Introduction à JavaScript

Stéphane Genaud

2005
Revision : 1.17
Date : 2007 – 01 – 1011 : 33 : 17

Table des matières

Chapitre 1

Introduction

A propos de ce document Ce document ne suppose que très peu de connaissances pré-requises. Des notions de HTML permettront néanmoins de faire clairement la distinction entre le JavaScript et le HTML quand ces deux langages sont présents dans un même texte. Le but principal est d'abord d'illustrer des concepts de base d'algorithmique et de langage de programmation. JavaScript n'a pas été choisi pour ses aspects didactiques mais pour ses atouts pratiques : les navigateurs modernes les plus répandus possèdent tous un interpréteur Javascript permettant de tester simplement ses programmes, et quasiment n'importe où.

J'esaye de compléter ce document en fonction des remarques des lecteurs ou des évolutions technologiques. Toutes les remarques (notamment les corrections d'erreurs) sont donc les bienvenues[1].

JavaScript est un langage de script orienté-objet utilisé pour le développement d'applications internet. Ce langage a été developpé par la société Netscape Corporation, qui l'a introduit, pour la première fois dans son *Navigator 2.0* en 1996.
Netscape met à disposition sur son site un guide de référence complet du langage [Net98].
Afin de rendre ce langage accessible à des navigateurs conçus par d'autres entreprises, un procédé de normalisation à été entrepris dès 1996. La norme porte le nom ECMA-262 [ECM99], et on se refère parfois à JavaScript sous le nom d'ECMA script. Cette norme est un sous-ensemble du langage JavaScript conçu par Netscape, et il est nécessaire de n'utiliser que les spécifications de cette norme pour s'assurer que le code écrit sera compris de tous les navigateurs interprétant le JavaScript.

Ne pas confondre JavaScript et Java, langage conçu et développé par Sun Microsystems.

Les programmes JavaScript s'intègrent dans le code HTML d'une page web. L'intérêt d'un langage comme JavaScript est de pouvoir contrôler dynamiquement le comportement d'un 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 d'une zone donnée, afficher des bandeaux publicitaires animés, orienter automatiquement le visiteur sur une autre page, etc ... . Pour des exemples de mises en application de JavaScript, vous pouvez vous reporter à [Bra99].

[1] Je remercie les gens ayant relu le document. Pour l'instant : Pierre Dureau (2005).

1.1 JavaScript est utilisé côté client

La charge d'exécuter le code JavaScript incombe au client, c'est-à-dire à un navigateur la plupart du temps. Le serveur envoie le code HTML et JavaScript au client qui l'interprète dès qu'il est chargé. Ce type de fonctionnement s'oppose aux langages orientés serveurs, comme PHP[BAS+99] ou ASP. Dans ce cas, le serveur exécute le programme pour produire du HTML et l'envoie au client.

1.2 JavaScript est interprété

On distingue habituellement deux modes d'exécution des programmes. Dans le premier mode, dit **compilé**, le texte de programme écrit par le programmeur (on dit aussi programme source) est traduit dans un autre format, directement compréhensible par la machine. La phase de traduction s'apelle la *compilation* et donne un fichier exécutable (ou binaire). Les avantages de ce mode sont la rapidité d'exécution (les instructions sont directement exécutées par la machine) et la validité syntaxique du programme source (pour que le compilateur puisse traduire le source il faut que les instructions soient syntaxiquement correctes). D'autre part, un avantage commercial est de pouvoir donner à une personne tierce, uniquement le fichier exécutable sans que cette personne puisse voir le programme source. L'inconvénient est que le fichier exécutable dépend à la fois de la machine et du système d'exploitation, et il est par conséquent plus difficile de le distribuer.

Le deuxième mode est dit **interprété** : un programme interprète lit le programme source, et essaye d'exécuter les instructions les unes après les autres. A la différence du mode compilé, il n'y a pas de fichier exécutable. L'avantage de ce mode est que toute personne ayant le programme interprète sur son ordinateur, peut exécuter le programme. L'inconvénient est la lenteur dûe à l'exécution et de l'interprète et du programme.

JavaScript est un langage *interprété*. Il faut donc un programme interprète pour exécuter des programmes écrits dans ce langage. Les deux navigateurs les plus répandus à l'heure actuelle sur le marché, Netscape et Internet Explorer, incorporent de tels interprètes. Netscape Navigator 2.0, et ses versions ultérieures sont capables d'interpréter les programmes JavaScript. Quand un navigateur demande l'accès à une page HTML contenant un programme JavaScript, le serveur envoie le document complet (incluant le HTML et les instructions JavaScript) au navigateur. Le navigateur affiche alors le texte HTML et exécute le code JavaScript.

Quels navigateurs sont capables d'interpréter JavaScript ? Netscape a introduit pour la première fois l'interprétation de code JavaScript dans son Navigator version 2. Microsoft, avec Internet Explorer version 3, a suivi, en implémentant une partie du langage. D'autres navigateurs intègrent aussi l'interprétation de JavaScript aujourd'hui. Il est difficile de donner une information exhaustive à un moment donné, sur les versions de navigateurs étant capables d'interpréter telle ou telle version du langage, du fait de l'évolution rapide et incessante des logiciels et des langages. Citons simplement trois autres navigateurs qui respectent la norme ECMA-262 :

- Opera, logiciel propriétaire disponible sous Windows/Linux/MacOs,
- Konqueror, l'explorateur de fichiers et navigateur de l'environnement KDE sous Linux,
- Safari, le navigateur fourni dans l'environnement MacOSX, basé sur Konqueror,
- Firefox, le navigateur de la fondation Mozilla, basé sur le navigateur Mozilla. Mo-

zilla aujourd'hui complètement remplacé par Firefox, a fourni son moteur Gecko à de nomberux autres navigateurs, dont les Netscape 6 et 7.

Détecter les erreurs Lors de la conception du programme javascript, des erreurs de programmation interviennent fréquemment. La première des choses à faire est bien sûr de vérifier la conformité de ce qu'on a écrit avec ce qu'on voulait dire. Il faut également faire particulièrement attention à la syntaxe, les fautes de frappe étant fréquentes. Le programme étant interprété par le navigateur, celui ci peut aussi mettre à votre disposition un outil d'aide à la détection des erreurs. Les anciens navigateurs (e.g. NETSCAPE séries 4.7x, Microsoft IE 5) vous signalent une erreur de manière discrète[2], voire pas du tout. dans le bandeau bas de la fenêtre du navigateur de manière discrète , sans plus de détails. Les navigateurs MOZILLA 1.x et NETSCAPE 7.x proposent dans le menu outils/développement web une console JavaScript (voir la figure 1.1) et même un *debugger*.

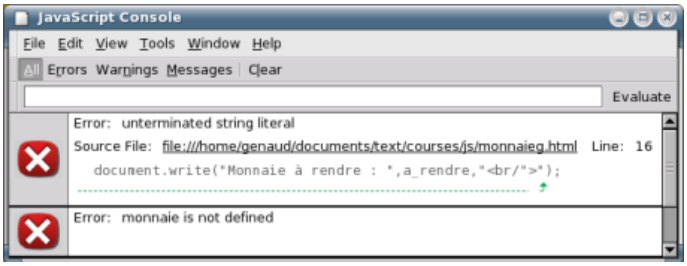

FIG. 1.1 – La console javascript de Mozilla 1.4

Enfin, les navigateurs récents comme Firefox 2.0 proposent des plugins puissants aidant la recherche d'erreurs et plus généralement le développement orienté web (e.g. plugin *web-developper*, voir illustrations 1.2 et 1.3).

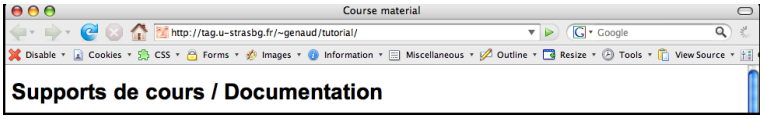

FIG. 1.2 – Le plugin web-developper dans FireFox 2.0

1.3 Où et comment insérer du JavaScript ?

Pour placer des intructions JavaScript dans une page HTML, il faut utiliser la balise ¡`script`¿ de la manière suivante :

[2]Cependant, NETSCAPE séries 4.7x permet (même s'il est ancien) d'obtenir une console javascript expliquant les erreurs : quand une erreur est signalée dans le bandeau bas de la fenêtre, tapez `javascript :` dans la barre d'adresse

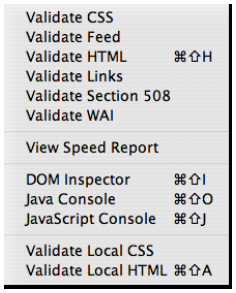

FIG. 1.3 – Le menu de *tools* dans le plugin web-developper

```
<script type="text/javascript">
       .....
       instructions javascript
       .....
</script>
```

Notons que l'ouverture `<script >` est suffisante pour une écriture rapide, mais que la type `text/javascript` est nécessaire pour que la page soit conforme à la norme HTML 4.01.
Si l'on préfère inclure le code JavaScript stocké dans un autre fichier, on peut l'indiquer par une ligne comme :

```
<script src="mon_code.js" type="text/javascript"></script>
```

Ces instructions peuvent s'insérer n'importe où à l'intérieur du code HTML. Dans l'exemple suivant, on note l'utilsation de la fonction `document.write` qui permet, en JavaScript, d'écrire un message dans la fenêtre du navigateur. Nous reviendrons sur cette fonction plus tard.

```
<html>
    ligne 1 de texte . <br>
<script type="text/javascript">
    document.write("ligne 2 de texte.")
</script>
    <br>ligne 3 de texte.
</html>
```

Lors du chargement de cette page HTML, l'utilisateur ne verra s'afficher dans son navigateur que les trois lignes :

```
ligne 1 de texte
ligne 2 de texte
ligne 3 de texte
```

sans savoir si c'est du texte produit par HTML ou par JavaScript.

Cacher JavaScript à certains navigateurs Certains navigateurs ne comprennent pas le JavaScript. Pour éviter les erreurs que signaleraient ces navigateurs lors de la lecture du JavaScript, on peut leur faire croire qu'il s'agit de commentaires HTML. Par exemple :

```
<html>
<script type="text/javascript">
<!-- cache le script .
       document.write("Bonjour !")
// fin du commentaire ici. -->
</script>

reste du texte HTML.
</html>
```

Chapitre 2

Le langage

Nous présentons dans ce chapitre deux notions distinctes. D'une part, les concepts de base de la programmation[1] et d'autre part, la *syntaxe* du langage qui permet d'exprimer ces concepts. La syntaxe du langage consiste en l'ensemble des règles qui nous sont imposées sur le vocabulaire employé lorsqu'on écrit un programme dans ce langage.

Tou au long du chapitre, les conventions suivantes sont utilisées. On appelle *mot-clé* tout mot ou symbole appartenant au langage JavaScript. Ces mots-clés sont typographiés en police télétype : par exemple, `if` est un mot-clé (il indique le début d'un test). En pratique, ceci signifie qu'on ne peut utiliser un mot-clé en dehors de son contexte. Vous ne pourrez pas par exemple, choisir if comme nom de variable. La table 2.1 recense les mots-clés du langage.

2.1 Variables

2.1.1 Déclaration et lecture/écriture

Une *variable* est une case mémoire à laquelle on a donné un nom. Dans les langages impératifs, (comme JavaScript, Java,C/C++) le contenu de cette case mémoire peut varier au cours de l'exécution du programme. On peut ranger une valeur dans cette case, ou de manière synonyme, dans la variable, par une instruction nommée *affectation*. On peut aussi lire son contenu à tout moment. On peut par exemple indiquer que l'on utilise un variable pour compter les points d'un match. Il faut alors l'indiquer par l'instruction suivante, qu'on appelle *déclaration* de variable :

```
var compteur;
```

Ici, `var` est un mot-clé et `compteur` un nom choisi par le programmeur. Après une telle déclaration et sans précision supplémentaire, la variable vaut `undefined`[2].
Si je veux signifier à un moment du programme que le nombre de points est quatre, j'écris l'instruction d'affectation :

```
compteur = 4;
```

Les noms de variables sont de longueur quelconques, peuvent contenir des lettres, chiffres et le caractère souligné (_) , mais doivent commencer par une lettre. Notons que JavaScript, contrairement à HTML, distingue les majuscules des minuscules, et qu'ainsi `MaVariable` et `mavariable` désignent deux variables différentes.

