modifications demeurent après l'appel de la fonction.

La fonction peut ne rien retourner, ou retourner une valeur unique. Dans le second cas on utilise le mot-clé *return*.

Une fois définie, une fonction s'appelle selon le format suivant:

```
function mult(y) { ... }

mult(23);// appel sans valeur retour
```

```
z + 3 + mult(44) // dans une expression suppose une valeur de retour
mult(b + 6)  // variable ou expression en paramètre
```

Des arguments optionnels avec des valeurs par défaut :

Les arguments d'une fonction JavaScript sont tous optionnels car l'interpréteur place les paramètres dans un tableau, et les retrouve en interne par le nom de variable qui sert d'index.

Exemple:

```
fonction maFonct(x, y, z)
```

La fonction peut être invoquée par:

```
maFonct(10);   // ou
maFonct(a, 5); // ou
maFonct(1, 2, 3);
```

En fait x, y, z sont pour l'interpréteur les clés d'un tableau associatif dont les valeurs sont les paramètres passés à l'appel de la fonction.

A l'inverse le programmeur peut ajouter lors de l'appel de la fonction, des paramètres non prévus dans sa déclaration. Par exemple:

```
maFonct(0, 10, 20, 30, 40, 50, 60);
```

Les paramètres ajoutés sont pris en compte avec le tableau des arguments, qui est un mot-clé du langage et s'appelle **arguments**. Le programmeur retrouve les paramètres par leur position dans le tableau, le premier ayant l'indice 0.

Les attributs de la variable prédéfinie argument sont:
length: le nombre d'éléments dans le tableau.
callee: le nom de la fonction dont ce sont les arguments.

Dans l'exemple d'appel de fonction avec paramètres surnuméraires précédent, cette ligne placée dans le corps de la fonction:

```
alert(arguments[3]);
```

afficherait 30.

Exemple montrant que l'on peut ajouter des paramètres lors d'un appel de fonction :

```
function myFun(x, y)
{
 var result=x;
 result += "-";
 result += y;
 for (var i = 2;  i < arguments.length; i++)
 {
  result += "-" + arguments[i];
 }
 return result;
}
function test()
{
 var x = myFun("a", "b", "c", "d", "e") ;
 alert(x);
}
```

JavaScript ne reconnaît pas les valeurs par défaut. Par exemple *function x(a = 33)* n'est pas valide dans ce langage alors que c'est reconnu par PHP ou C.

On peut utiliser différentes alternatives comme tester dans le corps de la fonction si la variable est déclarée pour lui assigner autrement une valeur. Par exemple:

```
var myFun = function(x)
{
 if (x === undefined) { x = 33; }
}
myFun();
```

Exemple de fonction déclarée dans une autre et comment on y accède :

Il est possible de définir des fonctions dans la déclaration d'une autre fonction, et les règles de visibilité en cascades s'appliquent:

```
function outer(a)
{
 function inner(b)
 {
  return b * 2;
 }
 return inner(a)
}
```

```
var z = outer(5);
alert(z);
```

Les variables de la fonction outer sont accessibles à la fonction inner. La fonction inner est alors dite une closure.

```
function outer(a)
{
 function inner(b)
 {
  return b * 2 + a;
 }
 return inner(a);
}
var z = outer(5);
document.write(z);
```

15

Cela permet en fait d'utiliser des fonctions pour créer des objets et leur ajouter des méthodes sous forme de fonctions internes

Syntaxe des expressions régulières en JavaScript et collection :

Un ensemble de règles décrivent une condition sous la forme compacte d'une expression régulière. Cela permet d'isoler un texte dans une page et éventuellement de le remplacer. Une expression régulière est définie par un objet ou un littéral. L'écriture littérale d'une expression à un format propre, elle est incluse entre deux barres inclinées:

```
var er = /xyz/
```

Tandis que l'objet est créé à partir d'une chaîne de caractères ordinaire, placée entre guillemets:

```
var er = new RegExp("xyz")
```

Lorsqu'on entre une expression régulière à partir d'un formulaire, on obtient une chaîne ordinaire, il convient alors d'utiliser l'objet pour assigner l'expression à une variable.

Construction d'une expression régulière, syntaxe et opérateurs :

La construction dépend uniquement de la connaissance des opérateurs d'expression régulière et caractères spéciaux, ainsi que des modificateurs globaux.

Les opérateurs, intervalles et groupes :

En regroupant des éléments dans une expression, on peut appliquer des opérateurs logiques. Ajoutant à cela les intervalles, il devient possible d'exprimer en peut de lettres un ensemble de règles.

Le point : .

Le point désigne tout caractère dans le texte à comparer. Sauf le code de fin de ligne.

Groupes : ()

Les parenthèses désignent un groupe de rappel, trouve l'élément entre parenthèse et le mémorise pour le restituer dans le tableau résultat ou dans les variables de l'objet RegExp.
Le masque (.) désigne un caractère quelconque. Associé à l'opérateur +, donc (.)+ Cela signifie un caractère quelconque au moins, donc un seul caractère ou une chaîne de caractères.

Par exemple (ari) permet de retrouver "ariane", ou "baril", mais "carquois" n'est pas retenu. Puis ari est mémorisé.

(?:x)

Parenthèses non capturant. On recherche l'élément x, mais il n'est pas mémorisé et n'apparaît pas dans le résultat pour la méthode qui retourne un tableau. Ni dans les variables internes.

[]

Les crochets désignent un groupe alternatif. On recherche l'un ou l'autre des éléments dans la liste.
Dans le cas ou l'on recherche [abc], alors "ariane", "baril", "corail" peuvent correspondre (si l'on teste la première lettre).

Intervalle : -

Le symbole tiret désigne entre deux lettres ou chiffres désigne un intervalle.
Exemples:
a-z liste des lettres minuscules. N'importe quelle lettre dans la liste peut correspondre.
A-Z liste des majuscules.
0-9 liste des chiffres.

Opérateurs de parties

Ces symboles servent à désigner une partie spécifique des textes à comparer avec l'expression régulière.

^

Spécifie que l'élément qui suit, caractère ou groupe, doit être placé au début du texte pour qu'il corresponde à la recherche. Si le masque est /^e/ le texte "enfin" est retenu et pas "terme".
Dans le cas d'un texte en plusieurs lignes, avec le modificateur "m" en option, cela s'applique au début de chaque ligne.

$

Spécifie que l'élément précédent, caractère ou groupe doit terminer la fin du texte. Si le masque est /e$/ le texte "enfin" n'est pas retenu, mais "ariane" le serait.
Dans le cas d'un texte en plusieurs lignes, avec le modificateur "m" en option, cela s'applique à la fin de chaque ligne.

?

L'élément précédant peut être présent ou non.
a? signifie qu'il peut y avoir une lettre a ou aucune. Cela permet de passer le caractère lorsqu'il est présent pour appliquer la suite de l'expression régulière sur la partie du texte qui vient après.

Opérateurs de quantité : +

Il doit y avoir ou moins un élément de la lettre ou du groupe précédant le symbole.

Exemples:

a+ il doit y avoir un a ou plusieurs.

[abc]+ il doit y avoir un a ou un b ou un c ou plusieurs de ces mêmes lettres (pas une combinaison car les crochets implique l'exclusion).

*

Il peut y avoir un nombre indéterminé d'occurrence du texte précédent, ou aucune.

{ n }

n représente un nombre entier quelconque. C'est le nombre d'occurrences que l'on recherche.

Exemple:

a{2} On recherche une chaîne qui contient "aa".

{ x, y }

x et y représentent deux nombres entiers positifs. Il y aura au moins x occurrences et au plus y occurrences.

Par exemple { 2, 3 } On recherche deux ou trois occurrences d'une chaîne.

Opérateurs logiques : x | y

La barre est l'opérateur OU inclusif.

Exemple: (abc | def)

On recherche la chaîne qui contient abc ou def (ou les deux).

[^]

Le symbole "^" quand il est entre crochets ne désigne pas le début d'un ensemble mais l'exclusion de cet ensemble.

Exemple:

[^xyz]

L'expression représente toutes les lettres sauf x, y ou z.

Opérateurs conditionnels : x(?=y)

Le texte correspond quand x est suivi par y.

Exemple:

moi(?=elle)

Quand moi est suivi directement par elle dans le texte, l'expression est satisfaite. Pour conserver les deux chaînes dans le tableau résultat, on écrira: **moi(?=(elle))**

Exemple:

(0-9)+(?=\.)(0-9)+

Représente un nombre décimal: suite de chiffres, point, et décimales. Cela peut s'écrire plus simplement: **\d+\.\d+**

x(?!y)

Le texte x correspond s'il n'est pas suivi par y.
Pour représenter un nombre entier on écrirait:
[0-9]+(?!\.) mais **[0-9]+** serait plus simple.

Note importante

Dans une chaîne de caractères, le code "\" doit être doublé. Par exemple on écrira **\\d** pour représenter le symbole **\d**, un digit. Ce n'est pas le cas quand on entre l'expression régulière dans un formulaire, ni dans la forme litérale:

/\d+/

Les caractères spéciaux :

Les caractères spéciaux sont introduits par le code d'échappement "\". Dans un littéral (ou un formulaire) mais dans une chaîne, le slash inversé est doublé.

Celui-ci associé à une lettre représente un code qui ne peut être affiché directement, mais il sert aussi, quand il est associé à un code opérateur, à désigner le caractère plutôt que l'opérateur d'expression régulière:

x = /a\r/
x = new RegExp("a\\r")
\n désigne la fin de ligne et non pas la lettre n.
* désigne le caractère étoile et non pas l'opérateur d'expression régulière étoile.
\t code de tabulation. \v pour une tabulation verticale.
\r code de retour à la ligne.
\f code de fin de page.
\s code de séparation quelconque, incluant:espace blanc, tabulation, retour à la ligne, fin de page.

\S tout caractère autre qu'un espace, c'est le contraire de \s.
\d tout digit, autrement dit tout caractère numérique. Equivaud à [0-9].
\D tout caractère non numérique. Equivaud à [^0-9].
\w tout caractère alphanumérique. Equivaud à [_A-Za-z0-9].
\W tout caractère autre qu'alphanumérique. C'est le contraire de \w et cela équivaud à [^_A-Za-z0-9].
\nnnn où nnnn est un nombre entier positif.
\0 Représente le code 0 dans le fichier binaire (et non le chiffre 0 dans le texte).
\xhh Où hh est un couple hexadécimal. Représente un code dans le binaire.
\uhhhh Code hexadécimal sur 4 digits.

Les modificateurs :

Ce sont des codes qui appliquent une règle générale à l'utilisation de l'expression régulière. Par exemple la lettre i signifie que l'on ne doit pas faire de différence entre majuscules et minuscules.

Les modificateurs sont les lettres i, g et m.

```
var er = /xyz/i
var er = new RegExp("xyz", "i")
```

On peut utiliser un ou plusieurs modificateurs à la fois. Par exemple:

```
var er = /xyz/igm
```

Majuscules

Le code i indique que l'on ne différencie par majuscules et minuscules dans le texte. Par exemple, si l'on applique l'expression régulière à la chaîne "untel", on aura le même résultat qu'avec la chaîne "Untel" ou "UNTEL".

Global

Le code g indique une recherche globale.

Multiple lignes

Le code m indique que l'on applique l'expression à plusieurs lignes. Les lignes sont des textes terminés par un code de fin de ligne. Dans le cas ou cette option est choisie, la comparaison est tentée pour chaque ligne.

Méthodes de RegExp et modificateur

On peut associer une méthode de l'objet RegExp à une chaîne littérale.

exec Chaîne de caractères Retourne les occurrences correspondant à l'expression régulière dans la chaîne.

test Chaîne de caractères Détermine si des occurrences sont contenues dans la chaîne de caractères en paramètre pour l'expression régulière.

```
var chaine = "Ceci est une chaine de test";

var expression = /test$/g;

var retour = expression.test(chaine);

// retour contient la valeur true

chaine = "Ceci est une chaine de test.";

retour = expression.test(chaine);

// retour contient désormais la valeur false
```

Pour exec on a :

```
var chaine = "Ceci est une chaine de test";

var expression = /test$/g;

var sousChaines = expression.exec(chaine);

// sousChaines est un tableau contenant un seul élément, « test »
```

Collection d'expressions régulières en JavaScript :

Quelques exemples d'expressions régulières d'usage courant, pour reconnaître une chaîne de caractère ou pour la modifier.
Les expressions doivent être encloses entre deux barres inclinées ou des guillemets, dans la source. Elles peuvent être testées telles qu'elles dans le testeur d'expressions régulières.

Nombre entier

-?[0-9]+

Nombre décimal

-?\d+\.\d+

Supprimer les guillemets

Cela peut être utile lorsqu'on parse le contenu d'un fichier HTML.

```
[\"\']([^\"\']*)[\"\']
var er = /[\"\']([^\"\']*)[\"\']/
var test="un texte quelconque";
document.write(test.length());
var arr = er.exec(test);
document.write(arr[1].length());
```

longueur avant: 21
longueur après: 19

Comment valider une adresse email

```
([\w-\.]+@[\w\.]+\.{1}[\w]+)
var er = /([\w-\.]+@[\w\.]+\.{1}[\w]+)/;
if (er.test(email)) document.write("valide");
```

Comment valider une URL

```
(http://|ftp://)([\w-\.)(\.)([a-zA-Z]+)
```

Exercices :

Faire une fonction de calcul d'un polynome de n'importe quelle degré

Corrections :

Polynome :

```
<html>

<head>
```

```
<title> EXOS JAVASCRIPT </title>

<script type="text/javascript">

function polynome(x)

{

var p=0;

document.write(" P(x) = ")

i=arguments.length-1;

do

{



document.writeln(arguments[i]," X<sup>",+i,"</sup>")

if (i!=1)

document.write(" + ");

i--;

}

while(i>=1);

for(var i=1; i<arguments.length;i++)

{

p+=arguments[1]*Math.pow(x,i)

}

return p;
```

```
            }


            val=prompt(" donner la valeur du polynome ")

            val=parseInt(val);


           vol=polynome(val,1,1,1,1,1)

           document.write("<br>")

           document.write(" Le polynome est  : ")

   document.write(" P(",+val," ) = ",vol);

 </script>

</head>

<body>

</body>

</html>
```

Les objets sont comme des fonctions et des tableaux :

En JavaScript, les objets sont similaires aux tableaux associatifs.
Comme les tableaux, ils sont déclarés avec un constructeur ou un littéral. Le littéral à une forme associative, les clés sont les noms d'attributs et les valeurs, leurs valeurs initiales.

```
obj1 = { 'type' : "integer", "nom" : "Objet X", taille : 11, ... }
x = obj1["nom"] // ceci assigne la chaîne 'Objet X' à l'attribut x
```

Les objets sont dynamiques, des attributs peuvent être ajoutés dynamiquement. On peut démarrer avec un objet vide:

```
obj1 = {}
```

La structure des objets est récursive, la valeur d'un attribut peut être un autre objet.

On crée une classe en déclarant une fonction-constructeur :

Les classes sont simulées en JavaScript par le moyen des constructeurs, et un constructeur est une fonction.
Le mot-clé *this* identifie les variables d'une fonction comme attributs de la classe.

```
function ClassA(x, y)
{
  this.propertyX = x;
  this.propertyY = y;
}

obj1 = new ClassA(12, 34);  // instance
obj1.propertyX = z;
obj1.properyZ = z; // ajout dynamique d'un nouvel attribut à l'instance.
```

On peut créer une classe vide et définir tous les attributs de cette façon.

Un attribut peut être effacé par la commande *delete*, ce qui complète l'aspect dynamique des objets en JavaScript.

```
delete obj1.propertyY;
```

Les méthodes sont des fonctions dans les fonctions :

On peut définir des fonctions à l'intérieur de fonctions et cela fournit des méthodes aux objets. C'est ce que l'on appelle les closures.

```
function ClassA
{
  this.methodA = function(x, y) { ...body... }
}
```

Exemple de déclaration en utilisant un littéral:

```
var ClassA = {
 nom : "class A";
 type : 3;
 changeNom : function(n)
 {
  this.nom = n;
 }
```

}

On peut simuler l'héritage

On peut hériter les membres d'une classe en déclarant une fonction qui référence les attributs de cette classe de base, ou appelle ses méthodes.

JavaScript et la programmation orientée objet :

Le langage JavaScript ne supporte pas la notion de classe afin de définir la structure des objets. Il se fonde à la place sur les principes des fonctions. Dans ce langage, les fonctions correspondent à des objets puisqu'elles y peuvent être référencées.C'est la raison pour laquelle certains auteurs préfèrent ne pas parler de classe JavaScript mais plutôt d'objet JavaScript, aussi bien pour l'entité définissant la structure d'un objet que pour ses instances. Par exemple, certains parlent de l'objet XMLHttpRequest plutôt que de la classe XMLHttpRequest.

Classes de base de JavaScrip :

Array : Représentation sous forme d'objet d'un tableau JavaScript

Boolean : Représentation sous forme d'objet du type primitif booléen

Date : Classe représentant une date

Error : Classe d'erreur générique

EvalError : Classe relative aux erreurs survenant lors de l'interprétation de code JavaScript par l'intermédiaire de la fonction eval

Function : Représentation sous forme d'objet d'une fonction JavaScript

Number : Représentation sous forme d'objet d'un nombre

Object : Classe de base de toutes les classes

RangeError : Classe relative aux erreurs survenant lors de l'utilisation de nombres excédant la plage autorisée (supérieurs à MAX_VALUE ou inférieurs à MIN_VALUE)

ReferenceError : Classe relative aux erreurs survenant lors de l'utilisation d'une référence incorrecte

RegExp : Classe permettant d'utiliser des expressions régulières

String : Class relative aux erreurs de syntaxe

TypeError : Classe relative aux erreurs de typage

URIError : Classe relative aux erreurs survenant lors d'une mauvaise utilisation des méthodes de traitements des URI de l'objet Global.

Attributs de la classe Object :

constructor : Référence la méthode utilisée pour créer une instance de la classe.

prototype : Sert à initialiser l'objet lors de son instanciation. Cet attribut est très important pour mettre en œuvre ses propres classes et l'héritage en JavaScript. Nous détaillons un peu plus loin dans ce chapitre la façon de l'utiliser.

hasOwnProperty (Nom de propriété) : Permet de déterminer si l'objet possède un attribut ou une méthode dont le nom correspond au paramètre.

isPrototypeOf (Classe) : Permet de déterminer si un objet est le prototype d'un autre.

propertyIsEnumerable (Nom de la propriété de l'objet) : Permet de déterminer si l'attribut dont le nom est passé en paramètre peut être énuméré par l'intermédiaire d'une boucle for…in.

toString(-) : Représentation de l'objet sous forme de chaîne de caractères

valueOf (-) : Valeur associée à l'objet. Cette méthode est utile dans le cas d'objets relatifs à des types primitifs. Sinon, elle renvoie la même valeur que la méthode toString.

La classe Object permet de définir facilement des instances vides, comme dans le code suivant

```
var o = new Object();
```

L'objet précédemment créé est équivalent à l'objet littéral suivant :

```
var o = {};
```

Le code suivant illustre la façon de créer un tableau avec cette classe :

```
var tableau = new Array();

tableau.push("première chaine","seconde chaîne");

for(var cpt=0;cpt<tableau.length;cpt++) {

    (...)

}
```

Ce code est similaire au code suivant, qui utilise une expression littérale afin de créer un tableau :

```
var tableau = [];

tableau[0] = "première chaine";

tableau[1] = "seconde chaîne";

for(var cpt=0;cpt<tableau.length;cpt++) {

    (...)

}
```

Le code suivant illustre la façon de créer une fonction par l'intermédiaire de cette classe :

```
var maFonction = new Function(parametre1, parametre2,

                              "return parametre1+parametre2");

//Appel de la fonction

var resultat = maFonction(13, 15);
```

L'utilisation de cette technique est équivalente au code suivant :

```
function maFonction(parametre1, parametre2) {

    return parametre1+parametre2;

}
```

//Appel de la fonction

var resultat = maFonction(13, 15);

Méthodes de la classe Function :

apply (Objet et tableau de paramètres) :

Appelle une fonction JavaScript pour l'objet spécifié en lui passant les paramètres fournis dans le tableau.

call (Objet et paramètres) : Appelle une fonction JavaScript pour l'objet spécifié en lui passant les paramètres fournis.

toSource, ou toString, () : Retournent le code source de la fonction.

Attributs de la classe Function :

arguments : Permet d'accéder aux paramètres d'invocation de la fonction.

constructor : Spécifie la fonction qui sert de base à la création de l'attribut prototype.

length : Correspond au nombre de paramètres déclarés pour la fonction.

prototype : Utilisé pour mettre en œuvre les concepts objet avec JavaScript.

Classes pré-instanciées :

JavaScript pré-instancie deux classes pour ses applications, dont les noms de leurs instances sont Global et Math.Global est utilisée implicitement par le langage, dont les méthodes « globales » ne sont apparemment rattachées à aucune instance. Cette dernière ne peut donc être utilisée explicitement.

Les méthodes de l'instance Global sont les suivantes :

• eval : afin d'interpréter une chaîne de caractères contenant du JavaScript.

• parseXXX : permet de convertir des chaînes de caractères en des types primitifs. Par exemple, pour les entiers, la méthode est parseInt.

• Méthodes de type isXXX : permettent de vérifier la véracité d'un objet pour un critère, comme, pour les nombres, les méthodes isFinite ou isNaN.

• Méthodes escape et unescape : afin de convertir des chaînes pour HTML.

Toutes ces méthodes sont rattachées à l'instance Global. Bien que cette dernière ne soit pas accessible ni utilisable directement dans les applications, ses méthodes sont utilisables telles quelles, sans avoir à être préfixées par Global.

L'instance Math offre des constantes mathématiques ainsi que des méthodes permettant deréaliser des opérations mathématiques.

Méthodes de l'instance Math :

Abs (Un nombre) : Retourne la valeur absolue d'un nombre.

acos, asin, atan (Un nombre) : Permettent de calculer respectivement les arc cosinus, sinus et tangente en radians.

Ceil (Un nombre) : Retourne l'arrondi supérieur d'un nombre.

cos, sin, tan (Un nombre) : Permettent de calculer les cosinus, sinus et tangente d'un nombre.

exp, log (Un nombre) : Permettent de calculer respectivement l'exponentiel et le logarithme d'un nombre.

Floor (Un nombre) : Retourne l'arrondi inférieur d'un nombre.

Max (Deux nombres) : Retourne la plus grande valeur des deux paramètres.

Min (Deux nombres) : Retourne la plus petite valeur des deux paramètres.

pow, sqrt (Un nombre) : Permettent de calculer respectivement le carré et la racine carrée d'un nombre.

Random (-) : Retourne un nombre pseudo-aléatoire.

Round (Un nombre) : Retourne l'arrondi d'un nombre.

Le code suivant illustre l'utilisation des trois premières :

```
var nombre = 34.567;

// Affiche 35
```

```
alert("Ceil : "+Math.ceil(nombre));

// Affiche 34

alert("Floor : "+Math.floor(nombre));

// Affiche 35

alert("Round : "+Math.round(nombre));
```

Mise en œuvre de classes :

Le mot-clé this :

Le langage JavaScript fournit le mot-clé this, qui offre une référence, dans une fonction, à l'objet sur lequel cette dernière est appelée. De ce fait, il est possible d'avoir accès à tous les attributs et méthodes de l'objet dans le code d'une fonction.

```
function test(msg) {
    return msg+" : "+this.monAttribut;
}
var premierObjet = new Object();
premierObjet.monAttribut = "mon attribut";
premierObjet.maMethode = test; ¨ 1
var retour = premierObjet.maMethode("Valeur de l'attribut"); ¨ 2
var secondObjet = new Object();
secondObjet.maMethode = test; ¨ 1
secondObjet.maMethode("Valeur de l'attribut"); ¨ 3
```

Les repères ❶ désignent l'affectation de la fonction test en tant que méthode maMethode pour les deux objets. Dans la première utilisation (repère ❷), la valeur de la variable retour est elle de l'attribut monAttribut pour l'objet premierObjet. La seconde utilisation (repère ❸) génère une erreur puisque aucun attribut monAttribut n'est défini pour le second objet.