[1] De la programmation impérative, c'est-à-dire le modèle de programmation le plus répandu.

[2] Depuis JavaScript ¿= 1.3. Auparavant, `undefined` n'existait pas et la valeur d'une variable non initialisée était `true`.

abstract	float	public
boolean	for	return
break	function	short
byte	goto	static
case	if	super
catch	implements	switch
chat	import	synchronized
class	in	this
const	instanceof	throw
continue	int	throws
default	interface	transient
do	long	true
double	native	try
else	new	var
extends	null	void
false	package	while
final	private	width
finally	protected	

TAB. 2.1 – Liste des mots-clés

Maintenant supposons que mon compteur a une valeur que je ne connais pas, et que je veuille indiquer qu'il faut ajouter un á cette valeur. Pour cela, je peux écrire l'instruction :

```
compteur = compteur + 1;
```

Pour comprendre la signification de cette instruction, il faut savoir que l'ordinateur va d'abord évaluer la partie à droite du symbole égal. Le contenu de la variable `compteur` est lu, et on additionne un à ce contenu. Dans un deuxième temps, la somme est ré-écrite dans la variable `compteur`. Cette opération est tellement courante en informatique qu'elle porte un nom : c'est l'*incrémentation* d'une variable.

Il faut ici faire attention au sens du symbole `=`. Il dénote en JavaScript (comme en Java ou en C) l'*affectation*, et non l'*égalité*, auquel cas le texte précédent n'aurait pas de sens. On signifie l'égalité par le symbole `==`, comme nous le verrons par la suite.

2.1.2 Types de données

On pourrait se poser la question suivante : peut on mettre n'importe quelle sorte de données dans une variable ? La réponse est non, car il se poserait alors des problèmes de cohérence : que se passerait il si on mettait le message ¡¡bonjour¿¿ dans `compteur` et que l'on essayait ensuite de lui ajouter un ?
Pour assurer la cohérence des instruction, les langages informatiques définissent habituellement des *types de données*. Un type de données est un ensemble de *valeurs* et un ensemble d'*opérateurs* sur ces valeurs. Par exemple, le type *entier* est constitué de l'ensemble des entiers et des opérateurs +, -,*, /, etc. Un autre type très utilisé est le type *chaine* (ou chaine de caractères) permettant de traiter les messages. Une chaine est un ensemble de caractères composé indifféremment de lettres, symboles ou chiffres. Sur l'ensemble des chaines de caractères, on peut définir des opérateurs comme la *concaténation* (mettre deux chaines bout-à-bout). La section 2.6 revient

en détail sur les types du langage JavaScript.

JavaScript, comme la plupart des langages de script, est un langage très faiblement typé. Ceci ne veut pas dire qu'il est capable d'effectuer l'instruction 1+"bonjour", mais qu'il n'impose pas au programmeur d'indiquer explicitement le type d'une variable lors de la déclaration de celle-ci.

2.1.3 Exemples de déclaration

Voici, quelques exemples de déclarations :

```
var texte;
var cle;
var compteur = 3;
var Erreur = "Connexion perdue.";
var Erreur2 = 'Pas assez de mémoire';
```

Notons que l'on peut *initialiser* les variables (y inscrire une valeur initiale) à leur déclaration, comme c'est le cas dans les trois dernières déclarations.
On remarque aussi dans l'exemple précédent que l'on peut indifféremment utiliser les guillements ou les quotes pour délimiter une chaine de caractères.

2.2 Opérations, expressions

La plupart des expressions que l'on rencontre dans des langages comme C/C++ et Java existent en JavaScript. Nous ne donnerons que quelques exemples ici, et nous découvrirons ces expressions au fil de ce document. Voici quelques exemples, les instructions sur la même ligne ayant la même signification.

```
total += 4;                          total = total + 4;
i++;                                 i = i + 1;
msg = "code erreur " + code;
```

2.3 Commentaires

Parfois, un programmeur souhaite annoter le code qu'il vient d'écrire, mais sans que ces annotations n'interfèrent avec les instructions proprement dites. On appelle ces annotations des *commentaires* du programmes. Pour que l'interprète ne confonde pas ces caractères avec les instructions, (pour qu'il les ignore), on fait précéder un commentaire disposé en fin de ligne par les deux barres obliques (`//`) et on utilise `/*` et `*/` pour délimiter un commentaire sur plusieurs lignes. Voici un exemple de code commenté.

```
/* test pour poser la bonne question */
  if (reduc && vieux) { // a une reduction et est vieille
     document.write("Voulez que quelqu'un vous accompagne ?");
  }
```

2.4 Déroulement d'un programe

2.4.1 La séquence

Quand on veut faire exécuter plusieurs instructions (c'est en pratique toujours le cas) on écrit ces instuctions de haut en bas, en insérant un point-virgule entre chaque instruction [3] . Pour incrémenter ma variable `compteur` d'une unité, puis de deux, j'écris :

```
compteur = compteur + 1;
compteur = compteur + 2;
```

Cependant, la séquence simple rendrait fastidieuse l'écriture d'instructions se répétant un grand nombre de fois. Si l'on doit par exemple afficher les cinquante premiers entiers, nous aurions besoin de cent instructions : cinquante instructions d'affichage, et cinquante incrémentations. Pour simplifier l'écriture, le langage fournit des *structures de contrôle* permettant de modifier la séquence d'exécution des instructions.

2.4.2 Structures de contrôle

Parmi ces structures on trouve la *boucle* (aussi appelée itérative) et le *test* (ou alternative).

La boucle `for`

La boucle permet de répéter un certain nombre de fois un ensemble d'instructions. La boucle est introduite par le mot-clé `for`.
Pour afficher nos cinquante premiers entiers, nous pouvons écrire une boucle qui nous permet de réduire le texte de programme à trois lignes (au lieu de cent).
Cette boucle utilise la variable `i` qui vaut 1 initialement. Ensuite le *corps* de la boucle, c'est-à-dire les instructions à l'intérieur des crochets (`{` et `}`) est exécuté tant que la condition `i<=50` est vraie. A chaque tour de boucle, on incrémente la variable `i`.

```
for (i=1; i<=50; i++) {
      document.writeln(i);
}
```

La syntaxe de la boucle en JavaScript est la suivante. La notation utilise les crochets pour indiquer les paramètres optionnels.

`for (`[*init−expr*`;`][*condition*`;`][*incr−expr*] `){`
instructions
`}`

Le comportement de cette instruction est :

1. exécute *init−expr*
2. si *condition* est faux aller en 6., sinon aller en 3.
3. exécuter *incr−expr*
4. exécuter *instructions*
5. aller en 2.
6. sortie de l'instruction `for`

[3] En réalté, le point-virugle n'est pas nécessaire en JavaScript. Il me semble cependant préférable de prendre cette habitude car de nombreux langages ayant une syntaxe très proche utilise le point-virgule pour séparer deux instructions.

La boucle `while`

Une autre structure itérative plus simple est la boucle `while`. Celle-ci dit simplement qu'on doit répéter un ensemble d'instructions tant qu'une condition est vraie.

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

Le comportement de cette instruction est :

1. évalue *condition*
2. si *condition* est vraie aller en 3, sinon aller en 5.
3. exécuter *instructions*
4. aller en 1.
5. sortie de l'instruction `while`

Nous pouvons par exemple refaire un calcul tant que l'utilisateur le désire. Supposons que nous disposions par ailleurs d'une fonction `calc_moyenne()` calculant une moyenne de notes. Dans l'exemple suivant, la variable `recommence` est un booléen qui contrôle la répétition du calcul de la moyenne. On relance ce calcul tant que l'utilisateur n'a pas cliqué sur *Annuler* dans la boite de dialogue affichée par `prompt()`.

```
var recommence = true;

    while (recommence) {
       m = calc_moyenne();
       recommence=prompt("La moyenne est " + m + ".Recommencer ?");
    }
```

Notons qu'il est important d'initialiser la variable `recommence` à la valeur vraie afin d'entrer dans la boucle la première fois.

L'alternative

La structure alternative permet d'exécuter un ensemble d'instruction quand une condition est vraie. Le type de la condition est booléen (la condition est soit vraie soit fausse). La structure alternative est introduite par le mot-clé `if`, et sa syntaxe est la suivante :

```
if (condition_1){
      instructions_1
}
[else  {
      instructions_2
}]
```

Le comportement de cette instruction est :

1. si $condition_1$ est vrai, aller en 2, sinon aller en 3.
2. exécuter $instructions_1$ et aller en 4.
3. si il y a une partie `else`, exécuter $instructions_2$ et aller en 4 sinon aller en 4.
4. sortie de l'instruction

Exemple : on peut simplement tester la valeur d'une variable, comme dans l'exemple suivant.

```
    if (i==50) {
       document.writeln("i est égal à 50")
    }
    else {
       document.writeln("i n'est pas égal à 50")
    }
```

2.4.3 Fonctions

De même que la boucle permet de répéter plusieurs instructions, la fonction est très utile pour accomplir une tâche répétitive. Le principe consiste à regrouper dans un ¡¡bloc¿¿ un ensemble d'instructions nécessaire pour accomplir une tâche, à nommer ce bloc, et à l'appeler à chaque fois qu'on a besoin d'effectuer la tâche.

Une sophistication supplémentaire réside dans la possibilité de pouvoir *paramétrer* ce bloc, c'est-à-dire exécuter le bloc avec des paramètres pouvant être différents à chaque appel. Il va de soi que si je construis un bloc permettant de calculer la factorielle d'un nombre, je veux que ce bloc puisse calculer la factorielle de n'importe quel entier. Cet entier est mon paramètre (ou de manière synonyme, l'*argument* du bloc).

De plus, les blocs en JavaScript, se nomment *fonctions*, comme les fonctions mathématiques, ils renvoient un résultat unique.

La définition d'une fonction est introduite par le mot clé `function` suivi de ses arguments, puis du code de la fonction entre accolades ({}). Le résultat de la fonction est indiqué par le mot-clé `return`. Par exemple, ci-dessous est définie la fonction `carre` qui rend le carré de son argument.

```
<script type="text/javascript">
  function carre(i) {
        return (i*i)
   }
</script>
```

Notons que rien ne se passe si on charge seulement la page contenant ce script. Il faut en effet appeler la fonction avec un argument effectif pour que l'évaluation se fasse. Pour cela, on pourra par exemple ajouter après le code de la fonction, un appel à cette fonction :

```
<script type="text/javascript">
      document.write("La fonction a retourné ",carre(4),".")
</script>
```

Pour appeler la fonction, il faut bien sûr qu'elle ait été définie auparavant. Par conséquent, il est conseillé de placer la définition de fonction dans l'entête de la page HTML, et les appels dans le corps de cette page. Pour notre exemple, la page HTML a donc la structure suivante :

```
<head>
<script type="text/javascript">
   function carre(i) {
     document.write("on a appelé carre avec l'argument ",i ,"<br>")
```

```
    return i * i
  }
</script>
</head>
<body>
<script type="text/javascript">
    document.write("La fonction a retourné ",carre(4),".")
</script>
</body>
```

2.5 Portée des variables

Les variables déclarées en dehors d'une fonction sont *globales*, c'est-à-dire qu'il est possible de les lire ou de les modifier de n'importe quelle partie du script ou de n'importe quelle fonction. Par opposition, les variables déclarées à l'intérieur d'une fonction sont *locales*, c'est-à-dire que leur valeur n'est connue qu'à l'intérieur de la fonction.

2.6 Les types de données en JavaScript

2.6.1 Le type booléen

Une expression de type *booléen* ne peut prendre que les valeurs *true* ou *false*. Ce type est notamment utilisé dans l'instruction `if`. L'ensemble des opérateurs définis sur l'ensemble de valeurs $\{true, false\}$ est présenté dans la table 2.2. (dans les exemples de cette table, les variables `a` et `b` sont de type booléen).

nom	syntaxe	exemple	définition
et logique	`&&`	`a && b`	$true, true \rightarrow true$ $true, false \rightarrow false$ $false, true \rightarrow false$ $false, false \rightarrow false$
ou logique	`||`	`a || b`	$true, true \rightarrow true$ $true, false \rightarrow true$ $false, true \rightarrow true$ $false, false \rightarrow false$
Négation	!	`!a`	$true \rightarrow false$ $false \rightarrow true$

TAB. 2.2 – Les opérateurs logiques (définis sur les booléens)

Remarque : l'implication est absente mais peut s'écrire à partir de la négation et du ou logique ($\vee$). On a : $(a \Rightarrow b) \equiv (\neg a \vee b)$, (c'est-à-dire `(!a || b)` en JavaScript).

2.6.2 Le type entier

2.6.3 Le type chaine de caractères

L'ensemble de valeurs que peut prendre une variable de type chaine est une combinaison quelconque d'un nombre quelconque de caractères quelconques. Comme indiqué dans l'introduction, des exemples de chaines sont `"abcdef"` ou `"az1r"` ou

"123". Dans les programmes, une chaine de caractères est délimitée par des guillemets ("") ou des quotes (’’).
La liste des méthodes sont listées en section A.20, page 56. On peut donner quelques exemples de méthodes qui existent sur les chaines :

– `indexOf` : cette méthode indique à quelle place se situe la première occurrence d'une sous-chaine dans une chaine. Par exemple, dans la chaine "le baba de ECMA-script", la première occurrence de la sous-chaine "ba" est en position 3 (on compte à partir de 0). Dans la chaine "jones@free.fr", la sous-chaine "@" est à la position 5. Si la sous-chaine n'existe pas dans la chaine, `indexOf` rend -1. En JavaScript, les deux exemples précédents peuvent s'écrire ainsi, `p1` et `p2` recevan les positions indiquées :

```
var s1 = "le baba de ECMA-script";
var s2 = "jones@free.fr";

p1 = s1.indexOf("ba");   // p1 = 3
p2 = s2.indexOf("@");    // p2 = 5
p3 = s2.indexof("abc");  // p3 = -1
```

2.6.4 Les tableaux

Un *tableau* permet de stocker plusieurs valeurs de même type, comme par exemple plusieurs entiers dans une seule variable. Les tableaux sont donc un type à part entière : ma variable de contenant plusieurs entiers est de type ¡¡tableau d'entiers¿¿.
En JavaScript[4], la déclaration d'une telle variable se fait par exemple comme suit :

```
var tableau_entiers = new Array();
```

Remarquez que nous n'avons pas précisé que les éléments du tableau étaient des entiers, du fait du typage faible de JavaScript. Ce sont les éléments que nous metterons dans le tableau qui détermineront le type du tableau.
Le lecteur connaissant d'autres langages de programmation de type impératif peut être surpris qu'il ne soit pas obligatoire de préciser la taille du tableau que l'on déclare. Les tableaux sont effet *dynamiques*, c'est-à-dire qu'on peut leur ajouter à tout moment, autant d'éléments que l'on veut.

Pour accéder à un valeur particulière du tableau, il faut indiquer sa place dans le tableau, à l'aide de crochets. Attention, le premier élément est numéroté 0. On peut par exemple mettre la valeur 56 dans la première case du tableau, et la valeur 9 dans la cinquième case :

```
tableau_entiers[0] = 56;
tableau_entiers[4] = 9;
```

Il existe une façon plus concise d'initialiser les valeurs d'un tableau, à la condition de mettre des valeurs dans toutes les cases du tableau (pas de “trou”). On peut alors initialiser le tableau par l'une ou l'autre des lignes suivantes :

```
tableau_entiers = new Array(56,12,43,4,9);
tableau_entiers = [56,12,43,4,9];
```

[4] cette syntaxe n'est valable que pour des versions de navigateurs supérieures à Netscape 2 et IE 3-DLL JScript 1. Pour une utilisation des tableaux avec des verions anciennes de navigateurs, vous pouvez consulter [Goo98].

Il faut noter que la deuxième ligne emprunte une notation uniquement acceptée par des navigateurs récents[5].
On peut ensuite lire le contenu d'une case, et le faire apparaitre à l'écran par exemple :

```
document.write(tableau_entiers[4]);
```

Tout tableau possède une *longueur*. La longueur du tableau est le numéro de case occupée le plus grand, plus un. Les tableaux étant dynamiques, l'interpréteur met à jour la longueur du tableau à chaque affectation. Dans les deux affectations précédentes, la longueur du tableau est 1 après la première affectation, et 5 après la deuxième affectation. Le programmeur peut obtenir cette longueur en consultant la propriété `length` du tableau considéré : par exemple, on peut faire afficher la longueur du tableau par :

```
document.writeln(tableau_entiers.length);
```

Les tableaux sont très utiles car ils permettent de manipuler de manière concise un ensemble de valeurs. En partriculier, on peut utiliser une boucle pour faire afficher toutes les valeurs contenues dans le tableau.

```
for (i=0;i<tableau_entiers.length;i=i+1)
    document.writeln(tableau_entiers[i]);
```

Tableaux multi-dimensionnels Les tableaux utilisés jusqu'à présent étaient des vecteurs, avec une dimension. Cependant, on a très vite besoin de tables, à deux dimensions. Une limitation importante et décévante de JavaScript est qu'on ne peut décrire simplement des *tableaux multi-dimensionnels*.

Cependant, il existe des moyens détournés de créer de tels tableaux (voir [Goo98] pour plus de détails). Il s'agit en fait de créer un tableau a une dimension, dont chaque case contient un nouveau tableau à une dimension. Dans l'extrait suivant, on créé d'abord un tableau de quatre cases (ligne 1), puis on met dans chacune des cases, un nouveau tableau de quatre cases (ligne 3). Enfin, on initialise chacune des seize cases du tableau avec la valeur 1 (ligne 5). On remarquera la notation utilisée pour accéder à une case d'un tel tableau bi-dimensionnel.

```
1  a = new Array(4);
2  for (i=0;i<4;i=i+1) {
3     a[i] = new Array(4);
4       for (j=0;j<4;j=j+1)
5         a[i][j] = 1;
6  }
```

Opérateurs sur les tableaux Les méthodes suivantes sur les tableaux sont définies à partir de JavaScript 1.3. Ces méthodes s'appliquent toutes sur un tableau et utilisent donc la notation pointée (voir la section 4.2.1, 31 pour des explications détaillées). Ainsi si `t`, `u` et `v` sont des tableaux, on peut écrire :

```
u = t.reverse();     /* u devient l'inverse du tableau t */
v = u.concat(t);     /* v devient la concaténation de u et t */
```

[5] A partir de Netscape 3 et IE 4

```
v = u.concat(t,u,t); /* v est u,t,u,t mis bouts à bouts */
v = u.sort();        /* v devient le tableau u trié */
```

`concat()`	Regroupe (concatène) plusieurs tableaux et rend un nouveau tabeau
`join`(delimiter)	Rend une chaîne contenant les éléments du tableau séparés par le délimiteur spécifié
`pop()`	Retourne le dernier élément du tableau et le supprime du tableau
`push`(e1,e2, ...)	Ajoute les éléments e1, e2, ... à la fin du tableau et retourne la nouvelle longueur.
`reverse`()	Renverse le tableau.
`shift`()	Retourne le premier élément du tableau et le supprime du tableau.
`slice`(begin[,end])	Retourne un nouveau tableau constitué d'une sous-partie d'un tableau existant.
`sort`([function])	Trie le tableau selon l'ordre lexicographique (tri ascendant). Possibilité de définir l'opérateur de comparaison à l'aide d'une fonction
`splice`(index,n,[,el1,el2])	Ajouter/Enlever des éléments d'un tableau
`toSource`()	Retourne une chaîne qui représente la définition du tableau.
`toString`()	Retourne le tableau sous la forme d'une chaîne.
`unshift`(e1,e2,...)	Ajoute les éléments e1,e2, ... en tête du tableau et retourne la longueur du nouveau tableau.
`valueOf`()	Retourne une chaine décrivant les valeurs initiales du tableau. Notez que cette méthode est utilisable sur tous les objets (fonctions, chaines, ...).

2.6.5 Conversion de type

Notons que des fonctions pré-défines sont fournies permettant d'effecteur des conversions de types.

Converion de chaines en nombres Ce type de conversion est très utilisé en JavaScript car les données saisies par l'utilisateur ont toujours le type chaine de caractères. Les fonctions `parseInt()` et `parseFloat()` permettent de convertir explicitement une chaine en un entier ou en un réel respectivement. Ainsi, `parseFloat("13.56")` rend le le réel 13.56 (à partir de la chaine "13.56"). La fonction `parseInt` agit de même sur les entiers, mais accepte un deuxième argument optionnel qui est la base dans laquelle est exprimé l'entier rendu. Si on ne précise pas cette base, l'interpréteur décide de la base en fonction du format de la chaine : pour une chaine commençant par 0, il choisit par exemple la base 8, *e.g.* `parseInt("011")` rend 9. Il est par conséquent recommandé de préciser la base 10 en deuxième paramètre. On utilise par exemple `parseInt("0345",10)` pour obtenir l'entier 345.
Pour compléter la conversion vers un nombre, il est souvent nécessaire de s'assurer de la validité du résultat obtenu car il est parfois impossible de traduire une chaine en une expression numérique. Dans ce cas, le résultat de `parseInt` est `Nan` (*Not a Number*). Pour savoir si la chaine à convertir ne représente pas une expresson

numérique, on doit utiliser la fonction `isNaN()`, qui renvoie vrai dans ce cas.

Notons qu'on peut aussi utiliser `parseInt()` pour extraire la *partie entière* d'un réel : par exemple `parseInt(5.67)` rend 5. La partie décimale peut être extraite par soustraction de la partie entière (`5.67 - parseInt(5.67)` rend `0.67`).

Converion vers une chaine La fonction `toString()` permet de faire la conversion inverse. Elle est cependant plus générale que les précédentes car peut s'appliquer, non pas uniquement à des nombres, mais aussi à une chaine (rend la même chaine), à un booléen (rend "true" ou "false") à un tableau (rend le contenu des tableaux séparés par des virgules), ou encore à une fonction (rend la définition de la fonction).
Sur les nombres on utilise la fonction de la manière suivante :

```
var i = 15;
i.toString(10);
```

La fonction `toString()` accepte un argument optionnel qui est la base utilisée pour représenter le nombre en chaine. Dans l'exemple, on obtient simplement la chaine "15".

Chapitre 3

Exercices corrigés

Voici une série d'exercices permettant d'illustrer les notions évoquées dans ce chapitre. L'énoncé est suivi d'un programme solution en JavaScript, puis de commentaires sur la solution. Bien évidemment, il ne s'agit que d'une solution parmi différentes possibles.

3.1 Fonction simple

Utilisation de `function()`, `max()`.