Appels de fonctions d'un objet :

Comme nous l'avons vu précédemment, JavaScript gère les fonctions en tant qu'objets de type Function. Cette classe possède les méthodes call et apply, qui se révèlent particulièrement utiles lorsqu'elles sont utilisées conjointement avec le mot-clé this. Elles permettent alors d'exécuter une fonction pour un objet donné sans affecter la fonction à l'objet. Ces deux méthodes permettent également de passer des paramètres à ce moment et deretourner la valeur de retour de la fonction appelée. L'unique différence entre ces méthodes réside dans la façon de passer les paramètres. La première, call, prend en paramètre l'objet cible suivi de la liste des paramètres de la fonction à appeler, comme le montre le code suivant :

```
function test(msg) {
  return msg+" : "+this.monAttribut;
}
var o = new Object();
o.monAttribut = "mon attribut";
var retour = test.call(o, "Valeur de l'attribut");
```

La seconde, apply, prend en paramètre l'objet cible, suivi d'un tableau contenant la liste des paramètres de la fonction à appeler, comme dans le code suivant :

```
function test(msg) {
  return msg+" : "+this.monAttribut;
}
var o = new Object();
o.monAttribut = "mon attribut";
var parametres = ["Valeur de l'attribut"];
var retour = test.apply(o, parametres);
```

L'attribut prototype :

L'attribut prototype est un attribut particulier que possèdent toutes les classes JavaScript.Il est utilisé lors de l'instanciation des objets afin de spécifier un modèle structurel pour leur création. Les différents attributs et méthodes présents dans l'attribut sont assignés à l'objet nouvellement créé. L'attribut prototype permet ainsi de spécifier tous les attributs et méthodes communs à toutes les instances de la classe.

Cette fonctionnalité permet de faire pointer toutes les instances d'une classe sur les mêmes méthodes et évite de devoir dupliquer leurs codes lors des différentes instanciations des objets. Cela allège la mémoire des applications JavaScript fondées sur des objets. L'attribut prototype offre également la possibilité d'affecter aux attributs des objets des valeurs par défaut. Le code suivant illustre la mise en œuvre de cette technique afin d'ajouter une méthode à une classe (repère ❶) et de spécifier la valeur par défaut d'un attribut (repère ❷) pour une classe :

```
Object.prototype.maMethode = function() { ¨ ❶
    (...)
};
Object.prototype.monAttribut = "Valeur par défaut "; ¨ ❷
var o = new Object();
o.maMethode();
```

L'utilisation de prototype peut se faire à n'importe quel moment dans un script JavaScript. Tous les objets dont l'attribut prototype de la classe correspondante a été modifié sont impactés, et ce, même s'ils ont été instanciés précédemment.

Le code suivant illustre ce mécanisme, désigné sous le terme late binding :

```
var monInstance = new Object();
Object.prototype.maFonction = function() {
    return "test de late binding";
};
```

```
var retour = monInstance.maFonction();
```

Techniques de création de classes :

Une classe peut être définie en se fondant uniquement sur les fonctions, les closures et le mot-clé this. La classe est alors définie par l'intermédiaire d'une fonction dont le nom est celui de la classe. Cette fonction correspond au constructeur. Elle peut contenir des variables locales, précédées du mot-clé var, et des variables de classe, correspondant aux attributs, préfixées par this. Ces variables peuvent correspondre aussi bien à des attributs (repère ❶) qu'à des méthodes (repères ❷), comme l'illustre le code suivant :

```
function StringBuffer() {
  this.chainesInternes = []; ¨ ❶
  this.append = function(chaine) { ¨ ❷
    this.chainesInternes.push(chaine);
  };
  this.clear = function() { ¨ ❷
    this.chainesInternes.splice(0,
                this.chainesInternes.length);
  };
  this.toString = function() { ¨ ❷
    return this.chainesInternes.join("");
  };
}
```

L'utilisation de cette classe se réalise de la manière suivante :

```
var sb = new StringBuffer();
sb.append("première chaîne");
```

```
sb.append(" et ");

sb.append("deuxième chaîne ");

var resultat = sb.toString();

//resultat contient « première chaîne et deuxième chaine »
```

Le problème avec cette première approche est que le code de chaque méthode est dupliqué à chaque instanciation. Comme indiqué précédemment, l'attribut prototype permet de résoudre ce problème. La technique décrite ici ne peut cependant être utilisée que lorsque peu d'instances de la classe sont utilisées ou si des attributs ou des méthodes privées doivent être mis en œuvre.

Une bonne pratique consiste à définir les méthodes de la classe précédente à l'aide de l'attribut prototype (repères ❶) de la classe StringBuffer. Ainsi, le code de la classe est désormais le suivant :

```
function StringBuffer() {

    this.chainesInternes = [];

}

StringBuffer.prototype.append = function(chaine) { ¨ ❶

    this.chainesInternes.push(chaine);

};

StringBuffer.prototype.clear = function() { ¨ ❶

    this.chainesInternes.splice(0, this.chainesInternes.length);

};

StringBuffer.prototype.toString = function() { ¨ ❶

    return this.chainesInternes.join("");

};
```

Cette façon de définir une classe à l'aide de l'attribut prototype est celle communément utilisée. Son principal inconvénient est de ne pas définir toute la classe dans le constructeur. Il est possible de l'améliorer afin de spécifier les méthodes sur l'attribut prototype lors d'un premier appel au constructeur. Nous utilisons pour cela une propriété du constructeur (repères ❶), comme l'illustre le code suivant :

```
function StringBuffer() {
   this.chainesInternes = [];
   if( typeof StringBuffer.initialized == "undefined" ) { ¨ ❶
      StringBuffer.prototype.append = function(chaine) {
         this.chainesInternes.push(chaine);
      };
      StringBuffer.prototype.clear = function() {
         this.chainesInternes.splice(0,
                     this.chainesInternes.length);
      };
      StringBuffer.prototype.toString = function() {
         return this.chainesInternes.join("");
      };
      StringBuffer.initialized = true; ¨ ❶
   }
}
```

Objets littéraux et JSON :

Comme JavaScript considère les objets en tant que collections d'éléments contenant des valeurs, des références à d'autres objets ou des fonctions, un objet peut être défini grâce à

une structure JSON. Cette dernière correspond à la représentation littérale de l'objet. JSON définit deux structures de données différentes. La première correspond à la définition d'un objet par l'intermédiaire d'une liste non ordonnée de clés et de valeurs, dont le format est le suivant :

```
{
  cle1: valeur,
  cle2: function() {
    (...)
  }
}
```

La seconde correspond à la définition d'un tableau simple par l'intermédiaire d'une liste non ordonnée de valeurs, dont le format est le suivant :

```
[
  valeur,
  function() {
    (...)
  }
]
```

Ainsi, la classe StringBuffer de la section précédente pourrait être écrite de la manière suivante :

```
function StringBuffer() {
  this.chainesInternes = [];
}
StringBuffer.prototype = {
```

```
    append : function(chaine) {
        this.chainesInternes.push(chaine);
    },
    clear : function() {
        this.chainesInternes.splice(0,
                    this.chainesInternes.length);
    },
    toString : function() {
        return this.chainesInternes.join("");
    }
};
```

L'héritage de classes :

Le code suivant en donne un exemple de mise en œuvre :

```
//Définition de la classe mère
function StringBuffer() {
    this.chainesInternes = [];
    //Spécification des méthodes de la classe mère
    if( typeof StringBuffer.initialized == "undefined" ) {
        StringBuffer.prototype.append = function(chaine) {
            this.chainesInternes.push(chaine);
        };
        StringBuffer.prototype.clear = function() {
```

```
            this.chainesInternes.splice(0,
                        this.chainesInternes.length);
        };
        StringBuffer.prototype.toString = function() {
            return this.chainesInternes.join("");
        };
        StringBuffer.initialized = true;
    }
}
//Définition de la classe fille
function ExtendedStringBuffer() {
    //Appel du constructeur de la classe mère
    StringBuffer.call(this);
    //Spécification des méthodes de la classe fille
    if( typeof ExtendedStringBuffer.initialized == "undefined" ) { ¨ ❷
        //Copie des méthodes de la classe mère
        for( var element in StringBuffer.prototype ) { ¨ ❶
            ExtendedStringBuffer.prototype[element]
                        = StringBuffer.prototype[element];
        }
        ExtendedStringBuffer.prototype.append = function(chaine) {
            this.chainesInternes.push(chaine);
```

```
        };
        ExtendedStringBuffer.initialized = true; ¨ 2
    }
}
```

La copie des éléments de l'attribut prototype de la classe mère dans la classe fille se réalise désormais dans le constructeur de cette dernière (repère ❶). Elle n'est réalisée qu'une seule fois grâce à la propriété initialized rattachée au constructeur de la classe ExtendedStringBuffer (repères ❷).

L'héritage multiple :

Le code suivant illustre la mise en œuvre de l'héritage multiple pour la classe ExtendedStringBuffer, dont les classes mères sont StringBuffer et StringSpliter :

```
function StringBuffer() { ¨ 1
    this.chainesInternes = [];
    //Spécification des méthodes de la première classe mère
    if( typeof StringBuffer.initialized == "undefined" ) {
        StringBuffer.prototype.append = function(chaine) {
            this.chainesInternes.push(chaine);
        };
        StringBuffer.prototype.clear = function() {
            this.chainesInternes.splice(0,
                        this.chainesInternes.length);
        };
        StringBuffer.prototype.toString = function() {
```

```
            return this.chainesInternes.join("");
        };
        StringBuffer.initialized = true;
    }
}
function StringSpliter() { ¨ 2
    if( typeof StringSpliter.initialized == "undefined" ) {
        StringSpliter.prototype.split = function(chaine) {
            this.chainesInternes = chaine.splite(" ");
        };
        StringSpliter.initialized = true;
    }
}
function ExtendedStringBuffer() { ¨ 3
    StringBuffer.call(this); ¨ 4
    StringSpliter.call(this); ¨ 4
    if( typeof ExtendedStringBuffer.initialized == "undefined" ) { ¨ 4
        //Copie des méthodes de la première classe mère
        for( var element in StringBuffer.prototype ) {
            ExtendedStringBuffer.prototype[element]
                            = StringBuffer.prototype[element];
        }
```

```
        //Copie des méthodes de la seconde classe mère
        for( var element in StringSpliter.prototype ) {
            ExtendedStringBuffer.prototype[element]
                        = StringSpliter.prototype[element];
        }
        ExtendedStringBuffer.prototype.append = function(chaine) {
            this.chainesInternes.push(chaine);
        };
    }
}
```

Les repères ❶, ❷ et ❸ pointent respectivement sur les définitions des classes mères StringBuffer et StringSpliter et de la classe fille ExtendedStringBuffer. Les repères ❹ illustrent l'appel aux constructeurs des classes mères ainsi que l'initialisation de l'attribut prototype de la classe fille à partir de ceux des classes mères.

Exercice :

Creation de la classe coordonnee avec ses fonctions membres : initialisation, addition, soustraction, norme et affichage.

Correction :

```
<html>
<head>
<title> EXOS JAVASCRIPT </title>
<script type="text/javascript">
 function coordonnee (){
```

```
  var x,y;
                    }
coordonnee.prototype.initialisation= function(a,b)
  {  this.x=a; this.y=b; }
coordonnee.prototype.addition= function(a)
  { var p=new coordonnee();
      p.x=a.x + this.x;  p.y=a.y + this.y;
  return p;
  }
   coordonnee.prototype.soustraction= function(a)
  { var p= new coordonnee();
      p.x=a.x-this.x;  p.y=a.y-this.y;
  return p;
  }
 coordonnee.prototype.norme= function()
  { var p= new coordonnee();
       p=Math.sqrt( Math.pow(this.x,2)+Math.pow(this.y,2));

  return p;
  }
   coordonnee.prototype.affichage= function()
  {
```

```
      document.write(" ( ",+this.x," , ",+this.y," ) <br>");
    }
var obj1= new coordonnee();
 q=prompt(",donner la coordonnee x ");
q=parseInt(q);
s=prompt(",donner la coordonnee y ");
s=parseInt(s);
obj1.initialisation(q,s);
var obj2= new coordonnee();
 q=prompt(",donner la coordonnee x ");
q=parseInt(q);
s=prompt(",donner la coordonnee y ");
s=parseInt(s);
obj2.initialisation(q,s);
var obj3= new coordonnee();
document.write(" le premier point est ");
obj1.affichage();
document.write(" le second point est ");
obj2.affichage();
document.write(" la somme des points donne : ");
obj3=obj1.addition(obj2);
obj3.affichage();
```

```
document.write(" la soustraction des points donne : ");

obj3=obj1.soustraction(obj2);

obj3.affichage();

document.write(" le point "); obj1.affichage();

document.write(" a pour norme <br>",obj1.norme());

document.write("<br> le point "); obj2.affichage();

document.write(" a pour norme <br>",obj2.norme());


 </script>

</head>

<body>

</body>

</html>
```

Ajout dynamique de code JavaScript avec eval :

JavaScript dispose d'une fonction pour créer du code dynamiquement et ce code peut contenir des variables, des objets et des fonctions.
Il est placé dans une chaîne de caractères qui est donnée en argument à la fonction intégrée *eval*.

Eval peut s'utiliser avec JSON. Ce format est une alternative à XML plus simple dans sa définition et dans son utilisation. En effet une déclaration JSON peut devenir directement un objet JavaScript, dès lors qu'elle est donnée en argument à la fonction *eval*.

On peut utiliser eval *pour ajouter du code au script en cours :*

L'exemple le plus simple est l'évaluation d'une expression :

```
var x = 10;
alert( eval('(x + 2) * 5'));
```

Affiche 60.

Mais la force d'*eval* est dans la génération de code, par exemple ajouter un assignement...

```
var strvar = "var z = 5;";

function evalvar()
{
  eval(strvar);
  alert(z * 3);
}
```

Afficher 15.

On peut même ajouter une fonction dynamiquement pour compléter le script en cours :

On crée dynamiquement la fonction *myfun*() avec eval, et la fonction de test usefun() appelera cette fonction et affichera la valeur qu'elle retourne en fonction de l'argument qu'on lui passe.

```
var strfun = "function myfun(arg) { return arg * 3; }";

eval(strfun);

function usefun(x)
{
  var res = myfun(x);
  alert(res);
}
```

Un autre exemple avec des variables et une fonction:

```
<script>
var doc = "var x=5;var y=6;function mult(a){return a * y;}";
eval(doc);
document.write("x=" + x + "<br>");
document.write("y=" + y + "<br>");
```

```
document.write("mult(x)=" + mult(x) + "<br>");
</script>
```

```
x=5
y=6
mult(x)=30
```

Ces possibilités sont bien sûr surtout intéressantes si le code ajouté ainsi est généré par programme en fonction de calculs et de nouveaux paramètres dans un traitement.

La fonction **eval** n'est pas le seul moyen d'augmenter le code du script en cours, on peut aussi générer un script et l'attacher au document avec les méthodes du DOM et l'exécuter durant le traitement.

eval *à une visibilité globale ou locale :*

Utilisée dans un contexte local, la fonction *eval* va créer des objets dont la portée est limitée au contexte. Pour leur donner une portée globale, il faut associer *eval* à la fenêtre:

```
window.eval(...)
```

ou avec une variable:

```
var gvar = this;
gvar.eval(...);
```

Les variables déclarées dans l'argument de eval seront alors globales.

Sécurité: Attention au code injecté :

La fonction *eval* pourrait être utilisée pour exécuter du code malicieux sur les navigateurs des internautes. On doit donc prendre garde à vérifier les utilisations possible du code qui peut être généré dynamiquement.

Pour éliminer tout code indésirable, la RFC 4627 propose une expression régulière, cependant elle n'empêche pas de modifier les variables globales ou de session.

Mais en fait, le problème de sécurité ne se pose que si le contenu vient d'une source externe, par exemple d'un utilisateur par un champ d'entrée de texte

Les fonctions prédéfinies dans JavaScript, interface de test :

JavaScript dispose de fonctions accessibles dans tout script. Vous pouvez utiliser cette interface pour les tester sur des navigateurs différents.

Les fonctions de test vérifient le type ou le contenu d'une variable.

isNaN

Teste si un objet est un nombre ou non.

```
if(isNaN(x))
{
  alert(x + " n'est pas un nombre.");
}
```

isFinite

Teste une expression et retourne vrai si le résultat c'est un nombre fini.

```
if(isFinite(x + 5 * y))
{
  alert("Le résultat est un nombre fini.");
}
```

Si la valeur assignée est NaN, le résultat retourné par isFinite sera *false*.

Les fonctions de codage transforment une chaîne de caractère en code UTF-8 qui puisse être utilisé dans une URL, donc dans un nom de fichier et ses paramètres. Ou l'inverse.

encodeURI

Transforme une chaîne de caractère en UTF-8 en remplaçant certains caractères par des codes.

Les codes suivants ne sont pas remplacés:

```
/ ? : @ & = + $ , - _ . ! ~ * ' ( ) # - _ . ! ~ * ' ( )
```

Pas plus que les lettres et les chiffres.
L'espace blanc est remplacé par %20.

encodeURIComponent

La différence avec *encodeURI*, est que certains caractères qui ont une utilité pour les méthodes GET et POST, sont aussi codés.

On a un chemin et un nom de fichier valide sous un système d'exploitation, contenant éventuellement des espaces, des virgules, des & et des +.
Or ces codes ont un rôle de séparateur dans une URL.
Pour créer une URL contenant le nom de fichier, ces caractères doivent être convertis:

/ ? : @ & = + $, #

decodeURI

C'est la fonction inverse à *encodeURI*. Elle recrée un texte ordinaire à partir d'une chaîne UTF-8.

decodeURIComponent

Fonction inverse à *encodeURIcomponent*, elle substitue des caractères normaux aux codes remplacés y compris les caractères spéciaux désignés ci-dessus.

Les fonctions de transformation en nombre ont pour argument une chaîne dont le contenu correspond à un nombre, ou une variable à laquelle est assignée une telle chaîne ou un nombre.

parseInt

Convertit une chaîne en nombre entier. Il est possible de choisir la base que l'on passe en second paramètre optionnel: 10 (décimal), 8 (octal), 16 (hexadécimal).
Par défaut la base 10 est appliquée.

Exemple:

```
var x = parseInt("123");
```

affiche 123, mais:

```
var x = parseInt("123" , 16)
```

affichera 291.

On peut spécifier un nombre hexadécimal avec le préfixe 0x et octal avec 0.

```
var x = parseInt("0xFFFF", 16);
```

Sans la base 16, cela retourne 0.

parseFloat

Convertit une chaîne contenant une valeur numérique en nombre réel. Si la chaîne contient un nombre avec des décimales, la fonction *parseInt* ne conserve que la partie entière, contrairement à *parseFloat*.

Les valeurs admises peuvent avoir la forme 1.2 ou 0.5 ou 1e-2 ou 0.1E+2.

Les objets dynamiques de JavaScript :

JavaScript a modernisé les langages à objets avec la déclaration dynamique. Il n'y a pas de classes comme en langage C, où l'on définit d'abord les méthodes et attributs, et qui servent ensuite de modèles aux instances.

Un objet JavaScript est défini sous la forme syntaxique d'une fonction, qui correspond à une première instance et que l'on clone pour créer d'autres instances.
En outre cette structure est dynamique, on peut lui ajouter durant l'exécution des méthodes (en fait des fonctions internes) et des attributs.
La principale question est de savoir si, une méthode ou un attribut étant ajoutés dynamiquement à un objet, ils deviennent aussi disponibles pour les objets dérivés, ce que l'on va voir.

Un objet est défini comme une fonction :

L'objet se crée en définissant un constructeur, on peut l'assigner à un identifieur pour avoir une instance.

```
function voiture(vitesse)
{

  var passagers = 4;

  this.vitesse = vitesse;
}
```

Le constructeur peut avoir des arguments, et il contient des propriétés.
On assigne le constructeur avec le mot-clé **new**:

```
mavoiture = new voiture(120);
```

Il est possible de créer un objet par assignation d'un littéral constitué d'une liste de propriétés/valeurs séparées par une virgule:

```
mavoiture = { vitesse:120, passagers:4 };
```

Exemple:

```
function voiture(v)
{
 this.vitesse = v;
}
var mavoiture=new voiture(120)
document.write(mavoiture.vitesse);
```

120

L'instance peut être déclarée directement lors de la définition de l'objet à condition d'assigner une valeur aux arguments si on veut y accéder après. Exemple:

```
var mavoiture = new voiture(v = 200)
{
 this.vitesse = v;
}
document.write(mavoiture.vitesse);
```

200

Les propriétés d'un objet sont ajoutées en cours de traitement :

Un objet peut se définir comme une liste de propriétés, équivalent aux attributs d'une classe.

On peut ajouter une propriété directement à un objet déjà défini.

```
nomobjet.nompropriété = xxxx
```

On accède à la propriété selon la même syntaxe. Exemple pour l'objet voiture dont la vitesse est une propriété:

```
alert(voiture.vitesse);
```

Le prototype peut être modifié après héritages en modifiant une instance :

Pour ajouter dynamiquement une propriété et faire qu'elle soit utilisable par tous les clones dérivés du même objet, même s'ils sont créés avant que la propriété ne soit ajoutée, on utilise le mot réservé *prototype*.

```
voiture.prototype.consommation = 10;
```

La propriété *consommation* appartient maintenant à *mavoiture* et autres objets éventuellement définis selon *voiture*.

Exemple d'héritage en JavaScript et modification rétroactive du prototype (avec toutes ses instances):

```
var mavoiture= new voiture(v = 200)
{
 this.vitesse = v;
}
camion = mavoiture;
document.write(camion.vitesse);

voiture.prototype.consommation=50;
document.write("consommation=" + camion.consommation);
```

200
consommation=50

Un objet peut contenir d'autres objets assignés comme valeurs d'attributs :

Une propriété peut être un autre objet. Dans ce cas on accède aux propriétés de l'objet contenu par une chaîne d'identificateurs séparés par un point.
Par exemple, si l'objet *mavoiture* contient un objet *untel* qui a pour constructeur *conducteur* avec la propriété *age*, on accède à *age* ainsi:

```
function conducteur(age)
{
  this.age = age;
}
untel = new conducteur(25);

mavoiture.conducteur = untel;
document.write(mavoiture.conducteur.age);
```

On peut aussi intégrer un objet dans un autre sous la forme d'un littéral:

```
mavoiture = {
  vitesse:120,
  passagers:4,
  untel : {
    age:25
  }
};
```

L'objet mavoiture contient l'objet untel défini lui aussi par un litéral. Notez bien les virgules qui séparent les éléments, au lieu de point-virgules.

On accède aux valeurs des attributs comme dans un tableau :

On accède aux valeurs des propriétés soit par leur nom, comme on l'a fait précédemment, soit par leur numéro d'ordre dans la liste des propriétés de l'objet, comme avec les éléments d'un tableau, à condition qu'ils correspondant à des objets du document HTML.

Si l'objet ne fait pas partie du document HTML, on accède au contenu par le nom des propriétés ou par un indice si on a ajouté des valeurs selon un indice.

Par exemple, si l'on assigne une valeur par un indice:

```
voiture[4] = "demo";
```

on accèdera à la valeur de la même façon:

```
alert(voiture[4]);
```

Un objet correspond ainsi à un tableau ambivalent (qui offre des aspects différents), avec des éléments ordinaux ou des propriétés, contrairement au langage PHP qui utilise les nombres à la fois comme clés et comme ordinaux, ce qui rend le contenu des tableaux indéterminable.

Exemples d'utilisation d'objets :

```
function voiture(vitesse)
{
  var passagers = 4;
  this.vitesse = vitesse;
}
mavoiture = new voiture(120);
```

```
document.write("vitesse: ");
document.write(mavoiture.vitesse);
```
vitesse: 120

```
function conducteur(age)
{
  this.age = age;
}
untel = new conducteur(25);
mavoiture.conducteur = untel;
document.write("age conducteur: ");
document.write(mavoiture.conducteur.age)
```
age conducteur: 25

```
mavoiture = { vitesse:120, passagers:4, untel : { age:25 } };
document.write("assignement d'un objet litéral: ");
document.write(mavoiture.untel.age);
```
assignement d'un objet litéral: 25

```
voiture[1] = "demo";
document.write("indice: ");
document.write(voiture[1]);
```
demo

L'objet Number :

L'objet *Number*, depuis notamment la version 1.5 de JavaScript, dispose de méthodes de conversion de nombres.

Utilisation de Number

Un objet *Number* se crée par l'appel du constructeur avec une valeur numérique en argument

```
var n = new Number(valeur numérique);
```

Les chaînes de caractères sont des arguments valides si elles contiennent une valeur numérique.

```
var n = new Number("10000");
```

De même les nombres hexadécimaux sous la forme 0xNNNN.

```
var n = new Number(0xFF);
```

Le format numérique de HTML n'est pas reconnu.

```
var n = new Number(#FF);   // non valide
```

Si l'argument n'est pas un nombre ou n'est pas directement convertible en nombre, l'objet vaudra *NaN*, ce qui signifie: Not A Number.