Ecrire une fonction qui, étant donné trois nombres entiers non nul, a, b et c, détermine si l'un des nombres est égal à la somme des deux autres. La fonction renvoie ce nombre s'il existe, 0 sinon.

```
function sommededeux (a,b,c) {

  var m = Math.max(a,Math.max(b,c));

  if (a+b+c== 2*m)
       return(m);
  else
       return(0);
}
```

La solution proposée repose sur la constatation suivante : si $a = b + c$ alors
– a est supérieur à b et à c
– $a + b + c = 2a$
La difficulté est que nous ne savons pas qui de a, b ou c est le plus grand. On doit donc le déterminer : il n'existe pas en Javascript de fonction donnant le *maximum* d'un ensemble d'éléments, mais on peut utiliser la fonction `max` du module `Math` qui rend le maximum entre deux nombres.

Notons que cette solution est plus élégante et plus facilement adaptable que la solution qui consiste à tester les différentes possibilités. On peut effectivement écrire :

```
  if (a==b+c) return (a)
  else if (b==a+c) return (b)
       else if (c=a+b) return (c)
              else return (0);
```

mais le nombre de tests augmenterait rapidement si le problème se complixifiait un tant soit peu.

3.2 Calcul d'une moyenne

Utilisation de `for, Array, prompt()` et `parseInt()`.

Ecrire un programme qui calcule la moyenne d'un ensemble de notes. Le programme doit ouvrir une fenêtre pour demander le nombre de notes, puis ouvrir une nouvelle fenêtre pour entrer chacune des notes. Quand toutes les notes sont saisies, une fenêtre affiche la moyenne.

```
1       var note = new Array();
2       var moyenne = 0;
3       var nb;
4
5             nb = prompt("Entrer le nombre de notes");
6             nb = parseInt(nb,10);
7             for (var i=0;i<nb;i++){
8                note[i] = prompt("Entrer la note " + (i+1));
9                note[i] = parseInt(note[i],10);
10               moyenne += note[i];
11            }
12
13            moyenne = moyenne / nb;
14            alert("La moyenne est " + moyenne);
```

A la ligne 6, nous demandons, avec `parseInt()`, la conversion d'une chaine vers un entier, car la fonction `prompt()` rend une chaine, et nous voulons traiter ce qu'a saisi l'utilisateur comme une valeur numérique. Nous faisons de même à la ligne 9.

3.3 Calcul de distance

Utilisation de `for, Array, prompt()`.

Ecrire un programme qui, connaissant un ensemble de villes et la distance les séparant d'un point de référence, soit capable de dire, après que l'utilisateur ait saisi une distance parcourue depuis le point de référence, quelles villes auraient pu être atteintes.

```
1  var villes =   new Array("Bordeaux","Nantes","Lyon",
2                           "Marseille","Monptellier",
3                           "Paris","Rennes","Strasbourg");
4  var distance = new Array(950,850,450,800,1000,460,840,0);
5  var dist;
6
7       dist = parseInt(prompt("Entrer la distance parcourue : "),10);
8       for (var i=0;i<villes.length;i++)
9           if (dist>=distance[i])
10             alert("ville atteinte : " + villes[i]);
11
```

Supposons que le point de référence soit Strasbourg. Nous mettons alors dans la case 0 du tableau des distances, la distance séparant Strasbourg de la ville inscrite dans la case 0 du tableau des villes (Bordeaux). La boucle parcout ensuite le tableau des distances, et quand la distance parcourue est supérieure à l'éloignement de la ville, la ville correspondante est affichée.

3.4 Écriture de la fonction factorielle

Utilisation de `function, for`.

Ecrire une fonction qui, étant donné un paramètre n, supposé être un entier positif, calcule et retourne sa factorielle. Rappelons que la factorielle de n est notée $n!$, et définie par :

$$n! = \begin{cases} n \times n-1 \times n-2 \times \ldots \times 1 & \text{si } n > 0 \\ 1 & \text{si } n = 0 \end{cases}$$

En examinant la définition de factorielle, nous nous apercevons qu'il faut effectuer n multiplications. Nous allons procéder avec une boucle qui fera n itérations, en énumérant les entiers consécutifs de n à 1 à l'aide d'une variable i. Concernant le calcul maintenant, la première étape doit nous donner comme résultat n. A la deuxième, nous devons obtenir le résultat de $n \times n-1$. A la troisième, nous devons obtenir le résultat de $n \times n-1 \times n-2$, c'est-à-dire le résultat calculé à la deuxième étape, multiplié par $n-2$.
Par conséquent, nous allons mémoriser le résultat de chaque multiplication dans une variable, que nous appelons ici `res`. Nous initialisons cette variable à 1 (qui est l'élément neutre pour la multiplication) et obtenons ainsi n après la première multiplication. A la deuxième étape, `i` vaut $n-1$, et nous multiplions donc n par $n-1$, résultat que nous remettons dans `res`. A la fin de la boucle, nous avons fait les multiplications nécessaires, et le résultat final est présent dans `res`.
Notons, que si $n = 0$, nous ne rentrons pas dans la boucle, et `res` vaudra 1.

```
1  function factorielle( n ) {
2
3      var res=1;
4      for (i=n;i>=1;i=i-1) {
5          res = res * i;
6      }
7      return(res);
8  }
```

3.5 Écriture de la fonction est premier

Utilisation de `function, while, modulo`.

Ecrire une fonction qui, étant donné un entier a, supposé être un entier positif, dit si a est un nombre premier. Par définition, un nombre est premier s'il n'est divisible que par 1 et par lui-même.

Nous utilisons une variable `b` qui sert à stocker touts les entiers consécutifs entre $a-1$ et 2. Si le reste de la division entre a et cet entier est nul, cela signifie que cet entier est un diviseur de a et que a n'est donc pas premier (par exemple $a = 6$ et l'entier est 3). Remarquez la façon dont le reste d'une division entière est noté : `a`

`%` `b` désigne le reste de la division de a par b, et on le désigne habituellement par le terme *modulo*.

```
1  function est_premier( a ) {
2    var b;
3    var prem=true;
4
5    b = a-1;
6    while (b>1 && prem) {
7      if (a%b==0) prem = false;
8      b--;
9    }
10   return(prem);
11 }
```

3.6 Un rendeur de monnaie

Utilisation de la notion de *partie entière* (`parseInt()`).

Ecrire un programme qui, étant donné une somme s introduite (on suppose cette somme multiple de 0.1 €) dans une machine à café, et le prix d'un café c, indique la composition de la monnaie rendue. On dispose de pièces de 1 , 0.5 , 0.2 et 0.1 €. On cherche à rendre le moins de pièces possibles.

Les variables `p1,p50,p20,p10` utilisées dans la solution doivent stocker le nombre de pièces rendues pour chacune des valeurs. L'algorithme repose sur l'observation simple suivante : si je dois rendre une somme r avec des pièces de valeur v, il faut r/v pièces. Si le reste de cette division est nul, la monnaie est rendue, sinon, il reste à rendre $r' = r\%v$ (le reste de la division entière). Par définition, on peut aussi calculer r' comme ceci : $r' = r - (r/v)$. si r' n'est pas nul, on peut recommencer avec une valeur $v'(v' < v)$. Pour minimiser le nombre de pièces, on commence par compter le nombre de pièces nécessaires dans la plus forte valeur (1 €). Dans le programme ci-dessous, est noté en commentaire la solution alternative équivalente utilisant le modulo (ligne 8,10,12). Notez aussi qu'à la lgine 5, on s'est abstenu de diviser `a_rendre` par la valeur de la pièce car elle est de 1 (élément neutre, donc division superflue).

```
1    var p1, p50, p20, p10;
2    var a_rendre;
3
4    a_rendre = s - c;
5    p1 = parseInt(a_rendre);
6    a_rendre = a_rendre - p1
7    p50 = parseInt(a_rendre / .5);
8    a_rendre = a_rendre - p50 * 0.5   // a_rendre = a_rendre % 0.5;
9    p20 = parseInt(a_rendre / .2);
10   a_rendre = a_rendre - p20 * 0.2   // a_rendre = a_rendre % 0.2;
11   p10 = parseInt(a_rendre / .1);
12   a_rendre = a_rendre - p10 * 0.1   // a_rendre = a_rendre % 0.1;
13   document.write("pièce de 1   E : ",p1,"<br/>");
14   document.write("pièce(s) de 0.5 E : ",p50,"<br/>");
15   document.write("pièce(s) de 0.2 E: ", p20,"<br/>");
16   document.write("pièce(s) de 0.1 E: ", p10,"<br/>");
```

Notons que l'énoncé facilite le problème dans la mesure où l'on est toujours capable de rendre la monnaie. Le problème ne serait pas aussi simple si l'on n'avait pas de pièce de 0.1 €par exemple. Dans ce cas en effet, notre programme ne pourrait pas rendre la monnaie sur 1.80 €(il donnerait 1 €+ $1 \times 0.50 + 1 \times 0.20$ et il manquerait 0.10) alors qu'on pourrait rendre la monnaie avec 9×0.20.

3.7 Un rendeur de monnaie (général)

Utilisation de la notion de *partie entière* et de *tableau* .

Ecrire un programme qui, étant donné une somme s introduite dans une machine à café, et le prix d'un café c, indique la composition de la monnaie rendue. On dispose de n types de pièces de valeur stockées dans un vecteur $v = (v_1, \ldots v_i, v_{i+1}, \ldots, v_n)$. Les valeurs sont classées par ordre décroissant, i.e. $v_i > v_{i+1}$. Représenter la monnaie rendue par un vecteur $p = (p_0, \ldots, p_n)$ qui indique combien de pièces de chaque valeur il faut rendre. La somme rendue est $\sum_{i=1}^{n} p_i \cdot v_i$. On cherche à rendre le moins de pièces possibles.

```
1  var v = new Array(1, 0.5 ,0.2, 0.1);  /* par exemple */
2  var p = new Array();
3  var a_rendre;
4  var i;
5  var fenetre;
6
7  a_rendre = s - c;
8  fenetre = window.open("","monnaie","width=200,height=200");
9  fenetre.document.write("<html><body>");
10 fenetre.document.write("Monnaie à rendre : ",a_rendre,"<br/>");
11 for (i = 0; i < v.length  ; i=i+1) {
12     p[i] = parseInt(a_rendre / v[i]);
13     a_rendre = a_rendre - p[i]*v[i];
14 }
15 for (i = 0; i< v.length ; i=i+1)
16     fenetre.document.writeln("pièce de ",v[i]," E: ",p[i],"<br/>");
17 fenetre.document.writeln("</body></html>");
```

Chapitre 4

DHTML : JavaScript et HTML

L'une des raisons du succès de JavaScript est qu'il permet de construire des pages web *dynamiques*. Le terme *Dynamic HTML* (ou DHTML) est souvent utilisé pour désigner l'utilisation de javascript dans les pages HTML.

Une page web construite simplement avec du HTML ne propose à l'utilisateur que le lien hypertexte en guise d'interactivité. L'objectif de JavaScript est d'impliquer beaucoup plus activement l'utilisateur dans la consultation d'une page web. Nous allons utiliser dans ce chapitre les concepts de programmation présentés au chapitre précédent afin de les appliquer à une page de présentation HTML. Dans la pratique, l'utilisation de JavaScript revient quasiment toujours à modifier dynamiquement la structure ou le contenu de la page HTML initialement chargée, et nous verrons quelques exemples de cette utilisation.

4.1 Évènements

L'interactivité avec l'utilisateur suppose que le navigateur soit capable de réagir quand l'utilisateur s'agite (clic de souris sur un lien, retour à la page précédente, *etc*). Pour cela, des *évènements*, correspondant à des actions de l'utilisateur, ont été définis. La table 4.1 liste les évènements gérés par JavaScript et leurs significations. Chaque objet JavaScript réagit à certains évènements seulement, qui sont listés, objet par objet, en annexe A, page 49.