Attributs de l'objet

L'objet *Number* à des attributs utiles pour le test de validité du nombre contenu:

MAX_VALUE: valeur maximale de l'argument.

MIN_VALUE: valeur minimale.

NEGATIVE_INFINITY: dépassement de capacité obtenu avec une valeur négative trop grande.

POSITIVE_INFINITY: dépassement de capacité obtenu avec une valeur positive trop grande.

NaN: valeur de l'objet lorsqu'on passe un argument non valide.

On n'utilise pas directement *Number.NaN* mais plutôt la fonction *isNaN*.

```
if(isNaN(new Number(x)) alert("Pas un nombre");
```

Méthodes de *Number*

Les méthodes essentielles sont des fonctions de conversion. Les méthodes peuvent s'appliquer autant à un nombre littéral qu'à l'objet:

```
alert(new Number(123).toString());
alert(123.toString());

var x = new Number(154.39);
alert(x.toFixed());
```

Méthodes:

toString: retourne l'objet sous forme de chaîne. Ainsi 123 deviendra "123". Si l'argument est une chaîne, cette méthode retourne la chaîne originelle.

toFixed: retourne le nombre en virgule fixe. Par défaut, retourne la partie entière.

```
var x = new Number(123.54)
x.toFixed() devient 123
x.toFixed(1) devient 123.5
```

toExponential: retourne le nombre sous notation exponentielle.

toPrecision: affiche le nombre avec le nombre de décimales donnée en argument à toPrecision. S'il n'y a par d'argument, le nombre est inchangé.

Number hérite aussi des méthodes d'*Object*.

Exercices : résolution de l'équation du second degrés avec une bonne présentations.

L'objet String. Essayer interactivement ses méthodes :

L'objet String permet d'associer des méthodes aux chaînes de caractères. Il y a des différences entre une variable qui est une instance de String, et une variable à laquelle une chaîne est assignée directement.
Invoquer le constructeur est aussi un moyen de convertir des valeurs en objets chaînes.

Le constructeur String sert aussi à convertir en chaîne :

Le constructeur admet un argument unique qui est une chaîne littérale ou un objet quelconque que l'on veut convertir en chaîne.

La syntaxe est:

```
var x = new String("chaine");
```

Exemple

```
var x = new String("demo");
```

Une variable peut être l'argument du constructeur, comme pour tout objet:

```
var y = "demo";
var x = new String(y);
document.write(x);
```

L'objet n'est pas interprété par la fonction *eval*.

```
var x = new String("5 + 5");
var y = eval(x);
document.write(y);
```

5 + 5

Ce n'est pas le cas d'une simple variable:

```
var x = "5 + 5";
document.write(x);
var y = eval(x);
document.write(y);
```

5 + 5

10

L'objet String a un attribut de longueur :

Length

Nombre de caractères dans la chaîne.

Les codes spéciaux HTML ne sont pas interprétés:

```
var x = new String(" ");
document.write(x.length);
```

6

A l'attribut *length* s'ajoutent les attributs hérités de Object.

On accède aux caractères par un indice :

Comme pour un tableau, on peut indicer une chaîne pour obtenir le caractère à la position donnée:

```
var x = new String("démo");
document.write(x[2])   // doit retourner m en position 2 à partir de zéro
```

m

On obtient le même résultat avec la méthode *charAt*

Description des méthodes

x = y.anchor(nom)

Crée une ancre à partir de la chaîne et d'un nom que l'on donne en paramètre.

x = y.charAt(position)

Retourne le caractère à la position donnée en argument à la méthode, en partant de zéro.

La méthode *charCodeAt* retourne la valeur Unicode du caractère.

x = y.concat(chaîne1 [, chaîne2, ...)

Combine la chaîne avec une ou plusieurs autres et retourne la nouvelle chaîne.

y.fontcolor(couleur)

Affiche la chaîne dans la couleur donnée en argument sous forme numérique comme #006600 ou textuelle comme "blue".

Les méthodes *fontsize, bold, italic, strike, sub, sup, big, small, blink* sont d'autres fonctions de présentation de String.

Voir les propriétés CSS correspondantes. Le nom est le même sauf *font-size*.

x = y.indexOf(chaîne [, position])

Retourne la position dans la chaîne originelle, d'une autre chaîne donnée en paramètre. La fonction est sensible à la casse.

Retourne -1 si la chaîne n'est pas trouvée. La position où commence la recherche est optionnelle.

Le méthode *lastIndexOf* retourne la position à partir de la droite de la première occurence de la chaîne en argument.

x = y.link(URL)

Crée un lien hypertexte avec la chaîne comme ancre et l'URL donnée en paramètre.

x = y.match(reg exp)

Retourne le résultat d'une expression régulière appliquée à la chaîne. La méthode retourne un tableau s'il y a plusieurs occurrences trouvées dans la chaîne.

x = y.replace(reg exp, chaîne ou fonction)

Remplace les occurrences de l'expression régulière donnée en premier paramètre, par la chaîne qui est le second paramètre, puis retourne le résultat sans changer l'objet originel. Le premier argument peut être plus simplement une chaîne.

Si le second argument est une fonction, la valeur de retour de la fonction remplace les occurrences trouvées.

x = y.search(reg exp)

Cherche l'expression régulière (ou une simple chaîne) et retourne la position.

Retourne -1 si elle n'est pas trouvée.

x = y.slice(début [, fin])

Extrait une partie d'une chaîne et retourne le résultat sans modifier la chaîne originelle. La partie à extraire est définie par les paramètres début et fin, la position de fin non incluse. Si fin est omis, c'est la fin de la chaîne qui est utilisée comme second paramètre.
Si on donne en paramètre: 0 et length, cela retourne donc la même chaîne.
Le second argument peut être négatif et dans ce cas, il est soustrait de la longueur de la chaîne.

x = y.split(séparateur [, limite])

Divise une chaîne en sous-chaînes qui sont assignées à un tableau. Le séparateur est l'élément utilisé pour reconnaître les parties, généralement un espace blanc ou une virgule, il ne sera pas intégré au tableau. L'argument optionnel "limite" indique le nombre maximal d'éléments à placer dans le tableau.

x = y.substr(début, longueur)
x = y.substring(début, fin)

Ces deux fonctions extraient une sous-chaîne d'une chaîne. *Substring* a les mêmes paramètres que *slice*. *Substr* correspond à la fonction du langage C, on donne en paramètres la position de la sous-chaîne et le nombre de caractères. Une sous-chaîne est retournée sans que la chaîne originale ne soit changée.

toLowerCase et toUpperCase

Convertissent la chaîne respectivement en minuscules et majuscules. Ces méthodes n'ont aucun paramètre et ne modifient pas l'objet.

Objet Date en JavaScript, accéder aux mois, jours... :

Une date est un objet dont la base est un nombre de milliseconde et qui s'affiche dans plusieurs formats par conversion des millisecondes en année, mois, jours...

La version JavaScript 1.3 à uniformisé et étendu les possibilités de l'objet. Elle est reconnue par tous les navigateurs mêmes anciens.

Un objet Date se crée par l'appel d'un constructeur dont le format est le suivant:

```
new Date()
new Date(millisecondes)

new Date(année, mois, jour [, heure, minutes, secondes, millisecondes])  // les arguments sont des nombres entiers
new Date(texte)  // une chaîne représentant une date
```

S'il n'y a pas d'argument, l'objet correspond au jour et à l'heure lors de la création de l'objet.

Dans le cas ou l'argument est une date sous forme de texte, son format devra être reconnu par la méthode *Date.parse*.
Par exemple:

Apr 16, 2008

La méthode *parse* reconnaîtra aussi le format IETF, comme:

Mon, 16 Apr 2008 12:20:05 GMT

Le nombre de milliseconde est relatif au 1 janvier 1970. On utilise les millisecondes pour les durées, mais il est préférable d'utiliser des dates autrement.

Les formats UTC et GMT sont reconnus :

GMT signifie *Greenwich Mean Time*, *temps moyen de Greenwich*. C'est le temps solaire mesuré à l'observatoire de Greenwich en Angleterre.

UTC signifie *Coordinated universal time*, soit *temps universel coordonné.* C'est l'unité standard internationale qui succède à GMT, et qui est basée sur le temps atomique plutôt que sur la rotation de la terre. Il est défini par la norme ISO 8601

Calcul de la différence entre deux dates :

L'objet Date supporte les opérations arithmétiques. Dans la pratique, on utilise surtout la soustraction pour connaître la durée qui sépare deux dates.

```
<script>
var debut = new Date(1000, 01, 01);
var fin = new Date(2011, 10, 15);
var diff = new Number((fin.getTime() - debut.getTime()) / 31536000000).toFixed(0);
document.write(diff);
</script>
```

1012

Formule simple pour connaître la date du jour :

Elle s'obtient par la création d'un objet Date sans argument. Par exemple avec le script suivant

```
<script type="text/javascript">
        document.write(new Date());
</script>
```

Ce qui affiche:

Sun Jan 08 2012 22:54:52 GMT+0000 (Maroc)

Et le jour de la semaine :

On affiche le jour de la semaine avec un tableau et noms et la méthode *getDay*() qui retourne le numéro à partir de 0.

```
<script type="text/javascript">
 var j = new Array( "Lundi", "Mardi", "Mercredi", "Jeudi", "Vendredi", "Samedi", "Dimanche" );
 var d = new Date();
 document.write(j[d.getDay()]);
</script>
```

Mercredi

Les equivalents anglais de lundi à dimanche sont: Monday, Tuesday, Wednesday, Thursday, Friday, Sunday, Saturday.
Les raccourcis sont: Mon, Tue, Wed, Thu, Fri, Sun, Sat.

Descriptions des méthodes

Lecture

getDay: Jour de la semaine (0-6) en commençant dimanche.

getFullYear: Année sur 4 chiffres.

getMonth: Mois en chiffres, de 0 à 11, zéro pour janvier.

getDate: Jour du mois (1-31).

getHour: Heure.

getMinutes: Minutes.

getSeconds: Secondes dans la minute de la date donnée.

getMilliSeconds: Nombre de millisecondes, de 0 à 999 dans la seconde de la date donnée.

getTime: Date en millisecondes depuis le 1 janvier 1970.

toString: Retourne la date sous forme de chaîne de caractère.

toDateString: Retourne la partie date avec le jour de la semaine (anglais).

toLocaleDateString: La partie date localisée, en français ou autre langue.

toGMTString ( ) : Retourne la date GMT sous forme de chaînes de caractères.

Assignement

setFullYear, setMounth, setDate, setHours, setMinutes, setSeconds, setMilliseconds.

Assigne les données à un objet Date.

setTime: Assigne un nombre de millisecondes depuis le 1 janvier 1970.

Exercices : faites une page qui demand votre date de naissance et vous signal le jour de votre naissance ainsi que le jour de votre prochain anniversaire.

Correction :

```
<html>

<head>

<title> EXOS JAVASCRIPT </title>

<script type="text/javascript">

 var annee=prompt(" donner votre annee de naissance ");

 var mois=prompt(" donner votre mois de naissance ")

 var jour=prompt(" donner le jour entre 1 et 31 de votre naissance")

 var date=new Date(annee,mois-1,jour);

   var jour_semaine=date.getDay();
```

```
var j= new Array (  "Dimanche","Lundi", "Mardi", "Mercredi", "Jeudi", "Vendredi", "Samedi");

  document.write(" votre jour de naissance est un  : ",j[jour_semaine])

  nextdate= new Date();

  nextdate.setMonth(mois-1)

  nextdate.setDate(jour);

  document.write(" <br> <br> <br> le jour de votre prochain aniversaire es  :
",j[nextdate.getDay()])

 document.write(" <br> qui en format GMT donne : ",nextdate.toGMTString())



 </script>

</head>

<body>

</body>

</html>
```

Gestion des exceptions :

Comme la plupart des langages objet, JavaScript permet de gérer les exceptions. Ce mécanisme offre la possibilité d'intercepter les erreurs au lieu de les laisser traiter par l'intercepteur du langage. Cela évite l'arrêt brutal de l'exécution d'un script et, en conjonction avec un gestionnaire d'exceptions robuste, facilite le débogage des applications.Le langage offre également la possibilité de lancer des exceptions applicatives.

Interception globale :

JavaScript offre la possibilité d'intercepter d'une manière globale ou pour un bloc HTML particulier toutes les erreurs qui surviennent. Lorsqu'une erreur se produit, l'interpréteur passe la main à l'observateur de l'événement s'il a été spécifié.

Attraper les exceptions :

JavaScript offre la même syntaxe que Java pour gérer les exceptions, au moyen des mots clés try, catch et finally. Le premier définit le bloc d'interception des exceptions, le second les traitements à réaliser en cas de levée d'exceptions et le dernier les traitements à exécuter, que des exceptions soient levées ou non. Le code suivant décrit la façon de gérer les exceptions (nous faisons volontairement appel à une méthode testException inexistante afin de déclencher une exception) :

```
try {
   testException(); //Cette méthode est inexistante!
} catch(error) {
   alert("Une exception a été levée");
   alert("Nom de l'exception levée : "+error.name);
   alert("Message de l'exception levée : "+error.message);
} finally {
   alert("Passage dans finally");
}
```

À l'exécution de ce code, la valeur de la propriété name de l'exception est ReferenceError, et la valeur de la propriété message est « testException is not defined ». Notons que le code défini dans le bloc finally est bien appelé.

Lancer des exceptions :

JavaScript offre la possibilité aux applications de lancer des exceptions afin de signaler une erreur. Cette fonctionnalité se réalise par l'intermédiaire du mot-clé throw de la même manière que pour le langage Java. Le code suivant en donne un exemple d'utilisation :

```
try {
   throw new Error("test");
} catch(error) {
   alert("Une exception a été levée");
   alert("Nom de l'exception levée : "+error.name);
   alert("Message de l'exception levée : "+error.message);
}
```

Le langage offre la possibilité de dé�nir ses propres classes d'exception. Ces dernières n'ont pas besoin d'hériter de la classe Error car le mot-clé throw permet de déclencher n'importe quelle classe JavaScript. Le code suivant illustre l'implémentation d'une exception utilisateur ainsi que son utilisation :

```
function MonException(message) {
   this.name = "MonException";
   this.message = message;
}
try {
   throw new MonException("test");
} catch(error) {
   alert("Une exception a été levée");
   alert("Nom de l'exception levée : "+error.name);
   alert("Message de l'exception levée : "+error.message);
}
```

Le DOM :

Tout langage de balises possède une structure logique arborescente dont la représentation objet correspond au DOM. Cette dernière fournit des méthodes permettant de parcourir cet arbre et éventuellement de le modifier.

Manipulation des éléments :

Pour accéder au nœud d'un document afin de pouvoir le manipuler par la suite, une méthode simple consiste à identifier ce nœud par un attribut id. La méthode getElementById de l'objet document peut ensuite se fonder sur ce dernier pour récupérer l'instance correspondante.

La méthode getElementsByTagName, permet de récupérer une liste de nœuds en se fondant sur le nom des balises, lequel correspond à la valeur de la propriété nodeName de la classe Node décrite précédemment.

La méthode getElementsByName retourne une liste de nœuds dont l'attribut name est égal à la valeur spécifiée en paramètre.

Exemple :

```
<form method="post" id="monFormulaire">
    <input type="radio" name="couleur" value="rouge"/>Rouge<br/> ¨ ❶
    <input type="radio" name="couleur" value="bleu"/>Bleu<br/> ¨ ❶
    <input type="radio" name="couleur" value="jaune"/>Jaune<br/> ¨ ❶
</form>
```

Le code JavaScript suivant permet de référencer directement les trois balises input (repères ❶ dans le code précédent) avec le support de cette méthode :

```
var elementsInput = document.getElementsByName("couleur");

var elements = document.getElementsBytagName("form"); // liste des balises form

var zone = document.getElementById("monFormulaire"); // spécifie l'unique balise dont
id= « monFormulaire»
```

Le code suivant fournit un exemple de création d'une balise, d'un nœud de type texte et d'un commentaire HTML :

```
var element = document.createElement("label");

var elementTexte = document.createTextNode("mon texte");

var commentaire = document.createComment("mon commentaire");
```

Une fois le nœud créé, il ne reste plus qu'à l'attacher à un nœud existant en utilisant les méthodes appendChild ou insertBefore, qui ajoutent un nœud enfant sous une balise. Le code suivant se fonde sur notre code HTML exemple pour ajouter dynamiquement une balise label une fois la page chargée :

```
var zone = document.getElementById("maZone");

var label = document.createElement("label");

var texte = document.createTextNode("mon label");

label.appendChild(texte);

zone.appendChild(label);
```

Ce code permet de modier l'arbre DOM en mémoire de la page an d'obtenir le résultat suivant :

```
<div id="maZone">
   <span><b>Un texte</b></span>
   <div id="monAutreZone">
      Un autre texte
   </div>
   <label>mon label</label>
</div>
```

Supposons que nous souhaitions ajouter sept paragraphes contenant le nom du jour de la semaine à la div de notre exemple. Les sept paragraphes peuvent être créés dans un fragment d'arbre. Ce dernier peut ensuite être ajouté sous la balise div. Le code suivant fournit un exemple de création de paragraphes par l'utilisation d'un objet DocumentFragment :

```
var zone = document.getElementById("maZone");
var fragment = document.createDocumentFragment();
var jours = ["Lundi", "Mardi", "Mercredi", "Jeudi", "Vendredi", "Samedi", "Dimanche"];
for (var i=0; i < jours.length; i++){
   var paragraphe = document.createElement("P");
   var texte = document.createTextNode(jours[i]);
   paragraphe.appendChild(texte);
   fragment.appendChild(paragraphe);
}
zone.appendChild(fragment);
```

Ce code permet d'obtenir l'arbre DOM suivant

```
<div id="maZone">
   <p>Lundi</p>
   <p>Mardi</p>
   <p>Mercredi</p>
   <p>Jeudi</p>
   <p>Vendredi</p>
   <p>Samedi</p>
   <p>Dimanche</p>
```

```
</div>
```

Manipulation des attributs :

getAttribute (Identifiant de l'attribut) : Référence un attribut d'un élément en utilisant son identifiant.

hasAttribute (Identifiant de l'attribut) : Détermine si un attribut est présent pour un élément.

removeAttribute (Identifiant de l'attribut) : Supprime un attribut pour un élément.

setAttribute(Identifiant de l'attribut ainsi que sa valeur) : Crée un attribut ou remplace un attribut existant d'un élément.

Le code suivant met en œuvre les attributs dans notre fragment HTML exemple :

```
<div id="maZone">
   <span><b>Un texte</b></span>
   <div id="monAutreZone">
      Un autre texte
   </div>
</div>
```

```
var zone = document.getElementById("maZone");
zone.setAttribute("type", "UnType");
```

le code donne :

```
<div id="maZone" type= « UnType»>
   <span><b>Un texte</b></span>
   <div id="monAutreZone">
      Un autre texte
   </div>
```

```
</div>

var elt = document.getElementById("maZone");

var type = elt.getAttribute("type");  // type content UnType
```

Les méthodes hasAttribute et removeAttribute permettent respectivement de vérifier la présence d’un attribut et de supprimer un attribut, comme dans le code suivant :

```
var zone = document.getElementById("maZone");

if( zone.hasAttribute("type") ) {

    zone.removeAttribute("type");

}
```

L’attribut innerHTML :

Le code ci-dessous donne un exemple d’utilisation de l’attribut innerHTML afin de remplacer le contenu de la balise span définie en début de chapitre par un ensemble

de balises HTML :

```
var zone = document.getElementById("maZone");

var span = zone.getElementsByTagName("span")[0];

span.innerHTML = "<p><b>mon texte</b></p>";

<div id="maZone">

    <span><p><b>Un texte</b></p></span>

    <div id="monAutreZone">

        Un autre texte

    </div>

</div>
```

Les formulaires en HTML :

HTML et JavaScript permettent de réaliser une application Web de base avec les composants de formulaires.

Les formulaires sont des objets d'interface graphiques dotés d'une gestion des données reconnue par tous les navigateurs.

L'utilisation des feuilles de style permet de donner au formulaire le look d'un document écrit avec des champs alignés, des couleurs, etc.

Voir le détail des objets de formulaire et comment on les utilise. La démonstration des objets de formulaire montre comment sont passés les paramètres pour chacun.

Ces articles sont basés sur la Spécification HTML 4.

HTML 5 complète l'ensemble des composants :

HTML 5 n'est pas encore un standard, mais il peut être intéressant de voir les nouveaux objets qu'il propose pour les formulaires d'autant qu'il se veut conçu pour les applications Web.

Input
Cet élément générique à déjà quelques types en HTML 4 (button, submit), mais dans la version 5 il en possède toute une collection. Notamment:

- number : pour une valeur numérique.
- date : pour entrer une date.
- telephone : un numéro de téléphone.
- color: un tableau pour choisir une couleur
- file upload: pour télécharger des fichiers. Les types peuvent être précisés (vidéo, image...).

menu
Plus complet que *select* en HTML 4, il permet facilement de créer un menu ou une barre d'outils avec des images.

datalist
Retourne une liste d'éléments choisis. Proche de *select*.

output
Affiche le résultat d'un calcul.

datagrid
Représente une arborescence ou une table interactifs. Le contenu peut évoluer facilement.

On accède aux données du formulaire coté client ou serveur :

L'accès aux données en JavaScript permet de faire des contrôles sur le contenu entré par l'utilisateur, de traiter les données dans la page ou avec Ajax.

Les données peuvent aussi être utilisées sur le serveur directement par un script dont le nom de fichier est assigné à l'attribut *action*.

Et tout cela permet de créer une interface utilisateur en HTML... :

Les objets de formulaires sont suffisamment variés pour permettre de créer une interface utilisateur graphique pour une application Web.
Cependant il est courant d'utiliser un framework Ajax pour disposer de plus de widgets. En fait HTML avec JavaScript et CSS suffiraient à créer tous les éléments d'interface mais les frameworks permettent surtout d'éviter de refaire ce qui a déjà été fait.
Ils ne sont pas indispensables. On peut sans framework créer par exemple un TabPanel, (un groupe de panels à onglets). Ce n'est qu'un objet de formulaire modelé graphiquement par une feuille de style.

Les autres composantes d'interfaces: barres d'outils, menus, fenêtres, boutons radios et autres sont définissable aussi facilement à partir d'objets de formulaires.

Création et utilisation d'un formulaire :

HTML 4 et JavaScript permettent de réaliser une application Web de base avec les composants de formulaires.

On crée un formulaire avec la balise form *:*

Cela se faire en insérant les balises <form> et </form> dans la section <body> de la page HTML. Mais les éditeurs de pages ont une commande pour insérer un formulaire

automatiquement.
On place ensuite les objets de formulaire à l'intérieur de la balise *form*.

La balise form a des attributs pour l'accès dynamique dans la page par le DOM et pour l'envoi des données.

Les attributs du formulaire :

id

L'identifieur unique.

name

Le nom, son rôle est équivalent à celui de l'identifieur sauf pour les fonctions du DOM, car la méthode getElementById se base sur l'id.

action

Le nom d'un script à charger lorsque l'on soumet le formulaire, avec le bouton du type submit. Noter que les noms et valeurs des composants graphiques sont donnés automatiquement en paramètres du script.

onSubmit

Evènement déclenché lorsque le formulaire est soumis et avant l'exécution de l'action. Elle peut permettre de déterminer si l'action doit être exécutée ou non.

On accède aux composants de formulaire par le DOM :

Pour accéder aux éléments du formulaire en JavaScript on peut utiliser les méthodes du DOM ou chainer le nom (ou l'identificateur) des éléments et sous-éléments à partir de l'objet racine qui est *document*.
Par exemple, un formulaire à pour nom *monform*, il contient le bouton *monbouton*, et ce bouton à un libellé assigné à l'attribut *value*, on peut le connaître avec la chaîne:

```
document.monform.monbouton.value
```

ou l'assigner dynamiquement selon la même voie:

```
document.monform.monbouton.value="valeur"
```

Il est possible comme on le verra dans l'exemple de bouton ci-dessous, si c'est le composant graphique lui-même qui indique au code JavaScript le nom du composant à traiter, d'utiliser le mot-clé **this**, (en français *ceci*), donc l'objet dont l'attribut contient la référence à lui-même.

Alternativement si on a donné un identificateur à un élément, on peut y accéder par l'attribut ID:

```
var val = document.getElementById("xxx").value;
```

Exemple simple du bouton

Les boutons sont de plusieurs types, lesquels sont spécifiés par l'attribut *type*:
button, qui correspond au composant graphique,
submit, qui est associé au formulaire et qui déclenche le chargement du fichier assigné à l'attribut *action*.
image: le bouton est une image chargée dans un fichier.

Les attributs sont donc type, id, name et value pour un texte ou src pour une image.
L'attribut value contient le libellé qui doit être affiché sur le bouton.

Pour associer une action à un bouton simple, de type "button", on utilise l'évènement onClick auquel est assignée une fonction JavaScript.
Exemple, à l'intérieur du formulaire:

```
<button name="Mon-Bouton" value="Mon Bouton" onClick="click(this);" />
```

et dans la section HEAD, le code JavaScript:

```
<script language="JavaScript">
function clic(element)
{
   alert("Cliqué sur " + element.name);
}
</script>
```

Si l'on attribue un id au bouton, on peut accéder directement au bouton grâce à cet id et à une fonction du DOM:

```
<button id="monid" name="Mon-bouton" onClick="clic();" />
```

et dans le code JavaScript:

```
var id = document.getElementById("monid");
alert("Cliqué sur " + id.value);
```

Code complet:

```
<form method="GET" action="">
  <input type="button" name="Mon bouton" value="Cliquez" onClick="clic(this);">
</form>

function clic(element)
{
  alert("Cliqué sur " + element.value);
}
```

Exemple avec une image

On peut utiliser une image comme bouton. Le code JavaScript est inchangé, seuls changent le type du bouton, et l'attribut value est remplacé par l'attribut src pour désigner le fichier de l'image à utiliser:

```
<input type="image" nom="Sauvegarde" src="save.gif" onClick="click(this);" />
```

Code:

```
<form method="post" action="">
  <input type="image" name="Sauvegarde"  onClick="clic(this);" src="save.gif"/>
</form>

function clic(element)
{
  alert("cliqué sur " + element.value);
}
```

Exemple avec OnSubmit

Nous avons parlé de l'attribut onSubmit à propos du formulaire. Nous allons en donner un exemple d'utilisation.
Le code du formulaire:

```
<form name="Monform" action="hello" onSubmit="return submitcheck(this);" />
```

Le mot réservé *return* indique que l'on doit exécuter conditionnellement l'action selon la valeur vrai ou faux qui est retournée par la fonction.

Le code JavaScript simplifié (dans un exemple réel on retournerait une valeur différente selon un traitement quelconque):

```
function submitcheck(element)
{
  alert("L'action à exécuter est " + element.action);
  return false;
}
```

Code complet:

```
<form method="post"  onSubmit="return submitcheck(this);" >
  <input type="submit" name="Mon bouton" value="Cliquez" onClick="clic(this);">
</form>

function clic(element)
{
 alert("Cliqué sur " + element.name);
}

function submitcheck(element)
{
 alert("L'action a exécuter est " + element.action);
 return false;
}
```

Frames et formulaires :

On déconseille aux webmasters d'utiliser les frames car elles barrent la route aux robots des moteurs de recherche entre autres inconvénients, notamment celui de la sécurité.

Mais pour une application Web, vous pouvez avoir besoin des frames et il importe peu que les robots indexent les pages.

En fait on constate un retour étonnant des frames sur des sites importants: Google (Image Search par exemple), Digg (pour sa barre d'outils), Facebook et Stumbleupon parmi d'autres. Le procédé à ses inconvénients et ses avantages.

Parmi les inconvénients, citons l'impossibilité de passer nativement des paramètres de formulaires aux frames dans un frameset. Mais on va voir comment il peut être dépassé.

On peut transférer de l'information entre frames avec l'objet window :

On définit un jeu de deux cadres:

```
<frameset rows="300,*">
  <frame src="frame1.html" name="topFrame" id="topFrame"  />
  <frame src="frame2.html" name="bottomFrame" id="bottomFrame"  />
</frameset>
```

Pour passer de l'information entre frame1 et frame2 on utilise:

```
window.top
```

A quoi on associe le nom de la frame, et le chemin d'accès à un élément, par exemple:

```
window.top.topFrame.document.getElementById("frame2");
```

Pour le tester on crée un formulaire dans chaque frame pour entrer du texte, et un champ pour insérer le texte venant de l'autre frame.

Dans la frame du haut, on va insérer des données venant de celle du bas dans l'élément dont l'id est *frame1*.

```
<div id="frame1">Empty </div>
```

Le code JavaScript dans la frame du bas, il lit la valeur d'un champ de texte du formulaire et l'envoie à la frame du haut en passant par *window.top*.

```
function sendit()
{
  var content = document.form2.mytext.value;
  window.top.topFrame.document.getElementById("frame1").innerHTML = content;
}
```

il est associé à l'évènement *onclick* d'un bouton:

```
<input type="button" name="Submit" value="Send to top" onclick="sendit()" />
```

On transmet les données d'un formulaire entre frames avec PHP :

Si l'on veut utiliser l'attribut "action" du formulaire et transmettre ses données à une autre page, on peut passer par la page contenant le jeu de cadres, mais celle-ci doit alors ajouter les paramètres, par exemple avec du code PHP.

```
<frameset rows="300,*">
  <frame src="frame1.html<?php parameters();?>" name="topFrame" id="topFrame"  />
  <frame src="frame2.html" name="bottomFrame" id="bottomFrame"  />
</frameset>
```

La fonction PHP qui récupère et transmet les données est très simple:

```
function parameters()
{
  $login=$_POST['login'];
  $password=$_POST['password'];
  if(!empty($login))
  {
    echo "?login=$login&password=$password";
  }
}
```

On peut généraliser le code en récupérant toutes les clés de $_POST et toutes les valeurs et en formant une chaîne débutant par le code "?" avec les couples clé-valeur séparés par "&".

Passer des paramètres à une autre page Web :

Comment passer des paramètres à une page web?

Pour cela on crée un formulaire dont les valeurs seront transmises automatiquement, et dans la page de destination, un script va recevoir les valeurs transmises.

Nous avons vu comment créer un formulaire, ici nous allons détailler comme extraire les données transmises.

1) Comprendre le format des paramètres d'URL :

Trois symboles sont utilisés pour ajouter une chaîne de paramètres à une URL:

```
?  concatène l'URL et la chaîne de paramètres.
&  sépare les paramètres multiples.
=  assigne une valeur à une variable.
```

Exemple:

```
http://www.xul.fr/demo.html?login="moi"&password="1234"
```

Dans cet exemple, nous avons deux paramètres, login et password, auxquels sont assignés les valeurs "moi" et "1234".

2) Les valeurs sont transmises du formulaire au serveur :

Il n'y a rien de particulier à faire pour envoyer les valeurs: toutes les variables et leurs valeurs dans un formulaire sont automatiquement envoyées pourvu que la propriété "action" du formulaire désigne le nom d'un fichier à charger.

L'attribut "name" de chaque élément de formulaire fournit le nom de la variable et l'attribut "value" fournit sa valeur.

```
<form action="param-back.html" method="GET">
...divers composants graphiques...
</form>
```

La méthode GET ajoute les données à l'URL, tandis que la méthode POST les transmettrait directement.

Envoyer des données sans formulaire :

Si l'on veut passer des paramètres à un script ou une page sans afficher un formulaire (mais en utilisant une balise *form*), on utilise le champ "hidden":

```
<form action="autrepage.html">
   <input type="hidden" name="nomvar"  value="12345" />
</form>
```

Ce formulaire non apparent passera à *autrepage.html* l'argument nomvar=12345.

3) Récupérer les données incluses dans l'URL dans la page :

L'attribut *location.search* contient la chaîne des paramètres, il reste à l'analyser.

Voici le code complet pour le traitement des données transmises:

```
<script language="JavaScript">
 function processUser()
 {
   var parameters = location.search.substring(1).split("&");

   var temp = parameters[0].split("=");
```

```
    l = unescape(temp[1]);
    temp = parameters[1].split("=");
    p = unescape(temp[1]);
    document.getElementById("log").innerHTML = l;
    document.getElementById("pass").innerHTML = p;
  }
</script>
```

Explications:

1. location.search est la propriété qui contient la liste des paramètres.
2. substring(1) saute le symbole ? et retourne la chaîne sans ce symbole.
3. split("&") découpe la chaîne et retourne un tableau dont les éléments sont les paramètres.
4. on assigne ce tableau à la variable "parameters". On peut alors accéder aux éléments particuliers en indiçant le tableau. Parameters[0] est le premier élément.
5. il faut encore découper le paramètre dans un autre petit tableau qui contiendra le nom de la variable et la valeur.
6. dans cet exemple, nous utilisons seulement la valeur, aussi nous indiçons le petit tableau sur le second élément, temp[1].
7. la fonction unescape convertit les caractères spéciaux.
8. on a assigné la variable l avec la valeur de login et la variable p avec le mot de passe.
9. le login est écrit dans le champ log grâce à la méthode getElementById.
10. et le mot de passe dans le champ pass.

4) Compléter la page chargée avec les données reçues :

Dans cet exemple, je suppose que l'on veut écrire les données dans la page qui reçoit les données.
La variable login a été assignée dans le code précédent.
Deux champs ont été définies dans la page:

```
<div id="log"></div>
<div id="pass"></div>
```

Les champs sont identifiés par la propriété id. Pour les remplir avec les données, on utilise la méthode de DOM getElementById("") et la propriété innerHTML.

```
getElementById("log").innerHTML = login;
```

GET vs POST :

Faut-il utiliser plutôt la methode GET pour envoyer les données d'un formulaire HTML, ou la méthode POST?

GET et POST sont des méthodes d'accès définies dans le protocole HTTP et reprises dans la spécification HTML.
Le choix de la méthode dépend de la façon dont les données sont reçues, de la taille et la nature des données.

La méthode GET ajoute les données à l'URL :

Dans un formulaire, elle est spécifiée ainsi:

```
<form method="get" action="page.html">
</form>
```

Avec cette méthode, les données du formulaire seront encodées dans une URL. Celle-ci est composée du nom de la page ou du script à charger avec les données de formulaire empaquetée dans une chaîne.
Les données sont séparées de l'adresse de la page pas le code ? et entre elles par le code &.

Ainsi si on envoie à *page.html* les valeurs "couleur bleu" et "forme rectangle", l'URL construite par le navigateur sera:

```
http://www.xul.fr/page.html&couleur=bleu&forme=rectangle
```

La spécification HTML 4 demande que l'on utilise GET quand la requête ne cause pas de changement dans les données, donc opère une simple lecture.

Les données de formulaire doivent être uniquement des codes ASCII. La taille d'une URL est limitée à par le serveur, souvent un peu plus de 2000 caractères, en comprenant les codes d'échappement.

Noter que lorsqu'on utilise le bouton retour, les requêtes GET sont exécutées à nouveau.

La méthode POST n'a pas de taille limite :

Dans un formulaire, elle est spécifiée ainsi:

```
<form method="post" action="page.php">
</form>
```

Elle envoie un en-tête et un corps de message au serveur. Le corps est généralement constitué des données entrées dans le champ de formulaire par l'utilisateur.

Les données du formulaire n'apparaissent pas dans l'URL. En conséquence, il n'est pas possible de récupérer directement les données en JavaScript , il faut ajouter du code PHP dans la page:

```
<?php
 $couleur = $_POST['couleur'];
 $forme = $_POST['forme'];
?>
... code HTML ...
```

On peut toutefois assigner les données récupérées en PHP à un script JavaScript:

```
<script>
 var couleur = <?php echo $couleur;?>;
 var forme = <?php echo $forme;?>;
</script>
```

En gros ,
La méthode GET est la valeur de méthode par défaut. On l'utilise de préférence sauf si on ne veut pas que les paramètres soient ajoutés à l'URL.Elle permet de récupérer les données passées à la page avec du code JavaScript.

On verra aussi comment transmettre les données de formulaire à des pages différentes ou des scripts sur le serveur.

Le formulaire lui-même est créé par la balise *form* :

```
<form name="myform" id="myform" action="page" onSubmit="return fun()">
...objets...
</form>
```

L'attribut *name* ou id permettent d'accéder par script à son contenu. Il est préférable d'utiliser les deux attributs avec le même identificateur, par souci de compatibilité.

L'attribut *action* indique la page à laquelle on envoie les données du formulaire. Si cet attribut est vide, c'est la page qui contient le formulaire qui sera rechargée avec en paramètre ces données.

L'attribut onSubmit permet d'associer une fonction de test sur le formulaire. Si la fonction returne false, les données de formulaire ne sont pas envoyées, on reste sur la même page.

- type="password" pour un champ au contenu caché.
- size="" largeur du champ de texte.
- maxlength="" nombre maximal de caractères
- L'attribut *checked* permet de cocher initialement la case.

Envoi et réception de données selon les objets de formulaire :

Les données d'un formulaire peuvent être traitées par un script contenu dans la même page, ou elles peuvent être transmises à une autre page afin que les valeurs définissent le contenu de cette page.

Dans un précédent article, on a vu comment transmettre des données d'une page HTML à une autre. Le présent article va expliquer plus en détail l'utilisation des formulaires pour transmettre des données, à une page ou un script sur le serveur.

Les données du formulaire sont utilisées dans la page, ou une autre :

Dans le cas où les valeurs sont traitée par un script dans la page qui contient le formulaire (ou un fichier inclus), l'attribut *action* de l'objet *form* n'aura pas de valeur.

```
<form action="" >
```

Au bouton qui envoie les données, on associe dans ce cas un évènement *onClick* qui appelle la fonction JavaScript définie pour traiter les valeurs du formulaire.

```
<script type="text/javascript>
 function formulaire() { ... }
</script>


<input type="button" onClick="formulaire()" >
```

Le formulaire pourrait aussi être envoyé avec un objet image.

Pour accéder localement aux éléments, on les identifie par une chaîne formée des mots document, du nom du formulaire, du nom de l'objet, et de l'attribut valeur.

Si un formulaire à pour attribut *name* "monform", un objet texte nommé "montext", la valeur s'obtient ainsi:

```
var x = document.monform.montext.value;
```

On assigne à l'attribut action le nom de fichier auquel envoyer les données :

Lorsqu'au contraire on veut transmettre les valeurs à une autre page ou un script, on assigne le nom du fichier à l'attribut *action*:

```
<form method="GET" action="monfichier.php" >
```

Que le fichier soit une page HTML ou un script PHP, ou dans un autre langage, cela ne fait pas de différence quand à l'envoi, c'est la réception des valeurs qui sera différente. Cependant il reste à définir comment sont extraites les valeurs des éléments du formulaire selon le type d'objet.

Les noms des objets de formulaire et leurs valeurs sont transmis :

La transmission des valeurs à une forme générale unique, mais des principes de fonctionnement qui dépendent de l'objet.

Dans la plupart des cas, les attributs *name* et *value* sont utilisés pour former une chaîne que l'on transmet en paramètre au fichier.

```
<input type="text" name="montexte" value="un texte">
```

donne la chaîne

```
montexte=un+texte
```

Les variables sont séparées par le symbole & et l'ensemble est séparé du nom de fichier par le symbole ?.

Exemple:

```
monfichier.php?montext=un+text&checkbox=cb1
```

Cette chaîne de paramètres est formée automatiquement, il suffit d'assigner le nom de fichier à l'attribut *action* du formulaire.

Les cas particuliers sont:

1. Les zones de texte.
   C'est le contenu de la balise qui constitue la valeur.

   <input type="textarea"> Contenu </>

2. Les cases à cocher.
   Le nom et la valeur sont transmises lorsque la case est cochée, on lorsque checked est ajouté à la définition et que l'utilisateur ne décoche par la case.
   Sinon l'élément est ignoré, rien n'est transmis.
3. Les groupes boutons radios.
   Les boutons radios appartenant à un groupe ont le même nom, par exemple "radio", et une valeur spécifique, comme "radio1", "radio2", etc.
   Le nom "radio" est transmis avec la valeur du bouton sélectionné, ce qui forme par exemple:

   radio=radio2

   si c'est le second bouton qui est coché.

4. Les menus et listes.
   Ils contiennent plusieurs balises *option*. C'est le contenu de la balise *option* sélectionnée qui forme la valeur associée au nom de la liste.
5. Champ image.
   Quand on clique sur un champ image, cela envoie les données du formulaire. Quand à l'objet image, deux chaînes sont créés pour les valeurs x et y indiquant la position où l'on a cliqué dans l'image.
   Si par exemple, le nom est "image", et que l'on clique en position x=10 et y=20, la chaîne suivante sera transmise:

   image.x=10&image.y=20

Cas particulier des cases à cocher :

La *checkbox* pose un problème dans le cas où elle n'est pas cochés, si le script qui reçoit les valeurs à besoin de tous les éléments du formulaire, puisque rien n'est transmis dans ce cas . On peut contourner aisément ce problème en assignant le champs *value* avec l'évènement *onClick*.

On veut que la checkbox "cb" ait une valeur "yes" quand elle est cochée, et "no" quand elle ne l'est pas. On initialise la valeur avec ce qui correspond à l'état initial, s'il est *checked*, avec "yes".

```
<input type="checkbox" name="cb" value="yes" checked onClick="cbChange(this)" >
```

Une fonction met à jour la valeur quand l'état de la case à cocher change:

```
function cbChange(element)
{
   if(element.checked)
     element.value="yes";
   else
     element.value="no";
}
```

Nous allons voir maintenant comment les valeurs envoyées sont récupérées par la page qui les reçoit.

On peut récupérer les données en JavaScript ou en PHP

Les paramètres associés à l'URL d'une page sont assigné à l'attribut *search* de l'objet *location*. On peut afficher la chaîne avec le script JavaScript suivant, qui est utilisé dans notre démonstration.

```
function receive()
{
   var parameters = location.search;
   document.getElementById("string").innerHTML = parameters;
}

window.onload=receive;
```

L'identificateur "string" est l'ID d'une balise où l'on va stocker la chaîne pour l'afficher.

Pour isoler les éléments de cette chaîne de paramètre, on procède par étapes

1. On saute le symbole ? avec un appel à la méthode *substring*.

   ```
   location.search.substring(1)
   ```

2. On découpe la chaîne en éléments selon le séparateur & et la méthode *split*.

   ```
   location.search.substring(1).split("&");
   ```

3. On dissocie chaque élément en nom et valeur de la même façon.

Le scripte complet devient:

```
var parameters = location.search.substring(1).split("&");
var data = "";
for (x in parameters)
{
   var temp = parameters[x].split("=");
   thevar = unescape(temp[0]);
   thevalue = unescape(temp[1]);
   thevalue = thevalue.replace("+", " ");
   data += thevar + "=" + thevalue + "<br>";
}
document.getElementById("data").innerHTML = data;
```

Les éléments de la chaîne de paramètres deviennent les éléments d'un tableau indicé par *x*. On obtient les noms et les valeurs, Pour cet exemple, on les concatène dans la variable *data* afin de les afficher.

Si les données sont envoyées à un script PHP sur le serveur, ou une page contenant du code PHP, elles seront disponibles dans les variables globales $_GET ou $_POST, selon la méthode d'envoi utilisées. Ce sont des tableaux associatifs.

On récupére les données par les clés. Par exemple, on a en HTML:

```
<form method="POST" action="xxx.php">
<input type="text" name="login" value="xxx" />
</form>
```

Le code PHP sera:

```
$login = $_POST['login'];
```

On peut annuler l'envoi en assignant *false* à onSubmit :

Il convient avant d'envoyer les données de vérifier que tous les champs utiles ont été remplis. Dans le cas ou ce n'est pas le cas, l'envoi doit être annulé.
Pour ce faire on ajoute le gestionnaire d'évènement *onSubmit* à la balise *form* :

```
<form method="post" action="sendmail.php" onSubmit="return test()">
```

La fonction de test retourne la valeur *true* si les données peuvent être envoyées ou *false* si l'envoi est annulé.

Personnaliser la balise HTML *input text* :

Un objet HTML pour entrer une simple ligne de texte dont la contenu fera partie des données d'un formulaire.

```
<form name="nomformulaire" action="" method="POST">
   <input type="text" name="nomtext" id="idtext" value="" />
</form>
```

On peut accéder aux attributs de l'objet par l'id ou par la chaîne de noms comme ceci:

```
document.nomformulaire.nomtext
```

Les objets *input password* et *textarea* sont similaires à *input text*.

Attributs de input text

La plupart sont des attributs de *input*. *Maxlength* et *readonly* sont spécifiques.

name et id

Nom de l'objet et l'identificateur.
Le nom a deux utilisations. Il sert à accéder à l'élément, mais devient aussi un nom de variable dans les données de formulaire quand elle sont passées par POST ou GET.
Ainsi si le nom est "montext" et le contenu de *value* "contenu", les données de formulaire contiendront &montext=contenu.

value

Contiendra le texte entré par l'utilisateur. On peut l'assigner avec un texte par défaut.

```
<input type="text" value="
Texte affiché par défaut" />
```

Ce contenu peut être assigné dynamiquement, pour ajouter un contenu aux données de formulaire, ou lu dynamiquement par un script.
Exemple de lecture:

```
var contenu = document.nomformulaire.nomtext.value;
```

size

Largeur initiale du champ d'entrée de texte. Dans le cas présent, la taille correspond à un nombre de caractères (pour les autres objets c'est un nombre de pixels).
La largeur en pixels correspond à une largeur moyenne pour les polices de caractères.

maxlength

Nombre maximal de caractères que l'utilisateur peut entrer, ou que l'on peut assigner à value.
Il peut être supérieur à *size*, dans ce cas le contenu défile dans le champ de texte.

disabled

L'objet est désactivé au chargement de la page, et affiché en gris.

```
<input type="text" disabled />
```

Pour l'activer, un script assigne l'attribut avec la valeur *false*.

```
document.nomformulaire.nomtext.disabled=false;
```

readonly

Cet attribut interdit de modifier le contenu de l'attribut *value*. A la différence de *disabled*, l'objet est affiché normalement bien que l'utilisateur ne puisse rien entrer.On modifie cet état comme pour l'attribut *disabled*:

```
document.nomformulaire.nomtext.readonly=false;
```

Parmi les attributs hérités: title, style, tabindex, class.

Evènements

onfocus

Réagit quand l'objet obtient le focus, donc quand l'utilisateur clique sur le champ de texte.

onblur

Le focus est perdu (l'utilisateur clique ailleurs).

onchange

Quand un caractère est entré ou modifié mais est déclenché quand le champ de texte perd le focus.

onselect

Une partie du texte est sélectionnée.

On commande aussi la sélection du texte par la méthode *select()*.

```
document.monformulaire.montext.select();
```

onclick

L'utilisateur clique dans le champs de texte.

accesskey

Permet d'assigner une touche ou une combinaison. Quand l'utilisateur tape cette touche, le champs de texte obtient le focus, il peut entrer du texte.

Exemple:

```
<input type="text" accesskey="t" >
```

A ces évènements s'ajoutent ceux de l'objet hérité: ondblclick, onmousedown, onmouseup, onemousemove, onmouseover, onmouseout, onkeypress, onkeydown.

Des balises spécialisées selon les types de données :

Password

Il est équivalent à l'objet *text* avec la différence que le texte assigné à l'attribut *value* est masqué, et remplacé par des astérisques.

```
<input type="password" value="">
```

Son utilisation et son fonctionnement sont les mêmes.

Textarea

Il fonctionne comme *text*, mais affiche plusieurs lignes et la valeur est contenue entre la balise ouvrante et fermante.

Attributs:

- **row**
  Nombre de lignes visibles. Remplace l'attribut size.
- **cols**
  Nombre de caractères par ligne. La largeur en pixels correspond à la largeur moyenne.

Exemple:

```
<textarea rows="5" cols="40">
  Contenu initial.
</textarea>
```

On peut utiliser *textarea* tout comme *text* pour ajouter des données statiques à un formulaire, avec l'attribut *readonly*.