Afin de gérer un évènement, il est nécessaire de spécifier quel évènement on veut surveiller, et quelle action entreprendre quand l'évènement se produit. Ceci s'écrit à l'aide d'un attribut HTML. Par exemple, la ligne suivante :

```
<a href="page2.html" onClick="compte++">Page suivante</a>
```

indique que l'on gère l'évènement "clic de souris" sur ce lien. Quand l'évènement se produit, l'action à entreprendre est d'incrémenter d'une unité la valeur de la variable `compte`. Notons que l'on aurait pu mettre n'importe quel code JavaScript en guise d'action à entreprendre (les différentes instructions sont alors séparées par un point-virgule (`;`)). Si l'action est compliquée, on appelle habituellement une fonction.

Evénement	Se produit quand ...
onAbort	le chargement d'une image a été interrompu
onBlur	la souris quitte une zone formulaire
onClick	on clique sur un formulaire ou un lien
onChange	on change une zone texte ou une sélection d'élément
onFocus	la souris entre dans une zone formulaire
onLoad	on charge la page dans le navigateur
onMouseOut	la souris sort d'une zone imagemap
onMouseOver	la souris passe au-dessus d'un lien ou d'un formulaire
onReset	on vide un formulaire
onSelect	on sélectionne un élément dans un formulaire
onSubmit	on envoie un formulaire
onUnload	on sort de la page

TAB. 4.1 – Nom et descriptions des évènements en JavaScript

Note : *Les noms d'évènements sont ici ceux de la définition initiale, avec une majuscule après* `on` *(comme dans* `onClick`*). Dans les recommandations courantes de HTML, ces noms ne sont plus sensibles aux majuscules/minuscules. On trouve maintenant écrit* `onclick` par exemple.

Les différents objets HTML sont donc sujets à des évènements particuliers. Etant donné une balise désignant un objet HTML, seuls quelques évènements parmi ceux listés dans la table 4.1 peuvent survenir. Autrement dit, on ne peut pas associer la surveillance de n'importe quel évènement dans le code HTML.

Par exemple, l'évenement `onSubmit` (validation d'un formulaire) ne peut concerner qu'un formulaire. La surveillance de cet évènement est insérée dans le HTML de manière similaire à ceci :

```
<form id="formu" onSubmit="fait_qquechose()">
....
</form>
```

De même, seul l'évènement `onClick` peut être associé à un bouton radio. On remarquera également que la balise `<input type="passwd">` utilisée pour des saisies confidentielles ne génèrent jamais d'évènement. Cette décision à été prise par les concepteurs du langage afin de ne pas permettre à des gens mal intentionnés de récupérer des mots de passe.

La table 4.2 indique quels évènements concernent quelles balises.
Nous verrons tout au long du chapitre, à travers les exemples, comment sont utilisés ces évènements. Mais avant cela, nous devons aborder la notion d'objet utilisée entre autres pour structurer les éléments d'une page web.

4.2 Notation orientée objet

JavaScript est un langage orienté objet. Je ne développe pas ici les concepts de la programmation objet, mais précisons certains termes et idées afin de comprendre

Balise	Evénement
`<a>`	onClick onMouseOver onMouseOut
`<area>`	onMouseOver onMouseOut
`<body>`	onBlur onFocus onLoad onUnload
`<form>`	onReset onSubmit
`<img>`	onAbort onLoad onError

`<input type="button">`
`<input type="checkbox">`
`<input type="radio">`
	onClick

`<input type="text">` `<textarea>`	onBlur onChange onFocus
`<input type="select">`	onSelect onBlur onChange onClick onFocus

TAB. 4.2 – Les évènements associés aux balises

les notations utilisées.

4.2.1 Classes et objets

Une *classe* est la description d'un ensemble de *propriétés* et de *méthodes*. Les propriétés sont les données, et les méthodes sont les fonctions qui manipulent ces données. Un *objet* est une instance de classe.
On pourrait par exemple définir la classe voiture, dont les propriétés seraient l'immatriculation, le modèle, la marque, la couleur, le propiétaire, etc. Les méthodes pourraient alors être repeindre() qui changerait la propriété couleur, revendre() qui changerait la propriété propriétaire, etc. On pourrait définir deux instances v_1 et v_2 de la classe voiture. Ces deux instances possèdent automatiquement les propriétés et les méthodes de la classe voiture. On peut ensuite appliquer à chacun de ces objets les méthodes qui ont été définies pour cette classe. Par exemple, v1.revendre() et v2.revendre() modifieront les propriétaires respectifs de ces deux objets de la classe voiture.

On peut considérer que la notion d'objet est une généralisation de la notion de type décrite en section 2.6, page 17. Les objets possèdent en effet des propriétés supplémentaires, comme la notion de classe et d'héritage. Nous n'utiliserons pas ces notions, et par conséquent nous pouvons considérer une classe d'objet comme un type dans ce qui suit.

Par conséquent, nous retrouvons naturellement, parmi les classes d'objets prédéfinies dans le langage, les classes :

`Array`	: les tableaux
`Boolean`	: les booléens
`Date`	: les dates
`Number`	: les valeurs numériques (entiers ou réels)
`String`	: les chaines de caractères

Par exemple, la classe `String` englobe les objets de type chaine de caractères. Créer un objet de cette classe se fait à l'aide de l'opérateur `new`. La déclaration suivante :

```
var texte1 = new String;
var texte2 = new String("bonjour ");
```

crée deux objets de noms `texte1` et `texte2` appartenant tous deux à la classe `String`. Notons que `texte2` à été initialisé simultanément à sa déclaration.

Les propriétés des objets

Pour accéder à une propriété p d'un objet o, on utilise la notation $o.p$. Par exemple, l'objet `texte2` déclaré précédemment appartient à la classe `String`. Or tous les objets de cette classe ont la propriété `length` qui définit la longueur de la chaine mémorisée dans cette objet. On peut donc faire afficher la longueur par :

```
document.write(texte2.length);
```

4.2.2 Les méthodes des objets

La même notation s'applique aussi pour exécuter des méthodes des objets. Là aussi, il est nécessaire de connaître le noms des méthodes existantes pour les objets prédéfinis. Toujours pour la classe `String`, la méthode pré-définie `concat()` permet de concaténer (mettre bout-à-bout) deux chaines. Si, dans la continuité de l'exemple précédent j'écris :

```
texte1 = texte2.concat(" monsieur");
```

alors le contenu de l'objet `texte1` devient "bonjour monsieur", c'est-à-dire la concaténation de la chaine contenue dans `texte2` et de la chaine " monsieur".

4.3 Les objets DHTML

4.3.1 Modèles DHTML et DOM

Quand une page web est chargée, le navigateur créé plusieurs objets pour représenter les informations y figurant. Les premières versions de DHTML ont mis en place un modèle d'objets présenté ci-dessous et toujours en cours. Plus récemment, un modèle plus général à été proposé par le W3C : c'est le *Document Object Model* (DOM). Il propose un ensemble de méthodes et d'interfaces standarts permettant de décrire, parcourir et modifier de façon uniforme des documents HTML mais aussi XML. Pour les documents HTML, il est donc proposé une représentation conforme au DOM [HTM03].

Quelque soit le modèle de document utilisé, les principes sont proches et les désignations d'objets totalement compatibles. La seule conséquence actuellement, est qu'il y a

souvent plusieurs manières de procéder. On peut aussi noter que les méthodes proposées par le DOM sont plus puissantes car permettent de modifier tous les éléments du document (par exemple supprimer un morceau de texte HTML).

4.3.2 Les objets instanciés

Quand une page web est chargée, le navigateur créé plusieurs objets pour représenter les informations y figurant. Ces objets sont classés de manière hiérarchique, l'objet le plus haut dans la hiérarchie (appelé aussi racine car la hiérarchie est arborescente) étant un objet de la classe `Window`. Le schéma ci-dessous (figure 4.1) montre l'organisation des différentes classes dans le modèle d'objets DHTML. On peut interpréter ce schéma comme ¡¡un objet `window` contient éventuellement des frames, une adresse, un document¿¿ et ¡¡un document contient lui-même éventuellement des formulaires et des liens¿¿, etc.

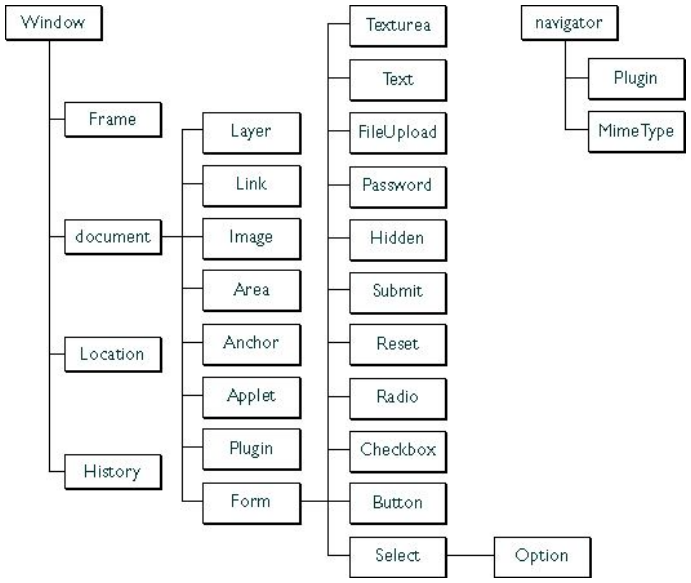

FIG. 4.1 – Les classes d'objets DHTML

Les noms choisis par le navigateur pour instancier les objets des différentes classes sont les mêmes que ceux de la classe mais typographiés en minuscule. Quand plusieurs objets appartiennent à une même classe, ils sont regroupés dans un tableau et peuvent être distingués par leur place dans le tableau.
Ainsi, l'objet (unique) de la classe `Window` s'appelle `window` et les différents objets cadres (éventuellement plusieurs dans une fenêtre) s'appellent `window.frames[0]`, `window.frames[1]`, etc.

Note : *Etant donné qu'on accède très souvent à des objets de la classe* `window`*, le langage permet, pour des raisons de commodité, d'ommettre le mot* `window` *quand on désigne un objet faisant partie de la hiérarche de* `window` *—il s'agit donc des objets* `location`*,* `history` *et* `document`*. Les deux déclarations suivantes sont donc équivalentes, la deuxième étant simplement plus concise :*

```
var url = window.location.href;
var url = location.href;
```

4.3.3 Comment désigner les éléments HTML

Pour chacun des éléments HTML figurant dans une page (c'est-à-dire ceux définis par des balises telles `<img>` pour une image), le navigateur les range dans la hiérarchie vue dans la section précédente. Pour chaque objet HTML nommé avec l'attribut `name` ou `id`, il créé un objet JavaScript correspondant de même nom.

Prenons le document HTML très simple suivant :

```
<html>
<head></head>
<body>
      <img id="header" src="header.jpg" alt="image">

      <form  id="quest">
            <input id="email" type="text"><br>
            <input type="submit">
      </form>

      <img id="footer" src="greyfoot.jpg" alt="image">
</body>
</html>
```

Vu dans un navigateur, cet exemple pourrait ressembler à l'image représentée figure 4.2 ci-dessous.

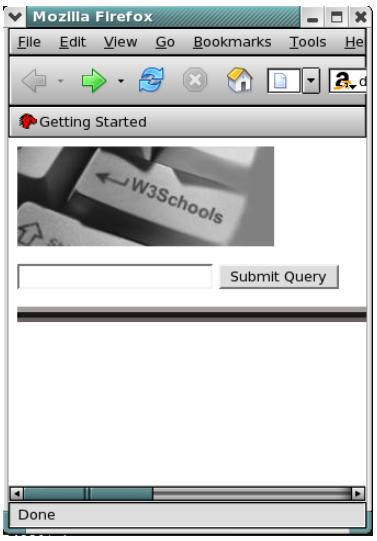

FIG. 4.2 – L'exemple HTML rendu par un navigateur

Ici `<img>` et `<form>` sont des balises HTML qui sont représentées dans la classification de la figure 4.1 par les classes Image et Form respectivement. Pour les deux balises `<input>` on remarque que leur type est aussi référencé dans la classification par les class Text et Submit.