Comment personnaliser un champ d'entrée de texte :

On utilise des gestionnaires d'évènements pour prendre compte les actions de l'utilisateur et CSS pour améliorer l'apparence de l'objet graphique.
Exemple:

Actions de l'utilisateur

Le code HTML:

```
<form name="nomformulaire" method="post" action="">
<input name="nomtexte" class="textfield" type="text" size="40" maxlength="60"
  value="Taper un texte"
  onChange="change(this)"
  onFocus="getfocus(this)"
  onBlur="losefocus(this)" >
<input type="button" value="Effacer" onClick="cleartext()">
</form>
```

Le code CSS:

```
.textfield
{
  font-family:"Segoe Print", Verdan, Arial;
  -moz-border-radius:4px;
  border:1px solid gray;
  border-radius:4px;
  background-image:url(images/papier.jpg);
  padding-left:8px;
}
```

Le code JavaScript:

```
function cleartext()
{
  document.nomformulaire.nomtexte.value="";
}
```

```
function change(element)
{
  var storage = document.getElementById("storage");
  storage.innerHTML += element.name + " change<br>";
}
function getfocus(element)
{
  var storage = document.getElementById("storage");
  storage.innerHTML += element.name + " en focus<br>";
}
function losefocus(element)
{
  var storage = document.getElementById("storage");
  storage.innerHTML += element.name + " perd le focus<br>";
}
```

Checkbox: Améliorer le transfert avec les données de formulaire :

La case à cocher peut avoir deux valeurs: vrai ou faux. Lorsque l'on passe les données de formulaire à une autre page, le nom et l'état de la checkbox sont passés quand elle est cochée, sinon, rien n'est passé.

Si la page qui reçoit les données connaît la liste des valeurs à passer, ceci est suffisant: lorsqu'on ne reçoit rien pour une case à cocher, on déduit qu'elle n'est pas cochée...

Si l'on veut quelque chose de plus sûr, il faut ajouter un peu de code JavaScript.

Utilisation basique d'une case à cocher :

Comme tous les objets de formulaire, la checkbox dispose des propriétés *value*, *name* et *id*.

Elle a aussi un attribut sans valeur: *checked*. Celui-ci donne par défaut l'état coché à la case. Mais il peut aussi est utilisé pour connaître son état.

```
<input type="checkbox" name="cb" onchange="alert(this.checked)" />
```

L'évènement onchange est déclenché lorsque l'on coche ou décoche la case. Le code assigned affiche dans une boite de message, l'état: coché ou non.

Avec JavaScript, on peut passer l'état false *avec les données de formulaire :*

Sachant donc comment on peut lire l'état de la case, on veut transmettre cette donnée à une autre page. Pour ce faire on assigne la valeur de l'état, *true* ou *false*, à l'attribut *value* de la checkbox avec du code JavaScript associé à *onchange*.
On assigne aussi à *value* la valeur par défaut.

```
<input type="checkbox"  value="false" onchange="if(this.checked) this.value='true'; else this.value='false';" />
```

On ajoute un bouton à notre démonstration pour tester l'état de la case.

```
<input type="button" onclick="alert(document.form2.cb.value')" />
```

Le formulaire est désigné par form2 et cb est le nom de la checkbox

On peut utiliser une fonction d'usage général.

```
function chState(element)
{
   if(element.checked)
      element.value='true';
  else
      element.value='false';
}
```

Ce qui simplifie le code de la checkbox:

```
<input type="checkbox"  value="false" onchange="chState(this)" />
```

Ou un code encore plus simple:

```
<input type="checkbox"  value="false" onchange="this.value=this.checked" />
```

Dès lors, l'état de la checkbox sera transmis avec les autres données de formulaire, que la case soit cochée ou non.

L'objet de formulaire *select* et l'envoi des données :

Pour créer des listes et des menus en HTML, on utilise les balises *select* et *option*. Mais comment récupérer l'option choisie par l'utilisateur avec les autres données de formulaire?

Voici la syntaxe de la balise *select*:

```
<form method="GET" action="page.html">
  <select name="">
    <option value="" selected></option>
    <option value=""></option>
    <option value=""></option>
    ....
  </select>
</form>
```

Les formulaires transmettent les données sous la forme nom=valeur ou nom est l'attribut *name* de l'objet et valeur l'attribut *value*.
Dans le cas d'une liste d'options, le nom qui est passé avec les données de formulaire est celui de la balise *select* et la valeur est celle de l'attribut *value* de l'option choisie.

Par exemple:

```
<select name="maliste">
  <option value="option1"></option>
  <option value="option2"></option>
</select>
```

Si on a choisi la deuxième option, avec les données de formulaire sera envoyé ceci:

```
maliste=option2
```

Et si une option à l'attribut *selected* et que l'utilisateur n'a fait aucun choix, c'est la valeur de cette option qui sera passée.

L'attribut selected *détermine un choix par défaut :*

Par défaut, c'est la première option qui est sélectionnée si l'utilisateur ne fait aucun choix.
On peut changer l'option sélectionnée par défaut en lui associant l'attribut selected.
Exemple:

```
  <select name="couleur">
    <option value="orange"></option>
    <option value="bleu" selected></option>
    <option value="rouge"></option>
    ....
  </select>
```

Dans l'exemple, le menu affiche la couleur bleue par défaut et c'est cette option qui sera envoyées avec les données de formulaire, si l'utilisateur ne change pas le choix.

On écrit quelque fois aussi l'attribut *selected*:

```
<option value="bleu" selected="selected"></option>
```

Cela ne semble rien apporter de plus, tout au moins avec les navigateurs modernes.

L'attribut *multiple* permet des sélectionner plusieurs options à la fois :

On peut sélectionner plusieurs options à la fois en combinant la touche CTRL avec la souris, si cet attribut est ajouté.

```
<select name="couleur" multiple>
  <option value="orange"></option>
  <option value="bleu" selected></option>
  <option value="rouge"></option>
  ....
</select>
```

Dans ce cas, le nom de l'objet select est passé parmi les données de formulaire avec successivement la valeur de chaque option cliquée.
Ainsi si on sélectionne à la fois la couleur bleue et la couleur rouge, on aura en paramètres dans l'URL :

```
?couleur=bleu&couleur=rouge
```

L'attribut size *définit le nombre de lignes à afficher :*

Il définit le nombre d'options affichées à la fois. Par défaut une seule ligne sera affichée.

Dans la démonstration en bas de page, on voit que l'attribut size="4" affiche quatre lignes, donc quatre options.

L'attribut label *contient l'option affichée en forme courte :*

Une option a une valeur assignée à l'attribut *value*, et un texte correspondant à afficher qui est soit inclut dans la balise, ou assigné à l'attribut *label*.

Dans la forme courte, l'attribut *label* spécifie le texte à afficher. Par exemple cette option en forme longue:

```
<option value="orange">Orange</option>
```

Peut être écrite en forme courte:

```
<option value="orange" label="Orange" />
```

Si un attribut *label* est ajouté alors que la balise à la forme longue avec un contenu inclus, c'est la valeur de *label* qui est affichée et non le contenu inclus.

Mais dans tous les cas c'est ce qui est assigné à *value* qui est envoyé avec les données de formulaire.

La balise optgroup *subdivise les options :*

Il permet de créer une hiérarchie de menus en regroupant les options d'un même sous-menu.

```
<select name="couleur" multiple>
  <optgroup label="Couleur">
    <option value="orange">Orange</option>
    <option value="bleu" selected>Bleu</option>
    <option value="rouge">Rouge</option>
  </optgroup>
  <optgroup label="Taille">
   <option value="un">Un</un>
 </optgroup>
</select>
```

Cela affichera ceci dans la boite de sélection:

Couleur
  Orange
  Bleu
  Rouge
Taille
  Un

Connaître l'option choisie par l'utilisateur grâce à JavaScript :

Pour utiliser *select* en JavaScript, et savoir quelle option l'utilisateur a choisie, on utilise l'attribut *selectedIndex* (défini dans DOM).
Si la balise *select* a pour nom "couleur", l'instruction suivante affiche le numéro de l'option choisie, en partant de zéro:

```
document.forms[0].couleur.selectedIndex;
```

Et pour accéder aux valeurs, avec le tableau *options*:

```
var object = document.forms[0].couleur;
var index = object.selectedIndex;
var value =  object.options[index].value;
var content = object.options[index].text;
```

On peut aussi choisir une option à partir de code JavaScript:

```
document.forms[0].couleur.selectedIndex = 3;
```

Démonstration de la balise select *avec une liste de fruits :*

Comment on passe le choix fait par l'utilisateur dans une liste parmi les données de formulaire envoyées à un script ou une autre page.

Choisir une valeur dans la liste et cliquer sur *Envoyer* pour passer le formulaire à la page, qui sera rechargée.
Cliquer sur *JavaScript* pour afficher le numéro de l'option sélectionnée.

Le code JavaScript:

```
function receive()
{
  var parameters = location.search.substring(1).split("&");
  var temp = parameters[0].split("=");
  varname = unescape(temp[0]);
  if(varname == "") return;
  value = unescape(temp[1]);
  alert(varname + "=" + value);
}
window.onload=receive;
```

Le code HTML avec la méthode GET pour avoir les données de formulaire dans l'URL:

```
<form name="form1" method="get" action="select.php">
  <select name="couleur" id="couleur" multiple size="4">
    <option>orange</option>
    <option selected>bleu</option>
    <option>vert</option>
    <option>rouge</option>
  </select>
  <input type="submit" value="Envoyer">
```

```
  <input type="button" value="Javascript" onClick="show()">
</form>
```

Images dynamiques avec Image Map :

Dans la première partie, nous avons vu comment afficher des informations sur les objets dans une image et comment quand on clique sur les éléments d'une image, afficher une autre page ou une autre image.

Nous allons perfectionner ce dernier point, avec, quand on clique sur un objet dans l'image, le remplacement de l'image par une image de l'objet seul.

Deux méthodes peuvent être employées: soit en utilisant un bookmarklet, soit en ajoutant un gestionnaire d'évènement onClick.

Le code de l'image reste le même dans les deux cas:

```
<img id="img1" src="asin2.jpg" width="500" height="540" border="0" usemap="#map1" >
```

Et le code de base de l'*image map* est aussi commun:

```
<map name="map1">
 <area href="asin1.jpg" shape="circle" coords="240,180,140" " >
 <area href="collar.jpg" shape="rect" coords="60,330,440,520" >
</map>
```

On peut accéder à la balise de l'image, par son ID:

```
document.getElementById("img1").src= "nouvelle url";
```

ou avec le DOM:

```
document.images[0].src = "nouvelle url";
```

Ajouter l'évènement onClick :

On associe l'évènement à la balise *area*, et on assigne une action qui change la source de la balise *img*, pour remplacer l'image affichée.

On neutralise aussi l'attribut *href* avec un bookmarklet nul: *javascript:void(0)*.

```
<map name="map1">
 <area href="javascript:void(0)" shape="circle" coords="240,180,140"
```

```
  onClick="document.images[0].src = 'http://www.xul.fr/images/asin1.jpg'">
 <area href="javascript:void(0)" shape="rect" coords="60,330,440,520"
      onClick="document.images[0].src = 'http://www.xul.fr/images/collar.jpg'">
</map>
```

Utiliser un bookmarklet :

Dans ce cas on remplace l'URL par une fonction JavaScript .

Le code du bookmarklet est le suivant:

```
javascript:(
  function()
    {
    document.getElementById('img1').src='http://www.xul.fr/images/asin1.jpg';
    }
)()
```

Il sera compacté dans l'attribut *href*:

```
<map name="map1">
 <area href="javascript:(function(){document.getElementById('img1').src=
    'http://www.xul.fr/images/asin1.jpg';})()" shape="circle" coords="240,180,140">
 <area href="javascript:(function(){document.getElementById('img1').src=
    'http://www.xul.fr/images/collar.jpg';})()" shape="rect" coords="60,330,440,520">
</map>
```

Limitations :

Le remplacement est irréversible car l'*image map* reste la même:

- On ne peut utiliser le bouton retour.
- Les mêmes zones restent cliquables sur la nouvelle image.

Exemple d'images dynamiques :

Une image peut être remplacée par une autre quand on clique sur un objet, une personne dans l'image initiale.

Dans cette démonstration, l'image map n'est pas remplacée pour la nouvelle image affichée.

Le code utilise deux procédés différents pour remplacer l'image, avec l'ID et avec le DOM. Cela n'a pas de rapport avec les images, c'est juste pour faire une démonstration des deux procédés.

Le code:

```
<map name="map1">
 <area href="javascript:(function(){
   document.getElementById('img1').src='http://www.xul.fr/images/asin1.jpg';})()"
   shape="circle" coords="240,180,140">
 <area href="javascript:void(0)"
   shape="rect" coords="60,330,440,520"
   onClick="document.getElementById('img1').src = 'http://www.xul.fr/images/collar.jpg'">
</map>
```

Remplacer dynamiquement l'image map :

Pour aller plus loin, il faut assigner dynamiquement une nouvelle *image map* a chaque nouvelle image affichée en remplacement.

Les maps

Une map générique sert de conteneur et est associée à l'unique balise *img*.

```
<img id="img1" src="" width="" height="" border="0" usemap="#mapgeneric" >
<map name="mapgeneric" id="mapgeneric">
</map>
```

Une carte est créée pour l'image initiale qui est l'affiche du film:

```
<map name="mapgalactica" id="mapgalactica">
</map>
```

Le contenu de cette carte est copié dans la balise *map* générique.

Une autre carte est créée pour chaque nouvelle image.

Le code JavaScript

Une fonction JavaScript unique effectue le remplacement des images et des cartes. Cette fonction est appelée quand on clique sur une zone définie par une carte.

Exemple:

```
<area href="javascript:replaceImage('img1', 'url', 'nom')" shape="rect" coords="0,0,170,350">
```

La fonction:

```
function replaceImage(imgid, source, mapid)
{
  var image = document.getElementById(imgid);
  image.src= source;
  var newmap = document.getElementById(mapid);
  var origin = document.getElementById("mapgeneric");
  origin.innerHTML = newmap.innerHTML;
}
```

La fonction remplace la source de l'image et utilise l'attribut *innerHTML* pour récupérer le contenu d'une balise *map* et l'assigner à la carte générique associée à la balise *img*.

Le premier paramètre est l'ID de la balise *img*.
Le second est l'URL de la nouvelle image.
Le troisième est l'ID de la nouvelle *image map*.

Image Map : Liens dans une d'image :

Pour permettre à l'utilisateur de cliquer à l'intérieur d'une image afin d'avoir une réponse de la page en rapport avec l'objet désigné dans l'image, on utilise une carte d'image.

Un exemple d'application commun: Sur une photo réunissant plusieurs personnes, lorsqu'on passe simplement la souris sur une de ces personnes, un message apparaît qui affiche son nom et les informations qui la concernent. Nous verrons comment réaliser une telle application.

Créer une carte d'image :

La spécification HTML définit précisément comment interagir avec une image.

1) Insérer une image dans la page :

```
<img src="image.jpg" >
```

2) Ajouter l'attribut "usemap" :

Il sert à indiquer le nom de la carte associée à cette image, sous forme de fragment d'URL.

```
<img src="image.jpg" usemap="#map1" >
```

3) Ajouter une balise "map" dans le code de la page :

```
<map name="map1">
</map>
```

Elle doit nécessairement avoir un attribut "name".

4) Dans cette balise, ajouter une ou plusieurs descriptions de zones cliquables :

Ce sont des balises **area** dotées des attributs **href**, **shape**, **coords**.

Href assigne une page, *shape* définit une forme et *coords* une liste de points.

```
<map name="map1">
   <area href="page.html" shape="rect" coords="0,0,120,120">
</map>
```

A chaque zone on attribue une forme: rectangle (rect), cercle (circle), polygone (poly), puis une série de points pour la délimiter qui dépendent de cette forme, ainsi que l'URL d'une page qui sera chargée quand on clique sur la zone.

Formes et coordonnées :

Dans le cas d'un rectangle les coordonnées sont de la forme x1, y1, x2, y2 (x gauche, y haut, x droit, y bas).
Pour un cercle, x, y, rayon.
Pour un polygone une succession de paires x, y.

Ces coordonnées sont relatives à l'image, le point 0,0 est le coin supérieur gauche de l'image.

Afficher des informations sur les éléments de l'image :

Comment faire pour afficher des messages quand la souris passe sur une zone délimitée dans l'image?

On ajoute simplement un attribut *title* à une zone. La balise <title> devient contextuelle et s'applique aux élément définis à l'intérieur de l'image, par des balises <area>.

```
<map name="map1">
   <area href="" shape="rect" coords="0,0,120,120" title="Message contextuel" >
</map>
```

Pour obtenir les coordonnées des rectangles, on peut utiliser Paint.Net et la fonction de sélection rectangulaire: elle affiche la position de la sélection et ses dimensions. On ajoute alors les dimensions à la position car <area> requiert les coordonnées des quatres coins du rectangle.

Code complet

```
<map name="hollywood" id="hollywood">
  <area href="#" shape="rect" coords="13,63,84,150" title="Gwynneth Paltrow" />
  <area href="#" shape="rect" coords="110,160,190,260" title="Naomi Watts" />
  <area href="#" shape="rect" coords="170,10,250,110" title="Salma Hayeck" />
  <area href="#" shape="rect" coords="300,30,380,130" title="Kirsten Dunst" />
  <area href="#" shape="rect" coords="250,260,340,370" title="Jennifer Aniston" />
</map>
<img src="http://www.xul.fr/images/hollywood.jpg" width="400" height="380"
   usemap="#hollywood" border="0">
```

Utiliser l'objet Window en JavaScript :

Window est un objet qui correspond à la fenêtre dans laquelle s'affiche une page Web. Une telle fenêtre peut être créée dynamiquement.
Ce n'est pas un objet JavaScript, et il n'est pas défini par la spécification DOM du W3C. C'est un standard de fait.

Les attributs et méthodes de l'interface *window* sont reconnus par Internet Explorer 4, Firefox 1, Opera 9, les versions plus récentes et Safari. Ce document se base sur ce qui est supporté par les navigateurs à défaut de standard officiel.

On designe la fenêtre courante par le mot-clé *window* et toute fenêtre créée dynamiquement par le nom donné à l'objet assigné par la valeur de retour de la méthode *open*.

Definition et test des attributs et objets internes à *window* :

Définitions des attributs et objets

Ces attributs peuvent être lus seulement pour certains, une valeur peut leur être assignée autrement.

frames[]

Les frames dans la fenêtre. Lecture seule.

length

Nombre de frames. Lecture seule.

name

Nom de la fenêtre.

status

Texte de la barre d'état.
Sous Firefox, la modification de ce texte est une option. Aller dans Option, Contenu, JavaScript et cliquer sur le bouton Avancé. Puis cocher l'option "Modifier le texte de la barre status".

defaultStatus

Texte par défaut de la barre d'état.

closed

Etat de fenêtre fermée ou non.

opener

Référence sur la fenêtre qui a ouvert cette fenêtre. Exemple:

```
x = window.opener;
```

parent

La fenêtre parent d'une fenêtre donnée. Exemple:

```
x = mywin.parent;
```

top

Le parent de plus haut niveau.

Objets contenus dans *window*

Ces objets qui ont leurs attributs et méthodes propres, ne sont pas détaillés ici, ils ont leur propre page.

document

Désigne une page, celle que contient une fenêtre.

history

Liste des pages précédemment vues dans la même fenêtre

location

Désigne l'URL d'une page, celle que contient la fenêtre

screen

Désigne l'écran et contient les propriétés: width, height, availWidth, availHeight, colorDepth.

Voir aussi les attributs de création de fenêtre dans le chapitre: Ouvrir une fenêtre en JavaScript.

Test des propriétés de l'objet Windows selon le navigateur :

Les propriétés non supportées par un navigateur sont **undefined**.
Certaines propriétés sont utilisables en paramètres du constructeur de fenêtre mais ne sont accessibles directement et ne sont pas listées ici.

innerWidth 1280 innerHeight 665

outerWidth 1296 outerHeight 744

locationbar [object BarProp]menubar [object BarProp]personalbar [object BarProp]statusbar [object BarProp]toolbar [object BarProp]scrollbars [object BarProp]

pageXOffset 0 pageYOffset 0

screenX -8screenY -8

directories undefined

status defaultStatus

opener null

parent [object Window]

top [object Window]

name

length 0

Ne fonctionnent pas avec IE 7

offscreenBuffering undefined

On peut obtenir des valeurs équivalentes sous Internet Explorer et tous navigateurs avec:

document.body.clientHeight 2565

document.body.clientWidth 1247

document.body.offsetHeight 2637

document.body.offsetWidth 1247

document.documentElement.offsetHeight 2725

document.documentElement.offsetWidth 1263

Autres

Crypto [object Crypto]

Attention: La déclaration DOCTYPE peut affecter ces valeurs.

Méthodes de window *et tests*

Définitions des méthodes

alert(message)

Affiche un message.

x = prompt(message)

Présente une boite de dialogue qui demande une réponse utilisateur et retourne la réponse.

x = confirm(message)

Affiche un message, une question normalement, avec deux boutons pour confirmer ou annuler. Retourne *true* ou *false*.

resizeTo(w, h)

Redimensionne à la largeur et hauteur données.

scrollTo(x, y)

Déroule à la position donnée. Le défilement doit être activé pour le fenêtre selon scrollbars, c'est le cas par défaut.

moveTo(x, y)

Déplace à la position donnée.

id = setInterval(expression, millisecondes)

Evalue une expression répétitivement selon l'intervalle de temps donné.

clearInterval(id)

Supprime l'intervalle donné.

id = setTimeout(expression, millisecondes)

Evalue une expression au bout du délai donné.

clearTimeout(id)

Supprime le délai donné.

focus()

Passe le focus à la fenêtre donnée.

blur()

Ote le focus à la fenêtre donnée courante.

createPopup()

Crée une sous-fenêtre.

print()

Lance l'impression du contenu de la fenêtre

Window location en JavaScript :

L'interface *location* permet de connaitre les éléments de l'URL du document dans la fenêtre courante: le nom de domaine, le nom de fichier, etc.

Location est un sous-objet de Windows. C'est un objet coté-client (donc créé par le navigateur) et dont l'interface est implémentée en JavaScript depuis la version 1.0. Dans HTML 5, location est une interface intégrée à DOM et aussi un attribut de *HTMLDocument*.

Une URL se décompose en domaine, nom de fichier... :

Prenons comme exemple d'URL à laquelle on ajoute tous les paramètres possibles:

http://www.xul.fr:80/ecmascript/tutoriel/window-location.php#content?name=value

Elle se décompose ainsi:

[http:]//[www.xul.fr]:[80][/ecmacript/tutoriel/windows-location.php]#[content]?[name=value]

Ce qui, traduit en attributs de l'objet *location* correspond à:

[protocol][host][port][pathname][hash][search]

Les propriétés de *location* dans l'exemple ont les valeurs suivantes:

protocol http:
host [www.xul.fr]:80
hostname www.xul.fr
pathname /ecmascript/tutoriel/window.location.php
port 80
hash #content
search ?name=value

En outre, **href** correspond à l'URL complète.

Location *assigne ces éléments à des attributs :*

Propriétés de location

protocol

Contient le protocole de la page, incluant le caractère deux points.

http:

Ou ftp: etc.

port

Numéro du port, qui est optionnel. Le caractère deux-points est un séparateur et ne fait pas partie de la valeur de la propriété.

hostname

Chaine composée des sous-domaines éventuels, du domaine avec son extension. Exemple:

www.xul.fr

host

Combinaison de hostname + port. Exemple:

www.xul.fr:80

pathname

Répertoire et nom de fichier de la page. Exemple:

/ecmascript/tutoriel/window-location.php

hash

Le terme hash désigne le symbole dièse: # qui introduit une adresse à l'intérieur d'une page. La valeur de la propriété et le nom de l'ancre qui suit le symbole. Exemple:

dans /ecmascript/window-location.php#content, l'ancre est content.

search

Liste des variables et leurs valeurs. Le code ? est inclut dans la valeur de la propriété. Exemple:

?nom=valeur&nom2=valeur2

Il sera facile avec le script de démonstration d'essayer toutes les combinaisons possible et voir les valeur retournées.

Ses méthodes permettent de changer d'URL de la page

Méthodes de location

assign

La méthode permet en changeant la location de charger une nouvelle page. Exemple:

```
window.location.assign("http://www.xul.fr/ecmascript");
```

reload

Recharge la page actuellement affichée.

```
windows.location.reload();
```

replace

Remplace la page affichée par une autre. L'historique devient celui de la page remplaçante.

```
window.location.replace("http://www.xul.fr/ecmascript/");
```

La différence entre *replace* et *assign* est que l'on ne peut plus utiliser le bouton retour après *replace*.

Exemple d'utilisation de location pour changer les composantes de l'URL

Les propriétés de *location* peuvent être lues ou assignées.
Par exemple on affiche le chemin de la page ainsi:

```
document.write(location.pathname)
```

Et on change la page ainsi:

```
location.href = "url"
```

La méthode *reload* permet aussi de recharger la page selon la nouvelle URL.

La démonstation permet d'entrer une URL dans un formulaire, d'afficher les propriétés et d'exécuter les méthodes de l'objet et voir les valeurs des propriétés ainsi changées.

```
var url = document.loc.url.value;
var storage = document.getElementById("storage");
```

```
var href = "href " + location.href
var protocol = "protocol" + location.protocol
var hostname = "";//"hostname" + location.hostname
var port = "port" + location.port
var host = "host" + location.host
var pathname = "pathname" + location.pathname
var hash = "hash" + location.hash
var search = "search" + location.search

var data = href + protocol + hostname + port + host + pathname + hash + search;
storage.innerHTML = data;
```

Pour utiliser la démo, on modifie les paramètres de l'URL qui doit cependant rester celle de la page de démonstration si l'on veut pouvoir afficher ces paramètres avec le bouton "propriétés".
Les autres boutons appellent les méthodes de leur libellé.

Code complet de la démo:

```
function properties()
{
 var url = document.loc.url.value;
 var storage = document.getElementById("storage");
 var href = "href " + location.href ;
 var protocol = "protocol" + location.protocol;
 var hostname = "";//"hostname" + location.hostname;
 var port = "port" + location.port;
 var host = "host" + location.host;
 var pathname = "pathname" + location.pathname;
 var hash = "hash" + location.hash;
 var search = "search" + location.search;
 var data = href + protocol + hostname + port + host + pathname + hash + search;
 storage.innerHTML = data;
}

function replace()
{
 var url = document.loc.url.value;
 window.location.replace(url);
}

function reload()
{
 window.location.reload();
}
```

```
function assign()
{
 var url = document.loc.url.value;
 window.location.assign(url);
}
```

Window.history et JavaScript:

History est un sous-objet de *window*, il sert d'interface avec l'historique de navigation conservé par le navigateur.

Les propriétés et méthodes s'appliquent à l'objet *history* de *window* ou de chaque frame dans *window*.

Obtenir des informations sur le contenu de history :

Les propriétés de l'objet se rapportent à des URLs.

length

Nombre d'URLs dans la liste de l'historique.

length= 1

current, **next** et **previous** ne sont pas communément implémentés.

On peut utiliser *location.href* à la place de *current* pour avoir l'URL de la page courante, mais il s'agit d'une autre objet.

Des méthodes pour utiliser la liste comme moyen de navigation :

Elle servent à passer d'une page à la précédente ou la suivante ou aller à une page donnée.

back()

Charger la page précédente.

forward()

Charger la page suivante, après un retour en arrière.

go(x)

Charger une page dont on donne le numéro dans la liste ou l'URL.
Exemple: history.go(-2) pour annuler les deux derniers clics.

Boites de dialogue et messages d'alerte en JavaScript :

Les boites de dialogues sont des méthodes de l'objet window. On peut cependant les invoquer dans le code JavaScript sans référence explicte à *window*. Elles servent selon le type (*alert, confirm, prompt*) à afficher des messages ou obtenir une réponse de la part de l'utilisateur.

Alert: Afficher un message

```
window.alert("Hello");
```

ou juste, dans la fenêtre courante:

```
alert("Hello");
```

Code de la démonstration:

```
<form name="form1" >
   <input name="alert1" type="text" value="Hello">
    <input type="button" value="Envoyer" onclick="alert(document.form1.alert1.value)">
</form>
```

Confirm: Demander si oui ou non

```
var reponse = window.confirm("Oui ou non?");
```

Une valeur booléenne est retournée, *true* pour oui, *false* pour non.

Code de la démonstration:

```
<form name="form2" >
   <input name="confirm2" type="text" value="Oui ou non?">
    <input type="button"  value="Envoyer" onclick="confirm(document.form2.confirm1.value)">
</form>
```

Exemple pratique d'utilisation de la réponse:

```
var reponse = window.confirm("Votre choix?");
if(reponse)
{
  alert("Oui");
}
else
{
 alert("Non");
}
```

Prompt: Entrer un message

var reponse = window.prompt("Votre réponse?", "Réponse par défaut");

La valeur initiale par défaut est optionnelle. Cette méthode retourne la chaîne que l'utilisateur a entrée ou la valeur par défaut s'il clique sur OK sans rien taper.
S'il clique sur Annuler, la méthode retourne *false*.

Code de la démonstration:

```
<form name="form3" >
   <input name="prompt3" type="text" value="Votre réponse?">
   <input name="prompt4" type="text" value="Peut-être...">
   <input type="button"  value="Envoyer" onclick="confirm(
         document.form3.prompt3.value, document.form3.prompt4.value)">
</form>
```

Exemple d'utilisation du texte entré par l'utilisateur:

```
var reponse = window.prompt("Votre réponse?", "Peut-être...");
if(reponse)
{
  alert(reponse);
}
```

Ouvrir une fenêtre en JavaScript :

Pour afficher des données de façon dynamique, on peut utiliser une iframe ou ouvrir une nouvelle fenêtre.

Propriétés de création de fenêtre :

Ces attributs sont assignés lors de la création de la fenêtre. Ils ont la valeur *yes* ou *no* selon l'activation ou non.

scrollbars Vaut *yes* si les barres de défilement s'affichent et *no* sinon.

statusbar Vaut *yes* ou *no* selon que l'on affiche la barre d'état ou non.

toolbar Vaut *yes* ou *no* selon que l'on affiche la barre d'outils ou non.

menubar Présenter un menu ou non.

resizable On peut changer les dimensions ou non. Le coin inférieur droit est modifié en conséquence.

directories Inclure les boutons de favoris.

Méthodes

open(url, nom [, liste de propriétés])
Ouvre une nouvelle fenêtre. L'URL et le nom sont donnés en paramètres ainsi que la liste des options. Comme exemple d'options, on a statusbar, toolbar, scrollbar, width, height...
Les options sont encloses entre guillemets simples ou doubles, et à l'intérieur, séparées par une virgule.

close()
Ferme la fenêtre. Exemple: x.close(); window.close().

Démonstration de création d'une fenêtre :

Cette démonstration par l'exemple permet de définir toutes les propriétés d'une fenêtre à partir d'un formulaire et de la créer en cliquant sur un bouton.
Il s'agit d'un programme de base à adapter à votre application.

Démonstration de la méthode *open* de l'objet HTML *window*.
Sélectionner les propriétés de la fenêtre à définir et cliquer sur le bouton *open* Le code des options va s'affiché sous le formulaire.

Dans cette démo, une page est chargée mais son contenu est effacé par la commande:

```
win.document.write("Hello!");
```

Sans cette commande la fenêtre ne reste pas affichée sous IE7. Il vous appartient d'adapter le code selon votre application.

location=yes,toolbar=non,menubar=non,scrollbars=non,statusbar=non,resizable=non,directories=non, width=600,height=300

Code

```
function yesno(arg)
{
  if(arg) return "oui";
  return "non";
}

function demo()
{
  var t = document.myform.checktool.checked;
  var m = document.myform.checkmenu.checked;
  var s = document.myform.checkscroll.checked;
  var u = document.myform.checkstatus.checked;
  var r = document.myform.checkresize.checked;
  var d = document.myform.checkdir.checked;
  var w = document.myform.wwidth.value;
  var h = document.myform.wheight.value;
  var options = "location=yes,toolbar=" +
         yesno(t) +
         ",menubar=" + yesno(m) +
         ",scrollbars=" + yesno(s) +
        ",statusbar=" + yesno(u) +
        ",resizable=" + yesno(r) +
        ",directories=" + yesno(d) +
        ",width=" + w +
        ",height=" + h;

  var url = "window-demo.php";
  var myname = "mywindow";
  document.getElementById("storage").innerHTML = options;
  var win = window.open(url, myname, options);
  win.document.write("Hello!");
}
```

SetTimeout et setInterval: Les délais en JavaScript :

On peut en JavaScript déclencher des action après un intervalle de temps donné, ou de la répéter après un intervalle de temps.

Les méthodes *setTimeout* et *setInterval* sont des méthodes de l'objet Window. On peut donc écrire *setTimeout* ou *window.setTimeout*.
Ce sont des processus indépendants qui, quand ils sont lancés par une instruction, ne bloquent pas l'affichage du reste de la page ni les actions de l'utilisateur.

SetTimeout indique un délai avant exécution :

Le méthode setTimeout définit une action à exécuter et un délai avant son exécution. Elle retourne un identifieur pour le processus.

```
var x = setTimeout("instructions", délai en millisecondes).
```

Exemple:

```
function mafonction() { ...


setTimeout(mafonction, 5000);
```

La fonction sera exécutée après 5 secondes (5 000 millisecondes).

Le délai commence à l'instant où setTimeout est exécutée par l'interpréteur JavaScript, donc à partir du chargement de la page si l'instruction est dans un script exécuté au chargement, où à partir d'une action de l'utilisateur si le script est déclenché par celui-ci.

ClearTimeout interrompt le décompte de setTimeout

Cette méthode interrompt le délai et l'exécution du code associé à ce délai. Le processus à supprimer est reconnu par l'identifieur retourné par *setTimeout*.

Syntaxe:

```
clearTimeout(identifieur);
```

Exemple:

```
var x = setTimeout(mafonction, 5000);
```

```
...
clearTimeout(x);
```

setInterval déclenche une opération à intervalles réguliers

Similaire à *setTimeout*, elle déclenche répétitivement la même action à intervalles réguliers.

```
setInterval("instructions", délai)
```

Exemple:

```
setInterval("alert(('bip')", 10000);
```

Affiche un message toutes les dix secondes.
L'action sera recommencée jusqu'à ce que l'on quitte la page ou que *clearInterval* soit exécutée.

clearInterval stoppe l'action de setInterval

Stoppe le processus déclenché par setInterval.

Exemple:

```
var x = setInterval(mafonction, 10000);
...
clearInterval(x);
```

Une fonction lambda peut être un argument

On peut définir une fonction dans les arguments de *setTimeout* ou *setInterval*.

Exemple:

```
var x = setTimeout(function() { alert("bip"); }, 2000);
```

L'intérêt est surtout d'utiliser une fonction récursive ce que l'on peut faire en désignant la fonction par la variable: *arguments.callee*.

Démonstration

Nous allons utiliser les deux fonctions en même temps:

- *setTimeout* pour déclencher une action dans 20 secondes.

- *setInterval* pour afficher le décompte des secondes.

La fonction *action* après 20 secondes affiche le message "TERMINE" et stoppe le décompte par un appel à *clearInterval*.

Cliquer sur le bouton pour commencer:

Code source:

```
<button onclick="start()">Lancer le décompte</button>
<div id="bip" class="display"></div>

<script>
var counter = 20;
var intervalId = null;
function action()
{
  clearInterval(intervalId);
  document.getElementById("bip").innerHTML = "TERMINE!";
}
function bip()
{
  document.getElementById("bip").innerHTML = counter + " secondes restantes.";
  counter--;
}
function start()
{
  intervalId = setInterval(bip, 1000);
  setTimeout(action, 20000);
}
</script>
```

Navigator en JavaScript, pour identifier le navigateur de l'internaute :

C'est un objet du navigateur disponible depuis la version 1.0 de JavaScript, mais certaines propriétés et méthodes sont apparues ultérieurement.

Les propriétés de navigator

Propriétés standards

Elles sont reconnues par tous les navigateurs récents. Prendre garde aux majuscules.

appName

Le nom générique de classe du navigateur. *Netscape* pour Firefox.

appCodeName

Nom du navigateur.

appVersion

Plateforme (*windows*, etc.) et version du navigateur.

userAgent

Chaîne de caractères envoyée au serveur sur lequel on lit une page.

platform

Code de système d'exploitation, par exemple *win32*.

Autres propriétés

Elle sont reconnues par Internet Explorer seul ou IE et Opera. Leur intérêt est donc très minime. Ce sont:

- userLangage: (IE et Opera). Code de langue du système d'exploitation: fr, en...
- appMinorVersion: Numéro de sous-version.
- browserLanguage: Code de langue du navigateur: fr, en.
- systemLanguage: Code de langue par défaut du système d'exploitation: fr, en.
- cpuClass: Type de système, par exemple x86 pour les PC et Mac récents.
- onLine: Navigateur en ligne ou non (il ne s'agit pas de la page).

La propriété mimeTypes est reconnue par Mozilla uniquement et retourne dans un tableau les types mime supportés par le navigateur.

Les méthodes de navigator

javaEnabled()

Retourne true si java est activé et si les applets peuvent fonctionner, false autrement.

taintEnabled()

Retourne vrai si l'option est activée auquel cas les script peuvent délivrer des informations sur le système avec un risque de sécurité.

Obtenir le numéro de version du navigateur

Le numéro de version commercial est dans *appVersion* sur certain navigateur, mais on utilisera plutôt *userAgent* qui le contient dans tous.

Il est dans la source de la démonstration qui suit. Ce n'est pas le numéro de version principal du navigateur, celui-ci suit la chaîne Firefox ou MERISE ou autre.

Il peut être extrait de *userAgent* en fonction du nom du navigateur, avec la méthode *indexOf* de l'objet *String*. On utilisera ces chaînes pour identifier le navigateur et obtenir le numéro:

Firefox/3.0.7
MSIE 7.0
Chrome/1.0.154.48
Opera/9.64
Version/4.0 Safari/528.16

Le script:

```
var ua = navigator.userAgent;


var x = ua.indexOf("MSIE");
var y = 4;
if (x == -1)
{
        x = ua.indexOf("Firefox");
        y = 7;
        if(x == -1)
        {
                if(x == -1)
                {
                        x = ua.indexOf("Chrome");
                        y = 6;
                        if(x == -1)
                        {
                                x = ua.indexOf("Opera");
                                y = 5;
                                if(x == -1)
```

```
                                        {
                                                x = ua.indexOf("Safari");
                                                if( x != -1)
                                                {
                                                        x = ua.indexOf("Version");
                                                        y = 7;
                                                }
                                        }
                                }
                        }
                }
        }

        if(x != -1)
        {
                y ++;
                ua = ua.substring(x + y);
                x = ua.indexOf(" ");
                var x2 = ua.indexOf("(");
                if(x2 > 0 && x2 < x) x = x2;
                x2 = ua.indexOf(";");
                if(x2 > 0 && x2 < x) x = x2;
                if (x == -1) document.write("Error");

                var v = ua.substring(0, x);
                document.write("Version: " + v);
        }
```

Ce script est contenu dans la démo et peut être réutilisé dans toute page Web.

Doivent fonctionner sur tous les navigateurs:

navigator.appName: **Netscape**

navigator.appCodeName: **Mozilla**

navigator.appVersion: **5.0 (Windows)**

navigator.platform: **Win32**

navigator.cookieEnabled: **true**

navigator.userAgent: **Mozilla/5.0 (Windows NT 6.1; rv:9.0.1) Gecko/20100101 Firefox/9.0.1**

navigator.javaEnabled(): **false**

Aide Mémoire JavaScript

Types de variables

var i = 123; nombre entier

var f=0.2 nombre réel

var t="texte" chaîne de caractères

var a=[1, "deux", 'trois'] tableau

var m={1:"un", "deux":2} tableau associatif

function x() { } objet

Gestionnaires d'évènements

onAbort chargement interrompu

onBlur perte du focus

onChange modification d'un état ou contenu

onClick clic sur un élément

onDblClick double clic

onDragDrop déplacement d'un élément amovible

onErrorchargement non réalisé

onFocus élement devient accessible

onKeyDown touche du clavier maintenue appuyée

onKeyPress touche pressée/relâchée

onKeyUp touche relâchée

onLoad au chargement

onMouseDown appui sur un bouton de souris

onMouseMove souris déplacée

onMouseOut souris hors de l'élément

onMouseOver souris au dessus

onMouseUp bouton de souris relaché

onReset bouton reset de formulaire

onResize dimensions changées dynamiquement

onSelect sélection d'une partie de contenu

onSubmit bouton soumettre

onUnload fermeture de la page

Méthodes d'objet (héritées par tous les objets)

toString() sous forme de chaîne
toLocaleString() chaîne localisée
valueOf() sous forme de valeur

Méthodes de Date

new Date() constructeur, arguments: millisecondes, chaîne, liste

getDate() jour du mois

getDay() jour de la semaine

getTime() nombre de millisecondes depuis le 1/1/1970

getYear() et getMonth/Hour/Minutes/Seconds

Méthodes de String

charAt() caractère en position donnée

charCodeAt() code d'un caractère

concat() concatène avec

indexOf() position d'un caractère

lastIndexOf() position à partir de la fin

localeCompare() comparaison localisée

matchapplique une expression régulière()

replace() remplace une partie

search() recherche une chaîne

slice() extrait une partie

split() découpe

substring() extrait une partie

toLowerCase() en minuscules

toUpperCase() en majuscules

toLocaleLowerCase() localisé

toLocaleUpperCaselocalisé()

Array, indices et méthodes

a["un"]=1 assignement par indice

a.un=1 assignement par attribut

delete a["un"] suppression par indice

delete a.un suppression par attribut

for(var k in a) {} itération sur le contenu

concat() ajoute un second tableau

join() concatène les éléments dans une chaîne

push() ajoute un élément

pop() récupère et retire le dernier élément

reverse() inverse l'ordre des éléments

shift() insère un élément au début

slice() extrait un sous-tableau

spliceinsère un sous-tableau ()

sort() classe les éléments

toString() retourne le tableau sous forme de chaîne

unshift() Récupère et supprime le premier élément

Méthodes de Number

new Number() constructeur avec un argument décimal/hexa/chaîne

toString() conversion en chaîne

toExponential() forme exponentielle

toPrecision() convertit avec un nombre de décimales donné

Fonction (Est aussi un objet)

function x(a, b) { return y; } déclaration

y = x(1, "deux") appel

var y = new x(1, "deux") déclarer une instance

x.prototype.methodx =

function() { } ajouter une méthode

Fonctions prédéfinies

eval() évaluation d'une expression

parseInt() code d'un caractère

parseFloat() convertit une chaîne en nombre flottant

isNaN() teste contenu variable non valide

ifFinite() teste dépassement capacité

decodeURI() conversion en texte

decodeURIComponent()décode un composant d'URL

encodeURI() conversion en nom de fichier

encodeURIComponent()encode un composant d'URL

escape() conversion en paramètres d'URL

unescape() conversion de paramètres en texte

Expression régulière, suffixes

g global
i insensible à la casse
s simple ligne
m multi-lignes

Expression régulière, masques

^ début de chaîne
$ fin de chaîne
(...) groupement
!() négation du groupe
. tout caractère
(x|y) soit x, soit y
[xyz] parmi x y ou z
[^xyz] tout sauf x y ou z
a? peut contenir a une fois
a+ au moins une fois a
a* zéro ou plusieurs a
a{5} cinq fois a
a{5,} cinq fois a au moins
a{1, 4} a entre 1 et 4 fois

Barre de navigation en JavaScript :

Les internautes n'apprécient pas vraiment que les pages soient divisées en plusieurs parties. Ce n'est pas non plus une bonne chose pour obtenir des backlinks. Cependant, cela peut quelquefois être nécessaire quand les pages sont trop longues, pour économiser la bande passante.

Nous allons donc montrer comment réaliser une barre de navigation, qui se présentera sous la forme suivante.

Page Première 1 2 3 4 5 >> Dernière

On peut bien sûr varier la présentation avec une feuille de style.

On veut que la barre de navigation présent en grisé le lien (ou les liens) sur la page courante et en bleu les liens des autres pages.
Cela ne peut pas se faire avec l'attribut "a:active", car la couleur pour les pages visitées surclasse la couleur de la page active.
Il faudra donc utiliser CSS et JavaScript pour obtenir ce résultat.

Le code HTML de la barre de navigation

```
<p>Page
<span id="navbar">
<a class="navpage" href="page1.html" target="_top">Premi&egrave;re</a>
<a  class="navpage" href="page1.html" target="_top">1</a>
<a  class="navpage" href="page2.html" target="_top">2</a>
<a  class="navpage" href="page2.html" target="_top">>></a>
<a  class="navpage" href="page2.html" target="_top">Derni&egrave;re</a>
</span>
</p>
```

On aura besoin de l'identificateur "navbar" pour localiser les modification de style en JavaScript, et de la classe "navpage" pour modifier individuellement le style de chaque lien.

La feuille de style

```
#navbar {  border:1px solid gray;  padding:2px; }
.navpage {   color:blue; }
.graystyle {   color:gray; }
```

Cette définition permettra de modifier la couleur des liens.

Le code JavaScript

```
function pagebar()
{
  var links=document.getElementById('navbar').getElementsByTagName("a");
  var current = location.href;
  for (var i=0; i < links.length; i++)
  {
   if(links[i].href == current)
   {
     links[i].href = "";
     links[i].className='grayStyle';
   }
  }
}
```

Le script construit la liste de tous les liens contenus dans la cadre "navbar". Il compare l'URL du lien avec celui de la page courante, en location.href, et lorsque il est identique, lui attribue le descripteur "grayStyle".

Attribuer des points à un article :

L'idée

Cela est utilisé notamment sur les sites d'information collaboratifs, on clique sur une image pour indiquer que l'on apprécie l'article, et les articles qui obtiennent le plus de points vont en première page du site.
On va voir comment réaliser ce système de notation avec quelques fonctions JavaScript et une feuille de style.
Pour cela on utilisera un fichier qui enregistrera le nombre de point obtenus par l'article, portant le nom de la page avec l'extension .dat.

La feuille de style

Elle définit l'image qui contient le nombre de points:

```
.markstyle
{
	font-family: "Trebuchet MS", Arial, sans;
	font-size: 40px;
	border-top:2px solid #0C0;
	border-left:2px solid #0C0;
	border-right:2px solid #060;
	border-bottom:2px solid #060;
	background-color:#0B0;
	text-align:center;
	width:64px;
	color:#FFF;
	padding-left:8px;
	padding-right:8px;
}
```

On peut facilement modifier les couleurs, la police de caractère, la taille...

Le programme JavaScript

Quelque fonction sont nécessaires pour récupérer le nombre de point dans un fichier et le sauver, ainsi que pour afficher ce nombre.

```
function read(url, fun, element)
```

Fonction Ajax appelée à chaque chargement de la page qui contient l'article à noter, avec le nom du compteur, la fonction initialize en callback et l'élément de la page pour l'affichage.

```
function writeFile(url)
```

La fonction writeFile complète la bibliothèque Ajax et permet de passer des données à un script PHP qui va stocker le nombre de points et autres données éventuelles.

```
function initialize(valeur, element)
```

C'est un callback, une fonction donnée en argument d'une fonction Ajax qui lit le fichier de données. Dans cette démo il ne contient que le nombre de points pour cette page.

```
function display(entier, element)
```

Cette fonction est appelée par la fonction *mark* et par le callback *initialize* pour afficher le score. Le nombre de points est limité à 99 dans cette démo.

```
function mark(element)
```

La fonction principale qui est appelée quand on clique sur l'image. Elle incrémente le nombre de votes affiché et utilise Ajax donner le nom de la page à un script qui conserve le nombre de vote dans un fichier. Elle appelle aussi la fonction *display*.

```
window.onload=function()
{
	dataURL = changeExtension(location.href, ".dat");
	var element = document.getElementById("mark");
	read(dataURL, initialize, element);
}
```

Quand on charge la page, il faut que l'image s'affiche avec le nombre de points obtenus jusque là. Ce sera accomplit par l'évènement onload de window, auquel on associe la fonction ci-dessus qui utilise Ajax pour lire le nombre de points conservé dans un fichier.

DataURL est le nom de ce fichier, il est fabriqué à partir du nom de la page qui appelle la fonction.

Empêcher les votes multiples

Pour empêcher un même lecteur d'ajouter plus d'un point à l'article, on peut:

1. Tester son adresse IP, l'ajouter dans une liste et ignorer les clics venant d'une même addresse IP.
   Mais l'inconvénent est que des adresses IP dynamiques sont utilisées par différents utilisateurs.
2. Installer des cookies sur les machines des utilisateurs.
   Ici l'inconvénient est qu'il en faudrait pour chaque article et chaque visiteur qui aurait cliqué, on n'en sortirait pas.
3. Demander aux utilisateurs de s'enregistrer avant de marquer les articles. Cela peut requérir aussi, pour empêcher les enregistrements multiples, d'installer des cookies, mais un seul par utilisateur enregistré. On peut ensuite gérer la liste des clics pour chaque utilisateur.

Le script d'authentification des utilisateurs va au-dela du sujet de cet article, mais il est développé par ailleurs dans le tutoriel *Construire un CMS*.