En suivant les branches de la figure 4.1, on déduit l'objet qui à été créé pour représenter chacun de ces éléments HTML. On part de `Window`, on passe par Document et on arrive au niveau Image et Form.

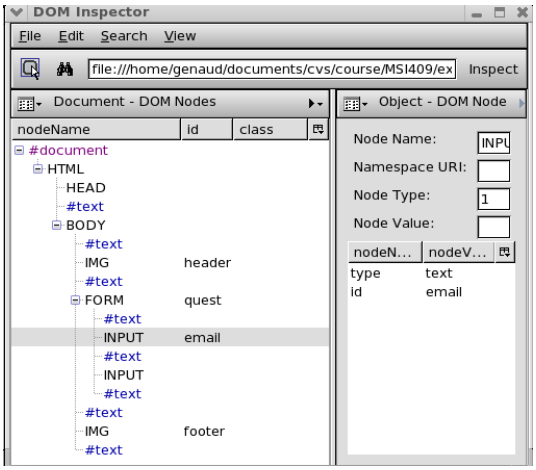

FIG. 4.3 – L'inspecteur DOM de Mozilla-Firefox montre la hiérarchie des objets

Certains outils comme l'inspecteur DOM du navigateur Mozilla-Firefox sont capables d'afficher comment sont hiérarchisés les éléments par le navigateur (fig. 4.3).

Désignation par le nom Comme des noms on été donnés[1] à ces éléments, on peut désigner en javascript ces éléments par (respectivement) :

```
window.document.bandeau
window.document.quest
window.document.footer
```

En suivant la hiérarchie, on peut aussi désigner le seul champ `<input>` portant un nom à l'intérieur du formulaire en suivant la même logique.

```
window.document.quest.email
```

Cette notation est concise et largement utilisée, mais n'est pas la seule.

Désignation par tableau Dès qu'une classe peut comporter plusieurs objets, le navigateur crée un tableau pour chaque classe avec les différents objets. Dans chaque tableau les objets sont numérotés dans leur ordre d'apparition dans la page. Le nom du tableau est celui de la classe, mis au pluriel (e.g. images, forms, anchors, links, ...). Ainsi, on peut aussi désigner les éléments précédents par leur numéro d'ordre dans la classe. Aussi les expressions suivantes sont équivalentes (i.e. désignent les mêmes objets).

[1] On a utilisé `id` mais le résultat aurait été le même avec la balise (obsolète) `name`.

```
window.document.bandeau     = window.document.images[0]
window.document.quest       = window.document.forms[0]
window.document.quest.email = window.document.forms[0].email
window.document.footer      = window.document.images[1]
```

Cette écriture est cependant risquée : si l'élément HTML change de place, sa désignation en javascript risque d'être fausse. Etant donné que javascript supporte les tableaux associatifs, il est recommandé de désigner l'élément de tableau à l'aide de son nom, et pas de son numéro. De manière équivalente, on peut écrire :

```
window.document.bandeau     = window.document.images['bandeau']
window.document.quest       = window.document.forms['quest']
window.document.quest.email = window.document.forms['quest'].email
window.document.footer      = window.document.images['footer']
```

Eléments Notons que les classes en dessous de Form ne sont pas distinguées. (Il n'y a pas de tableau d'objets texts par exemple). Tous les objets sous un objet Form sont rangés dans un tableau `elements`, quelque soit leurs types.
On peut donc utiliser ces différentes notations pour notre champ de texte :

```
window.document.quest.email
=  window.document.forms['quest'].elements[0]
=  window.document.forms['quest'].elements['email']
```

Méthodes DOM La recommandation DOM propose des méthodes pour accéder aux objets. Parmi celles-ci on a :
- `getElementById()` et `getElementsByName()` qui rend un objet correspondant à l'élément HTML désigné par son attribut `id` ou `name` respectivment,
- `getElementsByTagName()` qui rend un tableau des objets représentant une balise. Par exemple, `getElementsByTagName("a")` donne un tableau comportant les hyper-liens.

Voici quelques exemples d'équivalence avec les notations précédentes :

```
window.document.quest.email = document.getElementById('email')
window.document.images      = document.getElementsByTagName('img')
```

4.3.4 Désignation de l'objet courant : `this`

Il est souvent commode de pouvoir désigner l'objet manipulé sans devoir le nommer de manière explicite, comme nous l'avons décrit précédemment. Il existe le mot clé `this` qui permet de désigner l'objet en cours de manipulation. Illustrons ceci par un exemple :

```
<html> <head>
<script type="text/javascript">
function testResults (form) {
          var TestVar = form.elements['champ'].value;
          alert ("Vous avez tapé : " + TestVar);
}
</script> </head>
<body>
```

```
<form id="monformulaire" method="get" action="">
<p>Saisir quelquechose dans le champ :<br>
<input type="text" id="champ"><br>
<input type="button" id="bouton" value="Click"
       onClick="testResults(this.form)">
</p></form>
</body></html>
```

Dans la déclaration du formulaire HTML (lignes 9 à 14), on remarque l'insertion de l'évènement Javascript `onClick`, qui indique qu'on doit appeler la fonction `testResults()` (définie lignes 3 à 6) lorsque le bouton à été cliqué. On désire passer comme argument à cette fonction le formulaire `monformulaire`. On pourrait désigner cet objet par son nom complet, ce qui donnerait :
`onClick="testResults(document.forms['monformulaire'])"`. Cependant, il est aussi possible de le désigner, comme c'est le cas, par `this.form`. Cette désignation est en effet plus générale, puisqu'elle est valable quelquesoit le nom du formulaire.

4.4 Utilisation des objets

Nous passons en revue dans ce chapitre, quelques uns des objets les plus utilisés, en donnant des exemples dans lesquels ils ont utilisés. Vous trouverez plus de détails concernant chacune des classes dans l'annexe A où les propriétés et les méthodes des classes sont listées de manière exhaustive.

4.4.1 La classe `Window`

Cette classe fournit des méthodes utiles d'affichage, parmi lesquelles on trouve des fenêtres pop-up de dialogue :

`alert()`	affiche une fenêtre de dialogue
`confirm()`	affiche une fenêtre de dialogue avec boutons 'OK' et 'Cancel'
`prompt()`	affiche une fenêtre de dialogue avec une zone texte

Méthode `alert()` La méthode `alert()` est très utile pour le déboggage de code : on peut utiliser cette méthode pour faire afficher les valeurs des variables. La figure 4.4 illustre l'apparition d'un fenêtre pop-up de dialogue après l'appel à la méthode `alert()`.

FIG. 4.4 – Pop-up `alert()`

Cette fenêtre est obtenue grâce à la ligne de code suivante :

```
window.alert("Texte affiché par window.alert()");
```

D'autres méthodes permettent d'ouvrir et de fermer de nouvelles fenêtres ou de changer leur apparence. Une autre méthode très utile est la méthode `setTimeout()`. Elle permet d'exécuter un code JavaScript après un laps de temps donné (indiqué en millisecondes). Par exemple :

```
setTimeout("alert('bonjour');", 500);
```

fera apparaître la fenêtre de dialogue une demi-seconde après le chargement de la page.

Méthode `confirm()` Cette méthode permet de poser une question à l'utilisateur, qui répondra en appuyant soit sur "Ok" soit sur "Annuler". Si "Ok" est appuyé, la fonction retourne `true`, et si "Annuler" est appuyé, elle rend `false`.

Méthode `prompt()`

`String s = prompt()`(*message* [,*initial*,]))

Cette méthode permet à l'utilisateur, de lire une question, spécifiée par la chaine *message*, de saisir une valeur et d'appuyer ensuite sur "Ok". On peut spécifier une valeur par défaut à l'aide du deuxième paramètre optionnel *initial*. La fonction rend la valeur saisie sous la forme d'une chaine de caractère. Si le programme attend une valeur numérique, il faut la convertir avec des fonctions comme `parseInt()` que nous avons vu en section 2.6.5, page 20.

Dans l'exemple suivant, on demande à l'utilisateur son age, en supposant que 20 est une valeur qui fera l'affaire dans de nombreux cas. On convertit ensuite la réponse sous forme de chaine en une valeur numérique.

```
1 reponse = prompt("Quel est votre age ?","20");
2 age = parseInt(reponse,10);
```

Méthode `open()` Cette méthode permet d'ouvrir une nouvelle fenêtre de navigateur. Ce type d'action est utilisé en général pour afficher des informations annexes qui ne doivent pas interférer avec la fenêtre principale. Le prototype de la fonction est :

`window w = open()`(*URL* ,*titre* [,*options*,]))

URL	une chaine spécifiant l'URL à charger l'ouverture de la fenêtre. L'URL peut être un fichier ou une page internet.
titre	une chaine donnant un titre à la fenêtre.
options	une chaine contenant une liste de mots-clés, séparés par des virgules (ne pas insérer d'espaces dans cette chaine), qui indiquent quelle apparence doit avoir la nouvelle fenêtre, comme par exemple sa taille. Suit quelques unes des options qu'il est possible de spécifier.

height	hauteur de la fenêtre en pixels
width	largeur de la fenêtre en pixels
scrollbars	créé des ascenseurs de défilement si vaut *yes*
status	affiche une barre d'état si vaut *yes*
toolbar	créé la barre d'outil avec boutons de navigation
resizable	autorise le redimensionnement de la fenêtre si *yes* spécifié.
location	crée la fenêtre permettant de saisir l'adresse

Si on ne spécifie pas d'*URL* la fenêtre ouverte sera vierge. Si on ne précise pas d'options, certaines options seront activées ou non, automatiquement selon le contexte (place disponible pour afficher la barre d'outils, etc). Voici deux exemples :

```
if (confirm("ouverture w1 ?"))
  w1 = window.open("http://www.google.com","","toolbar=false");
if (confirm("ouverture w2 ?")) {
  w2 = window.open("","nouvelle","width=200,height=200");
  w2.document.write("<html><body><h1>Bonjour</h1></body></html>");
}
```

La ligne 2 permet d'ouvrir dans une fenêtre sans barre d'outil, la page d'un moteur de recherche. La ligne 4 ouvre une fenêtre vierge, de petite taille, dans laquelle on imprime un message à la ligne 5.

Les cadres (frames) Les *cadres* sont representés par des objet de la classe `Window`. Quand une page contient des cadres, l'objet de cette classe contient un tableau de tels objets. A l'intérieur d'un cadre, on peut faire référence à la fenêtre le contenant à l'aide de `window.parent` ou simplement `parent`. Pour des cadres imbriqués doublement, on utilise `window.parent.parent` pour accéder à la fenêtre originelle ou simplement `window.top`.

4.4.2 La classe `Document`

L'objet `document` est probablement le plus utilisé. Les propriétés de cet objet font référence au contenu de votre page web. Parmi celles ci, on trouve :

`bgColor`	la couleur de fond
`fgColor`	la couleur du texte
`lastModified`	une chaine donnant la date de dernière modification
`images`	un tableau d'objets Image
`forms`	un tableau d'objets Form
`links`	un tableau d'objets Link

La plupart de ces propriétés sont initialisées avec les balises HTML. La propriété `bgColor` par exemple, est celle qui est spécifiée par l'attribut déprécié bgcolor dans la balise `<body>` de la page.