Bookmarklet : Ajouter des fonctions aux pages :

Un bookmarklet ou favelet est un lien qui contient du code JavaScript assigné à l'attribut *href*. L'intérêt est d'incorporer des fonctions à une page Web sous forme de lien à cliquer, grâce à l'exécution d'un script.
Mais il est le plus souvent incorporé dans les bookmarks - d'où son nom - pour ajouter des fonctionnalités au navigateur.
Les exemples fournis dans la démonstration montreront ce que l'on peut faire avec les bookmarklets.

Les bookmarklets existent dans le navigateur Netscape depuis 1995 (qui est remplacé maintenant par Firefox).

Format d'un bookmarklet

Le mot-clé *javascript* introduit du code dans l'attribut d'une balise HTML, comme ci-dessous:

```
<a href='javascript:alert("hello");'> Ma fonction </a>
```

Le code est placé entre guillemets simples, ce qui permet d'intégrer des guillemets doubles dans les instructions. On peut faire l'inverse.
Tout ce qui suit **javascript:** à l'intérieur d'une chaîne de caractères est traité par le navigateur comme du code.

Le code peut être constitué de plusieurs instructions et être assez complexe pourvu qu'il soit formé d'une seule ligne. Il a accès à l'arborescence DOM de la page et donc à tous les éléments de celle-ci.

Exemples d'applications

On peut ainsi ajouter des fonctions pour:

- Modifier l'apparence d'une page. Les couleurs, les polices...
- Obtenir des informations sur la page. Par exemple le nombre de liens comme cela est montré dans la démo.
- Accéder à des sites externes en leur passant des paramètres.
  Pour cela on ouvre une boite de dialogue (commande prompt) dont le résultat est assigné à une variable, laquelle est passée en paramètre à l'URL du site.
- Placer sur son site des boutons de sites sociaux comme Digg, Delicious, afin que les visiteurs puissent voter pour l'article.

Rappel sur la fonction prompt

Elle ouvre une boite de dialogue comportant un champ d'entrée de texte et retourne le texte tapé par l'utilisateur, ou *false* si rien n'est entré.

L'interface de la fonction comporte deux paramètres:
- Le libellé affiché dans la boite.
- Une valeur par défaut pour le champ d'entrée de texte.

```
var x = prompt("Taper quelque chose:", "valeur par défaut");
```

On peut utiliser la variable de la façon suivante:

```
location.href='http://www.xul.fr?uri='+escape(x)
```

location.href permet de charger une autre page.
On ajoute un paramètre avec le symbole, la fonction escape permettant de convertir le texte en format valide pour une URL.

Bookmarklets: Informations sur la page :

Tout savoir sur la page web affichée avec une liste de fonctions que l'on va créer sous forme de bookmarklets, des liens contenant un code JavaScript que l'on place en bookmarks.

Ces codes ne fonctionnent pas si la page est dans une frame, ce qui par ailleurs devient rare.

Tous navigateurs

Nombre de liens

Utilise la propriété *link* de document.

```
javascript:alert(document.links.length+' liens')
```

Images sans attribut *alt*

Autre outil de design, ce bookmarklet fait apparaître les attributs alt manquants.

Pour les besoins de la démonstration, on incorpore deux images dans la page... La première sans attribut *alt* doit afficher *null*, mais la seconde en a un qui décrit l'image ainsi: "La plage du Prado".

```
javascript:{
var x='';
for(i=0;i<document.images.length;++i)
{
   x+=document.images[i].getAttribute('src') + "=";
   x+=document.images[i].getAttribute('alt')+  '\n';
};
alert(x);
}
```

Origine de la page

Affiche la page à partir de laquelle a été chargée la page courante, ou affiche *null* quand l'URL a été tapée directement dans la barre d'adresse.

```
javascript:alert('Source: ' + document.referrer)
```

Firefox et Internet Explorer

Date de dernière modification

Utilise l'attribut *lastModified* de document. Ce n'est pas un attribut standard et il ne fonctionne pas avec tous les navigateurs. Il est implémenté dans Firefox et Internet Explorer.

Code du bookmarklet:

```
javascript:alert('Last modified: '+document.lastModified);
```

- Date de dernière modification.

Le même code peut aussi être placé dans le corps de la page pour indiquer aux visiteurs quand elle a été mise à jour:

```
<script type="text/javascript"> document.write("Dernière modification " + document.lastModified); </script>
```

Firefox

Décompte des mots dans une sélection

Compte les mots dans un texte que vous avez sélectionné sur la page.
Ne peut fonctionner sur la page entière, car la lecture du contenu de la page retourne du code HTML.

Utilise la méthode *getSelection()* de l'objet *document* qui n'est pas standard et peut ne pas fonctionner avec tous les navigateurs. Ne fonctionne pas avec Internet Explorer mais comme c'est un outil de développement, il est surtout important qu'il soit reconnu par Firefox.

Bookmarklets: Fonctions de recherche :

Comme on l'a vu dans l'article Bookmarklet, cette fonctionnalité de HTML et JavaScript, supportée depuis l'origine par les navigateurs, peut offrir des possibilités étendues.

Nous allons voir comment ajouter des fonctions de recherche variées au navigateur: pour cela il suffira d'ajouter les bookmarklets sous forme de liens dans la barre latérale de favoris.

Recherche simple

Ouvre une boite de dialogue pour faire une recherche sur Google. Les résultats sont dans une nouvelle page.

```
javascript:(function()
{
  var p=prompt('Recherche Google:','');
  if(p)
```

```
  {
    document.location.href='http://www.google.com/search?query='+escape(p)
  }
})();
```

- Recherche Google
- Recherche Yahoo

Recherche sur le site

Effectuer une recherche limitée au site web auquel appartient la page courante.

```
javascript:(function()
{
  var p=prompt('Recherche Google:','');
  if(p)
  {
document.location.href='http://www.google.com/search?q=site:'+document.location.href.split('/')[2]+'+'+escape(p
)
  }
})();
```

- Recherche Google sur le site
- Recherche Yahoo sur le site

Pages similaires

Recherche des pages similaires à la page courante, service fourni par Google.

```
javascript:(function()
{
    document.location.href='http://www.google.com/search?q=related:'+escape(document.location.href);
})();
```

- Pages similaires avec Google

Images libres de droit

Trouver des images dans le domaine public avec Google Images.

```
javascript:(
```

```
 function()
 {
  p=prompt('Recherche%20Images:','');
  if(p)
  {
   document.location.href='http://images.google.com/images?query='+escape(p)+'&as_rights=cc_publicdomain';
  };
 }
)();
```

- Images libres de droits.

Ajouter le bookmarklet dans les favoris

Pour obtenir le code des bookmarklets, charger la page avec un éditeur HTML, Open Office par exemple, cliquer sur le lien et reprendre le contenu de l'attribut href.

Ensuite aller dans le panel de favoris du navigateur, choisir un dossier et cliquer le bouton droit de la souris pour faire apparaître le menu. Cliquer sur "Nouveau bookmark", placer le code dans le champ URL et choisir un nom pour le bookmarklet.

Bouton à deux états et définitions dans le texte :

L'idée

Nous utilisons un bouton pour passer une commande à une page Web, et nous voulons changer le bouton pour passer à l'état opposé quand la commande est exécutée, pour pouvoir envoyer la commande contraire.
Les exemples que nous utilisons sont les commandes "résumer" et "développer" permettant de réduire ou étendre le contenu d'une page selon ce que demande le lecteur.
Les exemples requièrent la bibliothèque Ajax Extensible qui est fournie sur ce site, gratuite et open source.

Changer le bouton

Le code à insérer dans la page Web est très simple et ne requiert aucune programmation:

```
<div onclick="switchButton(this);">
  <img src="developper.gif" />
</div>
```

C'est une balise standard pour insérer un bouton avec un attribut pour appeler une fonction.
Il faut aussi inclure la fonction JavaScript dans la section head de la page:

```
<script src="oneButton.js" type="text/JavaScript"></script>
```

Rien d'autre n'est requis pour créer un bouton à deux états.

Démonstration

C'est une démo écrite en JavaScript pour montrer un bouton a deux états.
C'est un framework Ajax permettant de contruire des pages extensibles.

Code source de la page

```
<style>
.aep { color: #009;     }
</style>

<script src="extensible.js" type="text/javascript"></script>
<script src="oneButton.js" type="text/javascript"></script>

<div onclick="switchButton(this);">
   <img src="develop.gif" width="62" height="16" align="middle">
</div>

<div id="x2" style="display:none;">un langage de script pour le web</div>

Une démo écrite en <span id="aep" name="x2" class="aep" onclick="quietVerbose(this);" >JavaScript</span>
 pour montrer un bouton a deux états.
C'est un  <span id="aep" src="definition.txt" class="aep" onclick="quietVerbose(this);" > framework Ajax</span>
 permettant de contruire des pages extensibles.
```

Code du script oneButton.js

```
var buttonDevelop = "developper.gif";
var buttonSummarize = "resumer.gif";
function switchButton(element)
{
  var d = buttonDevelop.length;
  var s = buttonSummarize.length;
  var image = element.firstChild;
  var name = image.getAttribute("src");
  var iname = name.substr(name.length - d, d);

  if(iname == buttonDevelop)
```

```
  {
   image.setAttribute("src", buttonSummarize);
   develop();
   return;
  }
  iname = name.substr(name.length - s, s);
  if(iname == buttonSummarize)
  {
   image.setAttribute("src", buttonDevelop);
   summarize();
   return;
  }
}
```

L'image mystérieuse :

Ce script en JavaScript fait apparaître les images dans une page uniquement lorsque l'utilisateur le demande.

On peut l'utiliser dans plusieurs cas:

1. Dans un quizz, lorsque l'image est la réponse à une question posée au lecteur.
2. Lorsque l'on veut ménager un effet visuel où l'image vient ponctuer une démonstration, mais ne doit être présentée qu'à la fin de cette démonstration.
3. Lorsqu'on veut superposer un texte et une image, le texte apparaissant avant l'image car étant plus lisible sans l'image.

Comment l'application fonctionne

L'image est en fait affichée en même temps que la page, pour éviter un délai de réponse, mais les propriétés CSS de la zone qui la contient ont été définies pour la rendre transparente.

On commande l'apparition en fondu avec l'évènement *onClick*, ou mieux encore avec *onOver*. Dans le second cas il suffit de passer la souris sur l'image pour qu'elle apparaisse. Le script qui la fait apparaître utilise une fonction de fondu pour faire passer la zone d'un niveau d'opacité zéro au niveau total.

Réalisation pas à pas

Superposition du texte et de l'image

Pour remplacer progressivement un texte par une image, on les place dans des calques superposés grâce aux propriétés CSS.
On place les deux calques dans un troisième, le conteneur, afin de réserver un espace sur la page.

Les calques

```
<div class="mback" style="width:500px;height:375px" onClick="showMysteryPicture('mbox')">
   <span class="mtext" style="width:500px;height:375px" >?</span>
   <span id="mbox" class="mbox">
            <img src="images/voyage03.jpg" width="500" height="375">
   </span>
</div>
```

mback est le conteneur.
mtext contient un point d'interrogation que ses propriétés CSS rendront géant.
mbox contient une image transparente au départ.

Il faut préciser la taille du conteneur et celle du texte. On le fait dans la page plutôt que dans le fichier mystery.css afin de pouvoir utiliser le script avec des images de taille différentes.

Leurs propriétés CSS

```
.mback
{
   position:relative;
}


.mtext
{
   display: block;
   position: absolute;
   background-color: #CCA;
   z-index:10;
   font-size:600%;
   text-align:center;
}
```

```
.mbox
{
   display:none;
   position:absolute;
   z-index:11;
   opacity:0.0;
   filter: alpha(opacity=0);
}
```

Le calque de fond à une position relative, mais les calques internes ont une position absolue à l'intérieur du calque de fond. Aucune position n'est donnée aux calques internes. C'est indispensable pour la superposition et cela fonctionne avec tous les navigateurs.

Le texte à une taille démultipliée pour afficher un point d'interrogation géant.

L'image à une opacité de 0 avec un filtre pour Internet Explorer. Donc elle est transparente au départ et l'on ne voit donc que le point d'interrogation. Pour la faire apparaître, on utilisera un script.

Lorsque l'image apparaîtra, le texte disparaîtra, cela est obtenu par les propriétés *z-index*: l'image avec un index de 11 est placée au-dessus du texte qui a un index de 10.

Apparition de l'image

Avec un timer, on modifie progressivement l'opacité de l'image pour la faire passer de 0 (transparence) à 255 (opacité totale).

Le script

```
function fadein(id)
{
   var level = 0;
   while(level <= 1)
   {
      setTimeout( "gradient('" + id + "'," + level + ")", (level* 1000) + 10);
      level += 0.01;
   }
}

function showMysteryPicture()
{
   fadein("mbox");
```

}

La fonction *showMysteryPicture()* qui est associée à l'évènement *onClick* de *mbox*, appelle *fadein()* avec l'identifieur du calque contenant l'image, et *fadein* fait passer la variable *level* de 0 à 1 pour faire évoluer l'opacité du calque.

La démonstration est fournie dans une archive qui contient tous les sources nécessaires.

Mise en œuvre d'Ajax :

Véritable clé de voûte des applications Web riches, Ajax (Asynchronous JavaScript And XML) permet d'exécuter des requêtes HTTP sans pour autant avoir à recharger la page qui l'exécute.

Ajax se fonde essentiellement sur la classe XMLHttpRequest, en cours de normalisation auprès du W3C. Cette dernière fournit un cadre a�n d'implémenter des requêtes Ajax de manière aussi bien synchrone qu'asynchrone. Nous détaillons dans ce chapitre la façon d'utiliser cette classe.

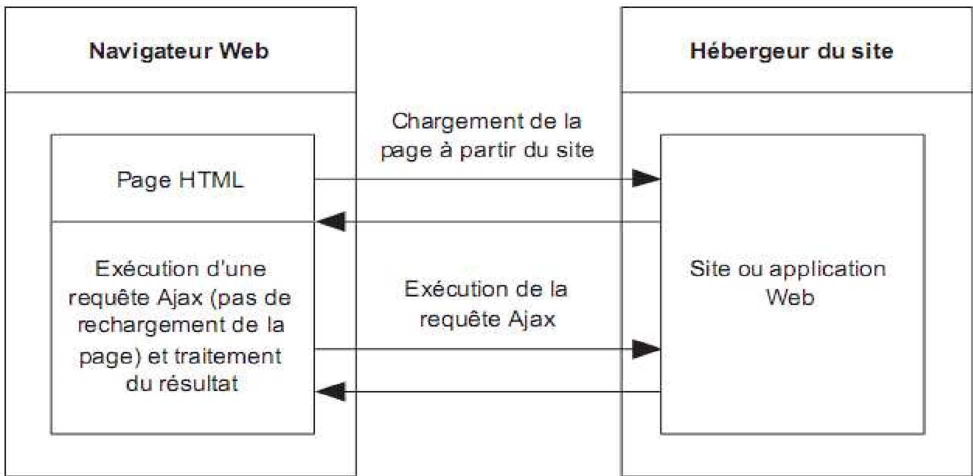

Figure 5.1
Fonctionnement général d'Ajax

La classe XMLHttpRequest :

L'entité de base permettant de mettre en œuvre Ajax dans un navigateur Web est la classe JavaScript XMLHttpRequest.

Attributs de la classe XMLHttpRequest :

onreadystatechange : Permet de positionner une fonction de rappel, qui est appelée lorsque l'événement readystatechange se produit. Cet événement est suppor té en mode asynchrone et correspond à un changement d'état dans le traitement de la requête.

readyState : Code correspondant à l'état dans lequel se trouve l'objet tout au long du traitement d'une requête et de sa réponse.

responseText : Réponse reçue sous forme de texte

responseXML : Réponse reçue sous forme XML

status : Correspond au code du statut de retour HTTP de la réponse. Il n'est disponible en lecture que lorsquela réponse à la requête a été reçue.

statusText : Correspond au texte du statut de retour HTTP de la réponse. Il n'est disponible en lecture que lorsque la réponse à la requête a été reçue.

Méthodes de la classe XMLHttpRequest :

abort () : Annule la requête courante et réinitialise l'objet.

getAllResponseHeaders () : Retourne tous les en-têtes HTTP contenus dans la réponse dans une chaîne de caractères. Cette méthode ne peut être utilisée que lorsque la valeur de la propriété readyState de l'objet équivaut à 3 ou 4 (état Receiving ou Loaded).

getResponseHeader (Le nom de l'en-tête) : Retourne la valeur de l'en-tête dont le nom est spéci􏰀é en paramètre. Peut-être utilisée dans le même cas que la méthode précédente.

open (La méthode HTTP à utiliser et l'adresse de la ressource à accéder. De manière optionnelle, le type de fonctionnement ainsi que l'identi􏰀ant de l'utilisateur et son mot de passe.) : Initialise l'objet XMLHttpRequest pour une requête par ticulière. Elle permet de spéci􏰀er le type de requête HTTP à utiliser (GET ou POST) et l'adresse de la ressource à accéder. Le type de fonctionnement détermine la manière dont est traitée la réponse à la requête (de manière synchrone ou asynchrone). S'il n'est pas spéci􏰀é, le mode asynchrone est utilisé. En􏰀n, des informations de sécurité (identi􏰀ant d'utilisateur ainsi que mot de passe) peuvent être spéci􏰀ées.

send (Une chaîne de caractères ou un document XML) : Permet d'envoyer une requête à l'adresse spéci￿ée avec la méthode HTTP souhaitée et éventuellement des paramètres d'authenti￿cation. Si le mode de réception est synchrone, la méthode est bloquante jusqu'à ce moment. Cette méthode ne peut être utilisée que lorsque la valeur de la propriété readyState de l'objet équivaut à 1 (état Open).

setRequestHeader (Le nom de l'en-tête et sa valeur) : Positionne un en-tête HTTP pour la requête. Cette méthode ne peut être utilisée que lorsque la valeur de la propriété readyState de l'objet équivaut à 1 (état Open).

Gestion des échanges :

Contrairement à ce que pourrait laisser penser son nom, *Ajax(Asynchronous JavaScript And XML)* ne gère pas nécessairement les réponses à des requêtes de manière asynchrone, puisque aussi bien les modes synchrone et asynchrone sont supportés. La sélection du mode de gestion des requêtes, synchrone ou asynchrone, se réalise au moment de l'appel de la méthode open, par l'intermédiaire de son troisième paramètre.

La sélection du mode de gestion des requêtes, synchrone ou asynchrone, se réalise au moment de l'appel de la méthode open, par l'intermédiaire de son troisième paramètre.

Dans le cas d'une mise en œuvre synchrone, la réception de la réponse se réalise séquentiellement, comme dans le code suivant :

```
var transport = getXMLHttpRequest();

transport.open("get", "/ajax/data.js", false);

transport.send(null);

var texte = transport.responseText;

(...)
```

En cas de mise en œuvre asynchrone, la réception se réalise par l'intermédiaire d'une fonction de rappel. Cette dernière est spéci￿ée en utilisant l'attribut onreadystatechange de la classe XMLHttpRequest.

États de l'instance de la classe XMLHttpRequest en mode asynchrone :

0 [Uninitialized] : État initial

1 [Open] : La méthode open a été exécutée avec succès.

2 [Sent] : La requête a été correctement envoyée, mais aucune donnée n'a encore été reçue.

3 [Receiving] : Des données sont en cours de réception.

4 [Loaded] : Le traitement de la requête est fini.

Chaque fois que l'instance change d'état, la fonction de rappel enregistrée est appelée. Cette dernière peut se fonder sur l'attribut readyState de cette instance afin de déterminer son état courant.

Le code suivant décrit la mise en œuvre d'une requête asynchrone simple :

```
var transport = getXMLHttpRequest();

function changementEtat() {

   if( transport.readyState == 4 ) {

      var texte = transport.responseText;

      (...)

   }

}

transport.onreadystatechange = changementEtat;

transport.open("get", "/ajax/data.js", true);

transport.send(null);
```

Notons que les données de la réponse ne sont accessibles par l'objet transport que lorsque

l'état de l'objet vaut 4 (Loaded).

Échanges de données :

Il existe deux façons de spécifier des données lors de l'envoi d'une requête.

La première consiste à positionner des paramètres directement dans l'adresse de la ressource du serveur. Cette technique se fonde sur la méthode GET du protocole HTTP. Le code suivant décrit la mise en œuvre de cette approche :

```
var transport = getXMLHttpRequest();
```

```
var parametre1 = "valeur1";

var parametre2 = "valeur2";

var adresse = [ "donnees.js" ];

adresse.push("?parametre1=");

adresse.push(parametre1);

adresse.push("&parametre2=");

adresse.push(parametre2);

transport.open("get", adresse.join(""), true);

transport.send(null);
```

La seconde façon consiste à envoyer un bloc de données et n'est réalisable qu'avec la méthode POST du protocole HTTP. La méthode send de la classe XMLHttpRequest permet de spéci er ce bloc par l'intermédiaire de son paramètre. Le code suivant décrit la mise en œuvre de cette approche :

```
var transport = getXMLHttpRequest();

var parametre1 = "valeur1";

var parametre2 = "valeur2";

var parametres = [ ];

parametres.push("{ parametre1: \"");

parametres.push(parametre1);

parametres.push("\", parametre2: \"");

parametres.push(parametre2);

parametres.push("\" }");

transport.open("post", "/ajax/data.js", true);

transport.send(parametres.join("\n"));

(...)
```

Dans ce dernier cas, les données peuvent être fournies en tant que texte ou document XML, car la méthode send accepte ces deux types de format de données.

Par contre, la récupération des données contenues dans la réponse de la requête se réalise d'une seule façon. Elle se fonde sur les attributs responseText et responseXML de l'instance de la classe XMLHttpRequest, pour une réponse respectivement au format texte et XML Ces attributs sont positionnés lorsque la réponse a été reçue (état Loaded). Le code suivant donne un exemple d'utilisation de ces attributs (repères ❶ et ❷).

```
var transport = getXMLHttpRequest();

(...)

//Dans le cas où la réponse est au format texte

var reponse = transport.reponseText; ¨ ❶

//Dans le cas où la réponse est au format XML

var reponseXML = transport.reponseXML; ¨ ❷
```

Les données reçues dans la réponse peuvent donc être manipulées aussi bien sous forme de texte que de document XML.

Structure des données échangées :

Format XML :

La classe XMLHttpRequest offre nativement un support pour l'utilisation de XML par le biais de sa méthode send et de son attribut responseXML, qui prennent et renvoient respectivement des documents XML. Cette approche structure les données tout en les rendant lisibles par les utilisateurs. Le code suivant utilise le format XML avec la classe XMLHttpRequest aussi bien au niveaude l'envoi (repère ❷) que de la réception (repère ❶).

```
var transport = getXMLHttpRequest();

function changementEtat() {

   if( transport.readyState == 4 ) {

       var donneesXML = transport.responseXML; ¨ ❶

       // donnees correspond à la balise de base de la reponse
```

```
        var donnees = donneesXML.childNodes[0];
        // Traitement des données XML reçues
        var differentesDonnees = donnees.childNodes;
        for(var cpt=0; cpt<differentesDonnees.length; cpt++) {
            var donnee = differentesDonnees[cpt];
            (...)
        }
    }
}
transport.onreadystatechange = changementEtat;
transport.open("get", "donnees.xml", true);
// Construction des données XML pour l'envoi
var parametres = document.createElement("parametres");
var parametre1 = document.createElement("parametre");
parametre1.setAttribute("nom","parametre1");
var texteParametre1 = document.createTextNode("valeur1");
parametre1.appendChild(texteParametre1);
parametres.appendChild(parametre1);
transport.send(parametres); ¨ ❷
```

Dans l'exemple ci-dessus, la structure XML envoyée est la suivante :

```
<parametres>
    <parametre nom="parametre1">valeur1</parametre>
</parametres>
```

La structure XML reçue peut être, par exemple, de la forme suivante :

```
<donnees>
   <donnee nom="donnee1">valeur1</donnee>
</donnees>
```

Format HTML :

Il est possible de recevoir directement des blocs de code HTML, notamment pour rafraîchir une partie d'une page Web. Cette méthode peut toutefois s'avérer complexe d'utilisation lors de l'incorporation des blocs dans une page, notamment pour la gestion des styles.