4.4.3 La classe `Image`

Les informations concernant les images peuvent être stockées par un objet de cette classe. Les propriétés de cet objet recensent entre autres l'URL de l'image, son état de chargement, ses dimensions, etc.
Pour illustrer simplement la manipulation d'images, voici quelques lignes permettant de changer l'image quand le pointeur de la souris passe au-dessus. En l'occurence, on veut que l'image du point vert (greenball.gif) se transforme en une flèche jaune (right.gif) quand la souris passe sur l'image. Bien sûr, il faut gérer l'évènement qui se produit quand la souris passe sur l'image. Pour cela, on écrit dans la partie HTML le code suivant, en supposant que notre image est la première de la page.

```
<a href="#"
  onMouseOver="document.images[0].src='right.gif'"
  onMouseOut="document.images[0].src='greenball.gif'">
  <img id="image1" src="greenball.gif" alt="">
</a>
```

Nous avons ici deux instructions HTML : celle qui crée un lien et celle qui crée une image. L'instruction de création de lien comporte une instruction JavaScript qui charge les fichiers right.gif ou greenball.gif dans la place réservée à l'image en fonction de l'évènement. Il est important de définir la gestion de l'évènement avant de créer l'image par la balise `<img>`.

4.4.4 La classe `Form`

Un objet de type `form` est utilisé pour décrire les propriétés et traitements inhérents à un formulaire HTML. Cette classe est celle qui contient le plus grand nombre de classes dérivées, comme le montre la figure 4.1. En effet, le formulaire peut contenir de nombreux objets de types différents, comme des champs de saisie, des boutons à sélection multiples, etc.

Pour illuster simplement la manipulation de formulaire, essayons de contrôler la saisie d'un formulaire comme celui de la figure 4.5.

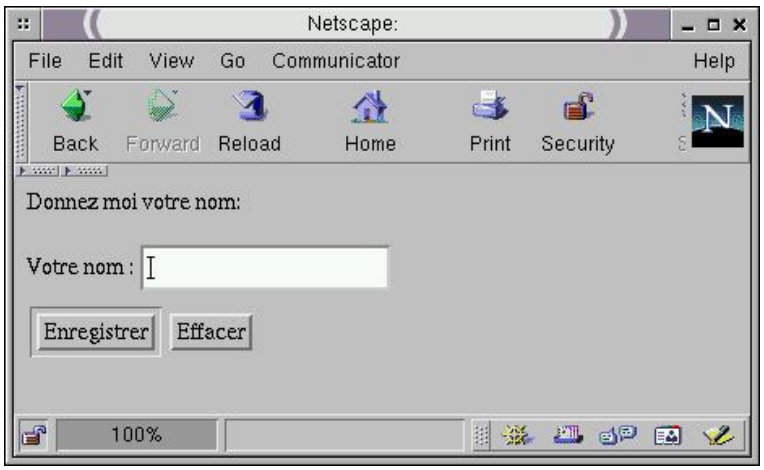

FIG. 4.5 – Un formulaire d'enregistrement

On veut interdire à l'utilisateur de valider le formulaire si il n'a saisi aucun caractère dans le champ nom.
Nous définissons donc le formulaire correspondant en HTML avec la balise ¡`form`¿.
On donne le nom `formulaire` à ce formulaire et on associe une action JavaScript à exécuter lorsque l'utilisateur validera le formulaire. En l'occurence on exécute une fonction que l'on a choisi d'appeler `verifie()`.
Notons qu'il est nécessaire de retourner la valeur faux au formulaire (d'où la présence de `return verifie()` ligne 5) pour que l'attribut `action` ne soit pas réalisé.
Pour cette vérification, nous aurons besoin de savoir ce que l'utilisateur a saisi dans le champ `nom` du formulaire.

```
1
2  <html>
3  <body>
4  Donnez moi votre nom:
5  <form id="formulaire" onSubmit="return verifie()"
6  action="merci.html" method="get">
7  <p>
8  Votre nom : <input id="nom" type="text"  value=""> <br>
9
10 <input type="submit" value="Enregistrer">
11 <input type="reset"  value="Effacer">
12
13 </p>
14 </form>
```

```
15 </body>
16 <html>
```

Le formulaire défini, il faut écrire en quoi consiste la vérification, c'est-à-dire écrire la fonction `verifie()`. Comme d'habitude, nous plaçons la définition de la fonction dans la partie `<head>` du document HTML.
La vérification consiste ici à vérifier que dans le formulaire, le champ `nom` n'est pas une chaine de caractères vide. Si c'est le cas, une fenêtre pop-up affiche un message d'erreur, le curseur revient dans le champ `nom`, et la fonction renvoie la valeur faux pour indiquer que la saisie n'est pas correcte. Dans le cas contraire, la fonction renvoie vrai et le formulaire affichera la page de remerciements pour avoir rempli correctement le formulaire.

```
1  <html>
2  <head>
3  <script type="text/javascript">
4  function verifie() {
5      if (document.forms['formulaire'].elements['nom'].value == "") {
6        document.forms['formulaire'].elements['nom'].focus();
7        alert("Le nom est obligatoire");
8        return false;
9      }
10     else return true;
11 }
12 </script>
13 </head>
```

4.4.5 Quelques éléments de la classe `Form`

Nous avons vu dans l'exemple du formulaire précédent qu'il était composé de trois types d'éléments différents : le type `input`, pour le champ `nom`, le type `submit` et le type `reset`. La balise ¡`form`¿ du langage HTML permet d'utiliser d'autres éléments. Tous les éléments qui peuvent composer un formulaire apparaissent dans la figure 4.1. Examinons maintenant comment nous pouvons manipuler certains de ces objets.

Les objets `select`

On peut par exemple créer des menus avec des éléments de type `select`. Ci-dessous, nous créons un formulaire permettant de choisir un système d'exploitation. Nous prévoyons d'ores et déjà d'appeler la fonction `choix()` lorsque l'utilisateur aura choisi un élément de la liste.

```
1  <form>
2    Systeme d'exploitation :
3    <select id="liste" onChange="choix();">
4       <option value="Vous avez choisi BeOs">BeOs</option>
5       <option value="Vous avez choisi Linux">Linux</option>
6       <option value="Vous avez choisi MacOs">Macintosh</option>
7       <option value="Vous avez choisi Solaris">Solaris</option>
8       <option value="Vous avez choisi Windows">Windows</option>
9    </select>
10 </form>
```

Ce formulaire nous permet d'obtenir l'écran de la figure 4.6. On voit à gauche l'aspect initial du menu, et à droite son apparence lorsqu'on clique dessus.

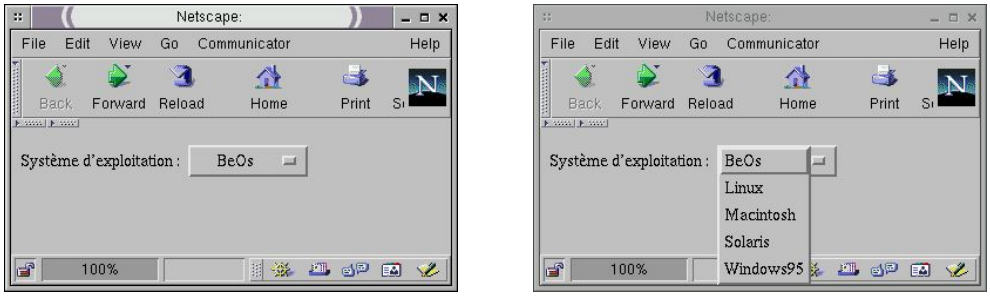

FIG. 4.6 – Un formulaire avec menu `select`

Maintenant nous voulons afficher le choix de l'utilisateur. Par conséquent, il faut retrouver l'élément choisi dans la liste. Notre formulaire n'a pas de nom mais comme c'est le seul (donc le premier), on peut le désigner par `document.forms[0]`. Nous avons donné un nom à notre menu : il s'appelle `liste` et nous pouvons le désigner par `document.forms[0].liste`. Ce type de menu possède des propriétés définies : la propriété `selectedIndex` conserve le numéro de l'élément choisi (les éléments sont numérotés du haut vers le bas, et le premier est numéroté 0). Les différents attributs `value` sont eux stockés dans la propriété `options`, qui est un tableau de chaines. Nous avons là les informations nécessaires pour afficher un message correspondant au choix fait par l'utilisateur. On écrit la fonction :

```
1  function choix() {
2       var i,s;
3       i = document.forms[0].elements['liste'].selectedIndex;
4       s = document.forms[0].elements['liste'].options[i].value;
5       window.confirm(s);
6  }
```

Une fois déterminé le message de `value` du formulaire associé avec le choix fait, on demande confirmation à l'utilisateur. Cette confirmation est illustrée par la figure 4.7.

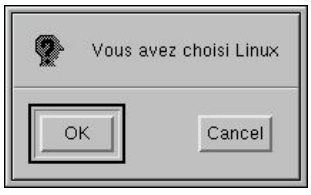

FIG. 4.7 – Fenêtre affichant la sélection et demandant confirmation

Les objets `radio`

Les boutons radio sont des boutons permettant un choix unique parmi plusieurs options. Graphiquement, le fait de cliquer sur un des boutons entraîne la dé-sélection des autres boutons. La figure 4.8 présente un exemple de choix par boutons radio. Dans cet exemple, nous voulons également afficher le choix fait par l'utilisateur après qu'il ait appuyé sur le bouton "Afficher choix".

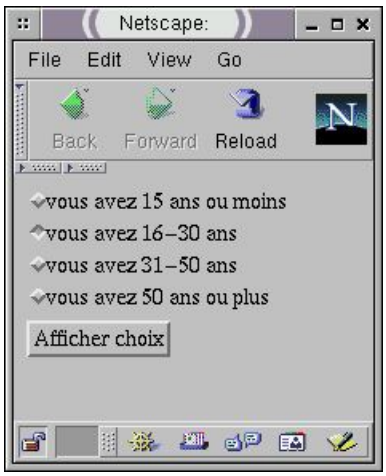

FIG. 4.8 – Sélection par bouton `radio`

Le bouton radio fait partie du langage HTML : comme le bouton `select`, il est défini dans un formulaire HTML. Pour créer l'exemple de la figure 4.8, nous avons décrit le formulaire :

```
<form id="formu_age" action=""><p>
<input name="age" type="radio" value="15-">vous avez 15 ans ou moins<br>
<input name="age" type="radio" value="16-30">vous avez 16-30 ans<br>
<input name="age" type="radio" value="31-50">vous avez 31-50 ans<br>
<input name="age" type="radio" value="50+">vous avez 50 ans ou plus<br>
<input type="button" value="Afficher_choix" onClick="afficheChoix();">
</p></form>
```

Il est important que les noms donnés aux boutons dans le formulaire soient tous les mêmes, car c'est parmi ces boutons de même nom que le choix unique est autorisé. La dernière ligne du formulaire, associe au bouton permettant de valider le choix, la fonction JavaScript appelée quand on appuie sur le bouton. Cette fonction permet d'afficher le texte correspondant au bouton sélectionné.
Pour déterminer quel bouton est sélectionné, le code décrit une boucle vérifiant tour à tour pour chacun des boutons, si la propriété `checked` est vraie. Si c'est le cas, on ouvre une fenêtre affichant le texte associé au bouton. Le code de la fonction est le suivant :

```
function afficheChoix() {
var nb = document.forms['formu_age'].age.length;
    for (var i=0;i< nb;i++) {
            if (document.forms['formu_age'].age[i].checked)
                alert(document.forms['formu_age'].age[i].value)
    }
}
```

Les objets `checkbox`

Les `checkbox` permettent de sélectionner plusieurs éléments dans une liste (choix multiple). En HTML, le formulaire présenté en figure 4.9 est obtenu par l'extrait de texte suivant :

```
1  Choisissez vos options :
2
3  <form id="opt" action=""><p>
4  <input type="checkbox" name="choix" value="radio">
5                         Auto-radio (1800 F)<br>
6  <input type="checkbox" name="choix" value="p. métal.">
7                         Peinture métalisée (3000 F)<br>
8  <input type="checkbox" name="choix" value="jantes">
9                         Jantes alliages (2500 F)<br>
10 <input type="checkbox" name="choix" value="clim.">
11                        Climatisation (7000 F)<br>
12 <input type="button" value="Total options" onClick="afficheTotal();">
13 </p></form>
```

Notons que tous les éléments `checkbox` portent le même nom. Comme précémment, ceci permet au navigateur de contruire un tableau (vecteur) regroupant tous ces objets et portant le nom `choix[]`. On note aussi, à la dernière ligne du formulaire, l'appel à la fonction `afficheTotal()`, qui sera chargée de faire le total du prix des options sélectionnées par l'utilisateur.