Le code suivant donne un exemple de mise en œuvre de ce format pour les échanges dans

une page HTML :

```
<html>
   <head>
      <script type="text/javascript">
         function getXMLHttpRequest() {
            (...)
         }
         function changementEtat() {
            if( transport.readyState == 4 ) {
               var html = transport.responseText;
               var div = document.getElementById(
                                 "conteneurReponse");
               div.innerHTML = html; ¨ ❶
            }
         }
         function executerRequete() { ¨ ❷
            var transport = getXMLHttpRequest();
```

```
            transport.onreadystatechange = changementEtat;
            transport.open("get", "donnees.html", true);
            transport.send(null);
        }
    </script>
</head>
<body>
    (...)
    <div id="conteneurReponse"></div> ¨ ❸
</body>
</html>
```

Cet exemple offre la possibilité, par l'intermédiaire de la fonction executerRequete (repère ❷ d'ajouter (repère ❶le contenu d'une réponse au format HTML dans la balise div d'identifiant conteneurReponse (repère ❸Les données HTML reçues sont alors les suivantes :

```
<p>Un paragraphe d'exemple</p>
```

Panel d'onglets en CSS le plus simple :

Il est possible de réaliser simplement une barre d'onglets pour une page Web, en CSS, avec quelques lignes de JavaScript optionnels, sans utiliser de framework Ajax.

Comment cela fonctionne

A chaque onglet on associe l'URL d'une page à charger. Les onglets sont basés sur des balises de liens, cliquer sur un onglet revient à cliquer sur un lien.

On peut utiliser les onglets sans aucun code JavaScript si chaque page possède le même panel d'onglets et remplace la page courante.
Mais si l'on veut charger les pages dans une iframe, sans quitter la page courante, du code JavaScript servira à lire l'URL de la page associée à un onglet et l'assigner à la propriété src de l'iframe.

Les deux cas seront vus dans la démonstration.

Le code CSS

L'utilisation des balises <li> et <a> nous facilite la tâche et rend le code plus clair.

La structure de la définition des onglets du Tab Panel est la suivante:

```
<div id="tabs">
   <ul>
      <li><a href="#" rel="url1">Libellé 1</a></li>
      <li><a href="#" rel="url2">Libellé 2</a></li>
   </ul>
</div>

<iframe id="container"></iframe>
```

Ce sera le rôle exclusif des descripteurs CSS de transformer la liste ci-dessus en une barre d'onglets horizontale.
La liste est encapsulée dans un <div> de sorte que le style donné aux balise <ul> et <li> soit local au <div> et ne s'applique pas à toute la page.

Onglets sans JavaScript

Le code JavaScript est utile pour deux choses:

- Modifier l'apparence de l'onglet actif.
- Charger une page dans l'iframe.

On peut se passer de JavaScript si la page chargée remplace la page courante. Se sera le cas si on utilise des onglets comme menu d'un site Web. Grâce à la construction de Tab Panel basée sur des balises <a>, le fait de cliquer sur un onglet revient à cliquer sur un lien.

Toutes les pages affichent le même menu d'onglets avec une différence: dans la présentation de chaque page, l'onglet qui lui correspond est mis en relief par l'assignation de la classe "selected" qui a un descripteur dans la feuille de style:

```
<li><a href="#" rel="url1" class="selected">Libellé 1</a></li>
```

Un exemple est donné dans la démo avec l'onglet No Script. La page correspondante à cet onglet fournit plus de détails.

Chargement des pages en iframe

Deux fonctions JavaScript suffisent.

A chaque onglet, on associe un événement *onclick* qui passe à la fonction *loadit* la source de l'appel.

```
<li><a href="#" rel="tab-dom.html" onClick="loadit(this)">DOM</a></li>
```

Fonction *loadit*: Charger une page correspondant à un onglet

```
  var container = document.getElementById('container');
  container.src=element.rel;

  var tabs=document.getElementById('tabs').getElementsByTagName("a");
  for (var i=0; i < tabs.length; i++)
  {
    if(tabs[i].rel == element.rel)
      tabs[i].className="selected";
    else
      tabs[i].className="";
  }
```

Pour modifier l'apparence de l'onglet actif on charge l'URL de toutes les balises <a> encapsulées dans "tabs" dans un tableau et l'on compare les URLs avec l'attribut *rel* de l'onglet sélectionné.
Pour la page choisie on associe la classe "selected" et aucune pour les autres.

Fonction *startit*: Au démarrage charger la page par défaut

Cette fonction est exécutée au chargement du Tab Panel pour remplir l'onglet affiché par défaut.

```
function startit()
{
  var tabs=document.getElementById('tabs').getElementsByTagName("a");
  var container = document.getElementById('container');
  container.src = tabs[0].rel;
}

window.onload=startit;
```

L'utilisation de onload permet de différer le chargement après que le DOM soit construit.

Compatibilité

Sous Internet Explorer, l'affichage est incorrect avec le DocType transitionnel et est correct avec ce DocType:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN">
```

Panel d'onglets avec frames :

Dans cette deuxième partie de notre , après l'utilisation de pages incorporant les onglets, l'utilisation d'iframes, voici la démonstration de l'utilisation de frames.
C'est en fait le cas le plus simple car dans ce cas les navigateurs gèrent le chargement des pages, il suffit d'associer le nom d'une frame en cible d'un, est les onglets contiennent des liens.

La différence pour les onglets avec les cas précédents est que nous n'utilisons plus d'attribut *rel* pour stocker l'URL de la page à charger, elle est stockée dans *href*.

Le code CSS est inchangé.

Structure des frames

```
<frameset rows="248,*">
  <frame src="onglets-menu.html"   />
  <frame src="tab-frame-css.html" name="mainFrame" id="mainFrame"  />
</frameset>
```

La première frame contient les onglets définis dans la page *onglets-menu.html*.
La seconde frame est celle où l'on charge les pages correspondants aux onglets

La page *tab-frame-css.html* est la page de l'onglet par défaut.

Structure des onglets

La définition des onglets est modifiée:

```
<div id="tabs">
<ul>
    <li><a href="tab-frame-css.html" class="selected" target="mainFrame"
           onclick="loadit(this)" >CSS</a></li>
    <li><a href="tab-frame-dom.html" target="mainFrame"
           onclick="loadit(this)" >DOM</a></li>
```

```
  <li><a href="tab-frame-javascript.html" target="mainFrame"
       onclick="loadit(this)" >JavaScript</a></li>
</ul>
</div>
```

L'attribut *rel* disparait mais on ajoute l'attribut *target* qui indique le nom de la frame où se charge la page.

Chargement des pages

Fonction *loadit* simplifiée

On n'a plus besoin d'assigner l'URL de la page, le code sert uniquement à modifier l'apparence des onglets.

```
var tabs=document.getElementById('tabs').getElementsByTagName("a");
for (var i=0; i < tabs.length; i++)
{
   if(tabs[i].href == element.href)
       tabs[i].className="selected";
   else
       tabs[i].className="";
}
```

Noter le remplacement de l'attribut *rel* par *href*.

La fonction d'initialisation pour charger la page par défaut est supprimée, la page par défaut est donnée dans le cadre de frames.

Ce code JavaScript simplifié est incorporé à la source de la page onglets-menu.html

Echanger des variables entre Javascript et PHP :

Comment les variables de Javascript peuvent-elles être utilisées par les scripts PHP?
Et inversement, comment utiliser les valeurs de variables PHP en Javascript ?

Les valeurs de formulaire seront dans $_POST ou $_GET :

Commençons par l'utilisation des éléments de formulaire en PHP. Le nom des éléments d'un formulaire sont également des variables de PHP dès lors que le script PHP est l'action du formulaire.

Exemple, ce formulaire complet:

```
<form  method="POST" name="myform" action="php-form-code.php">
   <input type="text" name="mytext" maxlength="80" size="30">
   <input type="submit" value="Submit" >
</form>
```

"mytext" est le nom donné à l'objet de saisie de texte dans le formulaire ci-dessus.
Le script PHP retrouve le nom comme indice du tableau $_POST:

```
<?php
 $mytext = $_POST['mytext'];
 echo $mytext;
?>
```

En fait, quand un fichier est chargé par la propriété action du formulaire, tous les objets de ce formulaire sont donnés comme paramètres au fichier de script, avec le format :

```
?nom=valeur&nom=valeur ...
```

Ces paramètres font partie de l'URL avec une requête GET, ils sont invisible avec POST.

Et ces paramètres deviennent des variables et des valeurs dans un script PHP.
Dans ce cas-ci :

```
?mytext="le texte entré"
```

La valeur est le texte que vous avez tapé au clavier.

Le type hidden *permet d'envoyer des données non fournies par l'utilisateur :*

Et si vous voulez envoyer au script sur le serveur quelques valeurs qui ne sont pas obtenues par le formulaire, comment faire?
Une solution simple est l'utilisation des éléments cachés, tel que celui-ci :

```
<input type="hidden" name="extra" value="Contenu de la variable supplémentaire" >
```

Ce type d'élément n'est pas affiché dans le formulaire, et n'a qu'une seule utilité, fournir une valeur qui sera ajouté aux autres valeurs fournies par la formulaire.

Si vous voulez changer dynamiquement la valeur par un script, vous pouvez assigner la valeur de la balise cachée, exemple :

```
document.myform.extra.value = "quelque valeur";
```

Les variables de PHP sont insérées dans une page HTML par *echo*

Cette page doit être parsée par l'interpréteur PHP, donc avoir l'extension PHP (a moins que le serveur ne soit configuré autrement).

Une fois que le code PHP est inclus, avec les instructions suivante:

```
<?php
  require("some-script.php";
?>
```

...n'importe quoi pourra être inséré dans la page avec la commande echo :

```
<?php
  echo "<p>Ce texte est affiché dans la page</p>";
?>
```

Ce texte est affiché dans la page

Vous pouvez assigner directement une valeur à une propriété d'un objet de cette manière.

```
<script>
var x = "<?php echo $x; ?>";
</script>
```

C'est une erreur commune d'inclure le code PHP dans celui de Javascript en s'attendant à ce que le code PHP utilise les résultats de Javascript. En fait PHP est traité par le serveur avant que la page soit chargée et le code de Javascript est traité après que la page ait été chargée. Il n'y a aucun moyen au code PHP d'employer directement les résultats du code de Javascript, on doit employer Ajax à la place.

Application :

Voici un code qui traite des fonctions (surface, resolution du second degre, image mysterieux (il faut mettre dans le repertoire du fichier html une image portant le nom w.jpg), horloge tounante(il faut mettre dans le même repertoire deux images de boules bleu.gif rt rouge.gif), le game de rebont (mettre une image de boule bille1.gif).

```
<html>

<title>FONCTION</title>
```

```
<head>

<style type="text/css">

   #chiffre12{position: absolute; top: 470px; left: 300px}

         #chiffre11{position: absolute; top: 500px; left: 230px}

         #chiffre10{position: absolute; top: 540px; left: 190px}

         #chiffre9 {position: absolute; top: 590px; left: 180px}

         #chiffre8 {position: absolute; top: 640px; left: 190px}

         #chiffre7 {position: absolute; top: 680px; left: 230px}

         #chiffre6 {position: absolute; top: 700px; left: 300px}

         #chiffre5 {position: absolute; top: 680px; left: 370px}

         #chiffre4 {position: absolute; top: 640px; left: 410px}

         #chiffre3 {position: absolute; top: 590px; left: 420px}

         #chiffre2 {position: absolute; top: 540px; left: 410px}

         #chiffre1 {position: absolute; top: 500px; left: 370px}

         #centre {position: absolute; top: 590px; left: 300px; font-family: arial}

         #aiguille_h {position: absolute; top: 480px; left: 300px}

         #aiguille_mn {position: absolute; top: 480px; left: 300px}


</style>

<style type="text/css">

.poser{position: absolute; top:800px}

.poserp{position: absolute; top:900px}

.decaler{position: absolute; left:100px}

.game{position: absolute; top:1060px}
```

```
</style>

 <script type="text/javascript">

  function surface(){

 var lar; var lon;

 lar=prompt(" donner la largeur du rectangle ");

 lon=prompt(" donner la longueur du rectangle ");

 alert(" la surface est : "+(lar*lon));

                    }

</script>

<script type="text/javascript">

 function resolution()

      {  var a,b,c,delta; var ch;

                          var tresul=[];

                                         a=prompt(" donner le premier coefficient : a = ");

                                                b=prompt(" donner le second coefficient : b = ");

                                                c=prompt(" donner le dernier coefficient : c = ");

          delta=b*b-4*a*c;

                           alert(" delta = "+delta);

    if(a==0 && b==0 && c==0)

                        alert(" tout reel est solution ");

                                                    else

                                                              if (a==0 && b==0)

                                                                                         alert(" pas de solution ");
```

```
if(a==0 && b!=0)
        alert(" la solution est : "+(-c/b));
    else if ( a!=0)
    {
    if (delta>0)
        {
    tresul[0]=(-b-Math.sqrt(delta))/(2*a);
        tresul[1]="   et
X2 =  ";
tresul[2]=(-b+Math.sqrt(delta))/(2*a);
        alert(" Nous
avons deux solutions distinctes : X1 = "+tresul.join(""));
        }
    else if(delta==0)
        alert(" Nous avons une solution double : x1 = "+(-
b/2*a));
        if (delta<0)
        {
tresul[0]=-b/2*a;
    tresul[1]=" - ";
    tresul[2]="i";
```

```
tresul[3]=+Math.sqrt(-delta)/(2*a);
tresul[4]="
et  X2 =  ";
tresul[5]=-b/2*a;
tresul[6]=" + ";
tresul[7]="i";
tresul[8]=+Math.sqrt(-delta)/(2*a);
alert(" Nous avons deux
solutions distinctes : X1 = "+tresul.join(""));



}

}



}
</script>
<script type="text/javascript">
var cpt_h=1;
```

```
var cpt_mn=0;
function tourne()
{
heures=document.getElementById("aiguille_h");
minutes=document.getElementById("aiguille_mn");
switch(cpt_mn)
{

case 0: minutes.style.top="510";minutes.style.left="370";break;
case 1: minutes.style.top="560";minutes.style.left="410";break;
case 2: minutes.style.top="610";minutes.style.left="420";break;
case 3: minutes.style.top="660";minutes.style.left="410";break;
case 4: minutes.style.top="700";minutes.style.left="370";break;
case 5: minutes.style.top="720";minutes.style.left="290";break;
case 6: minutes.style.top="700";minutes.style.left="220";break;
case 7: minutes.style.top="660";minutes.style.left="180";break;
case 8: minutes.style.top="610";minutes.style.left="170";break;
case 9: minutes.style.top="560";minutes.style.left="190";break;
case 10:
minutes.style.top="510";minutes.style.left="240";break;
case 11:
minutes.style.top="480";minutes.style.left="300";break;
default:alert(" il y a un bug ");break;
}
cpt_mn=(cpt_mn+1) % 12;
```

```
                    if(cpt_mn==0)
                     {
                      switch(cpt_h)
                        {
                       case 0: heures.style.top="480";heures.style.left="300";break;
                              case 1: heures.style.top="510";heures.style.left="370";break;
                              case 2: heures.style.top="560";heures.style.left="410";break;
                              case 3: heures.style.top="610";heures.style.left="420";break;
                              case 4: heures.style.top="660";heures.style.left="410";break;
                              case 5: heures.style.top="700";heures.style.left="370";break;
                              case 6: heures.style.top="720";heures.style.left="290";break;
                              case 7: heures.style.top="700";heures.style.left="220";break;
                              case 8: heures.style.top="660";heures.style.left="180";break;
                              case 9: heures.style.top="610";heures.style.left="170";break;
                              case 10: heures.style.top="560";heures.style.left="190";break;
                              case 11: heures.style.top="510";heures.style.left="240";break;
                               default:alert(" il y a un bug ");break;
                              }
                              cpt_h=(cpt_h+1) % 12;
                    }
            if (choix) window.setTimeout("tourne()","500");
            }
    </script>
    <script type="text/javascript">
```

```
function date_courant(){
    var tresul=[];
        var bess= new Date;
            var second=bess.getSeconds();
            var minute=bess.getMinutes();
            var heure=bess.getHours();
            var jour=bess.getDay(); var mois=bess.getMonth(); var
annee=bess.getYear(); var jour_mois=bess.getDate();
    switch(jour)
            {
                case 0 :tresul[0]=' Dimanche
';break;
                case 1 :tresul[0]=' Lundi
';break;
                case 2 :tresul[0]=' Mardi
';break;
                case 3 :tresul[0]=' Mercredi
';break;
                case 4 :tresul[0]=' Jeudi
';break;
                case 5 :tresul[0]=' Vendredi
';break;
                case 6 :tresul[0]=' Samedi
';break;
                default :tresul[0]=' jour
inexistant ';break;
                }
```

```
switch(mois)
{
case 0 :tresul[4]=' Janvier
';break;
case 1 :tresul[4]=' Fevrier
';break;
case 2 :tresul[4]=' Mars
';break;
case 3 :tresul[4]=' Avril
';break;
case 4 :tresul[4]=' Mais
';break;
case 5 :tresul[4]=' Juin ';break;
case 6 :tresul[4]=' Juillet
';break;
case 7 :tresul[4]=' Août
';break;
case 8 :tresul[4]=' Septembre
';break;
case 9 :tresul[4]=' Octobre
';break;
case 10 :tresul[4]=' Novembre
';break;
case 11:tresul[4]=' Decembre
';break;
default :tresul[4]=' mois
inexistant ';break;
```

```
}
if(annee<100) {annee+=1900; tresul[6]=annee.toString(10);}
else if( annee>=100) {annee+=1900; tresul[6]=annee.toString(10);      }

tresul[1]="  ";
tresul[2]=jour_mois;
tresul[3]="  ";
tresul[5]="  ";
tresul[7]="  à :";
tresul[8]=heure;
tresul[9]=" H: ";
tresul[10]=minute;
tresul[11]=" mn : ";
tresul[12]=second;
tresul[13]=" S ";
alert(" Nous somme au: "+tresul.join(""));

}
function modifie_paragraph()
{
var par=document.getElementById("texte");
par.innerHTML=" Le <b> JAVASCRIPT</b> permet
de faire des pages web <b><i this.style.color='red'>inter-actives</i></b> de haut niveau"

}
```

```
function centrer()
    {
      var titre=(document.getElementsByTagName("h1"))[0];
          titre.setAttribute("align","center");
            }


function droite()
    {
      var titre=(document.getElementsByTagName("h1"))[0];
          titre.setAttribute("align","right");
            }
function ajout()
    {
            var lis=document.getElementById("liste");
            var texte1=document.getElementById("commande").value;
                   var elt=document.createElement("li");
                    texte=document.createTextNode(texte1);   // on pouvait aussi faire :
elt.innerHTML=texte1; lis.appendChild(elt);
                    elt.appendChild(texte);
          lis.appendChild(elt);
              }
function supprimer()
```

```
{
        var lis=document.getElementById("liste");
        lis.removeChild(lis.lastChild);
        }
var x=420;
var y=740;
var dx=5, dy=10, sensx=1, sensy=1;
function coordonnee()
    {
            x=x+(sensx*dx);
        if(x>858)
            {
                sensx=-1;
                x=x-(x-858);
                }
                else if (x<404)
                    {
                        sensx=1;
                        x=x+(404-x);
                        }
            y=y+(sensy*dy);

            if(y>1178)
            {
```

```
sensy=-1;
y=y-(y-1178);
}
else if (y<724)
{
sensy=1;
y=y+(724-y);
}

}
function progression()
{
document.getElementById("ballon").style.left=x+"px";
document.getElementById("ballon").style.top=y+"px";
}
function jouer()
{
coordonnee();
progression();
}
function accelerer() {
dx +=2
if (dx>200) dx=200
dy +=2
```

```
        if (dy>200) dy=200
    }
    function ralentir() {
        dx -=2
        if (dx<0) dx=0
        dy -=2
        if (dy<0) dy=0
    }
function gauche()
        {
                    sensx=-1;
                    sensy=0;

                    }
    function droite()
        {
                    sensx=1;
                    sensy=0;
                    }
    function haut()
        {
                    sensx=0;
                    sensy=-1;
                    }
```

```
function bas()
{
sensx=0;
sensy=1;
}

</script>

</head>
<body>
<h3> FONCTION DE CALCUL DE LA SURFACE D UN RECTANGLE </h3>
<p> Cliquez sur le bouton pour demarrer le calcule : <button onclick="surface()"> LANCER LA FONCTION </button> </p>
<h3> RESOLUTION DU SECOND DEGRE </h3>
<p> Cliquez sur le bouton pour demarrer la résolution : <button onclick="resolution()"> LANCER LA RESOLUTION </button> </p>
<p> Cliquez ici pour connaitre la Date précise du jour : <button onclick="date_courant()"> LANCER LA DATE  </button> </p>
<p> <span style="text-decoration: none; font-family: Times New Roman; font-size: 200%;"onmouseover="this.style.color='red'; this.style.fontFamily='algerian'; this.style.lineHeight='200%'; this.style.fontSize='200%'; this.style.textDecoration='line-through'" onmouseout="this.style.color='yellow'; this.style.fontFamily='Times New Roman';this.style.lineHeight='200%'; this.style.fontSize='200%'; this.style.textDecoration='none'" onclick="this.style.color='green';this.style.position='relative';this.style.left='600px'" ondblclick="this.style.color='blue'"> KING-WARRIORS</span></p>
```

```
<ol>

    <li> <img src="w.jpg" alt="w" onmouseover="this.src='../images/snoopy.jpg'; this.width='100';
this.height='120';this.alt='snoopy!'" onmouseout="this.src='../images/w.jpg'; this.width='50';
this.height='70';this.alt='w'"  width="50" height='70'></li>


    <li><span onclick="this.innerHTML=' une police <i> italic </i>'"> une police italic</span></li>

        </ol>

<h1 onclick="modifie_paragraph()" title=" quand on click, le paragraphe en dessous est modifier">
JAVASCRIPT </h1>

<p id="texte" onclick="centrer()" ondblclick="droite()" title=" quand on click centre le titre "> Le
javascript permet de faire des pages web inter-actives de haut niveau </p>

<p>Demarrer l'horloge <button onclick="choix=true;tourne()">lancer</button><button
onclick="choix=false">arreter</button></p>

<!--<input value="lancer!" onclick="choix=true;tourne()" type="button"><input value="arreter!"
onclick="choix=false" type="button"> -->

<div id="chiffre1"><p>1</p></div>

<div id="chiffre2"><p>2</p></div>

<div id="chiffre3"><p>3</p></div>

<div id="chiffre4"><p>4</p></div>

<div id="chiffre5"><p>5</p></div>

<div id="chiffre6"><p>6</p></div>

<div id="chiffre7"><p>7</p></div>

<div id="chiffre8"><p>8</p></div>

<div id="chiffre9"><p>9</p></div>

<div id="chiffre10"><p>10</p></div>

<div id="chiffre11"><p>11</p></div>
```

```
<div id="chiffre12"><p>12</p></div>

<div id="centre"><p>o</p></div>

<div> <img src="bleu.gif" id="aiguille_h" width="21" height="21"></div>

<div> <img src="rouge.gif" id="aiguille_mn" width="21" height="21"></div>

<p class=poser> Lister vos produits <button onclick="ajout()">AJOUTER</button><button
onclick="supprimer()">supprime</button>

<br><br><input id="commande" type="text" class=decaler>

 </p>


<ul id="liste" class=poserp>

   <li>1 kg de farine</li>

         <li>5 miche de paim</li>

</ul>

<img src="bille1.gif" id="ballon" style="position: absolute; top:724px; left: 404px">

<p class=game><input value="<-" onclick="gauche()" type="button" style='position:absolute;
left:110px; top:40px'>

<input value="->" onclick="droite()" type="button" style='position:absolute; left:200px; top:40px'>

<input value="haut" onclick="haut()" type="button" style='position:absolute; left:150px'>

<input value="bas" onclick="bas()" type="button" style="position:absolute; left:150px; top:70px">

<input value="accelerer" onclick="accelerer()" type="button" style='position:absolute; top:120px' >

<input value="ralentir" onclick="ralentir()" type="button" style='position:absolute; top:120px; left:
250px'>

<input value="play" onclick="setInterval(jouer,100)" type="button" style='position:absolute;
top:160px; left: 150px'>

</p>
```

```
<table border="4" style="width: 500px; height: 500px; position:absolute; top: 700px; left: 400px"
cellspacing="0" cellpadding="0">
 <caption> GAME </caption>
 <tr>
 <td>



 </td>
 </tr>
</table>
</body>
</html>
```

CONCLUSION :

Dans ce résumé, nous avons essayé de vous donner le maximum de connaissances pour bien gérer vos pages web avec le langage javascript, toutefois pour plus de détailles, nous vous renverrons à l'Internet et aux manuels. Dans nos prochains articles, nous mettrons l'accent sur la communication entre le client et le serveur.

Abdourahmane FALL

fallprofessionnel87@yahoo.fr

+221 77 274 46 71