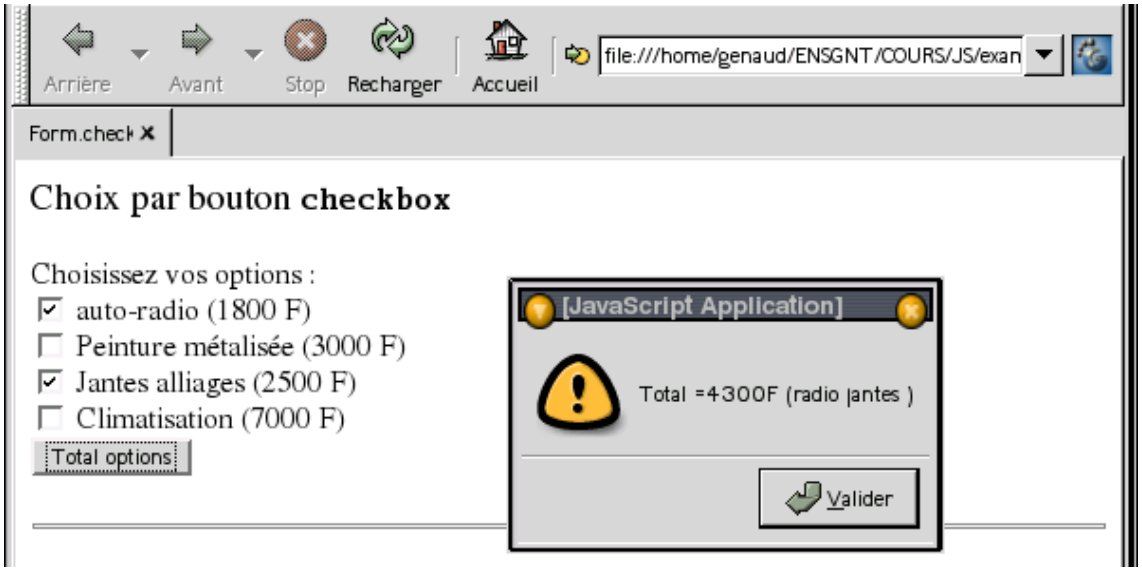

FIG. 4.9 – Sélection dans un formulaire à l'aide de cases `checkbox`

Le code de la fonction illustre l'accès au contenu des cases `checkbox` du formulaire. On détermine le nombre `nb` de cases de nom `choix` dans le formulaire nommé `opt` (ligne 4). Puis, on peut parcourir le tableau des cases, et pour chaque case (`document.opt.choix[i]`), vérifier si la case à été cochée (ligne 8). Il faut utiliser ici la propriété de type booléen `checked` de l'objet `checkbox`. Si c'est le cas, on cumule le prix de cette option stockée dans le tableau `tarif` dans le même ordre (choix arbitraire) que les cases (ligne 10). On ajoute aussi l'intitulé de cette option au message récapitulatif final (ligne 9).

```
1  <script type="text/javascript">
2  function afficheTotal() {
3  tarif = new Array(1800, 3000,2500,7000);
```

```
4    var nb = document.forms['opt'].choix.length;
5    var somme = 0;
6    var recap = "";
7      for (var i=0;i< nb;i++) {
8          if (document.forms['opt'].choix[i].checked)  {
9                recap += document.forms['opt'].choix[i].value + " ";
10               somme += tarif[i];
11         }
12       }
13       alert("Total =" + somme + "F (" + recap +")");
14     }
15     </script>
```

Annexe A

Objets - manuel de référence

A.1 anchor

Une ancre (anchor) est un repère dans une page. L'utilité de l'ancre est de pouvoir amener l'utilisateur directement sur la zone de la page marquée par ce repère. Ceci n'est généralement utilisé que quand la page est longue. zone d'une page. An example of an anchor would be

Syntaxe `<A NAME="mon-ancre">`
qui fait référence à l'ancre définie par `<A HREF="#mon-ancre">`

Propriétés aucune

Méthodes aucune

A.2 array

Les tableaux servent à stocker d'autres objets. comme par exemple plusieurs champs dans un formulaire. Les différents éléments sont rangés dans un tableau dans l'ordre où ils apparaissent.

Propriétés length

Méthodes aucune

A.3 button

L'objet button ainsi que ses propriétés sont définis dans un formulaire. La propriété value est le texte apparaîssant sur le boutton.

Syntaxe `<INPUT TYPE="button" name="Name" value="value">`

Propriétés name et value

Méthodes click()

Evènements onBlur, onClick, onFocus, onMouseDown, onMouseUp

A.4 checkbox

Une checkbox a l'apparence d'une boîte carrée qu'on sélectionne ou déselectionne par un clic de souris. Les checkbox sont définies à l'aide de la balise `<FORM>`.

Syntaxe `<INPUT TYPE="checkbox" name="Name" value="value">`

Propriétés checked, defaultChecked, name, and value

Méthodes blur(), click(), focus()

Evènements onBlur, onClick, onFocus

A.5 Date

L'objet date est un objet doté de nombreuses méthodes de manipulation. JavaScript mémorise les dates comme le nombre de millisecondes (ms) écoulées depuis le 1er Janvier 1970.

Propriétés aucune

Méthodes

getDate()	rend le jour du mois de la date courante
getDay()	rend le jour de la semaine de la date courante
getHours()	rend le jour de la semaine de la date courante
getMinutes()	rend les minutes de la date courante
getMonth()	rend le mois de la date courante
getSeconds()	rend les secondes de la date courante
getTime()	rend la valeur numérique de la date courante
getTimezoneOffset()	rend le décalage horaire en minutes avec GMT
getYear()	rend l'année de la date courante
parse(d)	rend le nombre de ms écoulées entre d et le 1/1/70
setDate(j)	met à la jour la date au jour j
setHours(h)	met à la jour la date au jour h
setMinutes(m)	met à la jour la date à la minute m
setMonth(m)	met à la jour la date au mois m
setSeconds(s)	met à la jour la date à la seconde s
setTime(n)	met à jour la date étant donné le nombre n de ms depuis le 1/1/70
setYear(d)	met à jour l'année pour une date d
toGMTString()	convertit la date une date GMT
toLocaleString()	convertit la date une date locale
UTC(a,m,j)	rend le nombre de ms entre la date et le 1/1/70

A.6 document

L'objet document contient les onformations sur le document courant. Il fournit aussi des méthodes pour afficher du texte au format HTML dans le navigateur.

Propriétés

alinkColor	couleur d'un lien actif
anchors	tableau regroupant les ancres du document
bgColor	couleur de fond
cookie	
fgColor	couleur d'avant plan
forms	tableau des formulaires
lastModified	date de denière modification
linkColor	couleur des liens
links	tableau des liens
location	URL du document courant
referrer	URL de la page précédemment visitée
title	titre de la page
vlinkColor	couleur d'un lien visité

Méthodes

close()	ferme un flux ouvert par la méthode open()
getSelection()	rend la chaine selectionnée dans le document courant
open()	ouvre un flux pour affichage de texte avec write()
write($e_1, .., e_n$)	affiche les expressions $e_1, .., e_n$ dans le document.
writeln($e_1, .., e_n$)	idem mais ajoute un retour à la ligne

A.7 form

L'objet form permet de créer un formulaire. Les formulaires peuvent contenir des champs, des zones de texte des boutons, des checkboxes, etc.

Syntaxe

```
<FORM NAME="name" TARGET="target" ACTION="file"
 METHOD="POST/GET" ENCTYPE="encodingtype">
```

où `TARGET` désigne la fenêtre dans laquelle les résultats doivent aller, `ACTION` défini quelle action doit être entreprise quand le formulaire à été rempli. Ce peut être entre autres un nom de fichier HTML à afficher, un script cgi à exécuter, ou une adresse electronique vers laquelle envoyer un courrier. `METHOD` peut prendre les valeurs POST ou GET, qui définissent comment la réponse est envoyée au serveur. `ENCTYPE` indique l'encodage MIME à utiliser dans le transfert.

Propriétés action, elements, encoding, length, method, target, button, checkbox, hidden, password, radio, reset, select, submit, text, et textarea

Méthodes submit()

A.8 frame

L'objet frame permet de manipuler les cadres. Un cadre est une division de la page web en plusieurs zone. La balise `<FRAMESET>` permet de diviser une zone en deux, chacune des zones pouvant être nouveau divisée en deux.

Syntaxe

```
<FRAMESET ROWS="x1,x2" COLS="y1,y2">
<FRAME SRC="file" NAME="name">
</FRAMESET>
```

`ROWS` spécifie les nombres de lignes dans le premier cadre et le deuxième cadre par `x1` et `x2` respectivement, qui sont soient des entiers, soit des pourcentages de la zone divisée. De la même manière, `COLS` spécifie la largeur des cadres. `SRC` indique quel fichier doit être chargé dans le cadre.

Propriétés frames
name
length
parent
self
window

Méthodes clearTimeout(), et setTimeout()

A.9 hidden

L'objet hidden est un champ texte invisible pour l'utilisateur, permettant d'ajouter des informations pré-définies dans un formulaire.

Syntaxe `<INPUT TYPE="hidden" NAME="name" VALUE="value">` où `VALUE` est la valeur du champ.

Propriétés name et value

Méthodes : aucune

A.10 history

L'objet history contient des informations sur la session de navigation en cours. Les informations sont essentiellement les URL des sites visités.

Propriétés length

Méthodes back(), forward(), et go()

A.11 Image

L'objet Image contient les informations concernant les images et cartes insérées dans le document HTML à l'aide des balises `<IMG>` et `<AREA>`. Les propriétés sont disponibles pour Netscape 3, Netscape 4 et Internet Explorer 4 à l'exception des propriétés x et y uniquement disponibles dans Netscape 4.

Propriétés

border	valeur de l'attribut de même nom dans la balise `<IMG>`
complete	booléen indiquant si l'image est complétement affichée
height	la hauteur de l'image en pixels
hspace	valeur de l'attribut de même nom dans la balise `<IMG>`
lowsrc	similaire à src, mais pour des images longues à charger
name	valeur de l'attribut de même nom dans la balise `<IMG>`
src	le nom du fichier image
vspace	valeur de l'attribut de même nom dans la balise `<IMG>`
width	valeur de l'attribut de même nom dans la balise `<IMG>`
x	l'abscisse du coin haut-gauche de l'image
y	l'ordonnée du coin haut-gauche de l'image

Méthodes aucune

A.12 link

Syntaxe `<HREF="location" NAME="name" TARGET="target">` avec `HREF` l'URL du fichier à charger quand on clique sur le lien, et `TARGET` est le nom de la fenêtre ou l'afficher.

Propriétés

hash	ancre dans l'URL
host	nom du serveur dans l'URL
hostname	adresse IP du serveur
href	URL complète
pathname	texte après le symbole / dans l'URL
port	port que le serveur utilise
protocol	début de l'URL (http,ftp,...)
search	texte après le symbole ? dans l'URL
target	où afficher la page de cette URL

Méthodes aucune

A.13 location

L'objet location contient les informations sur l'URL courante.

Propriétés

hash	ancre dans l'URL
host	nom du serveur dans l'URL
hostname	adresse IP du serveur
href	URL complète
pathname	texte après le symbole / dans l'URL
port	port que le serveur utilise
protocol	début de l'URL (http,ftp,...)
search	texte après le symbole ? dans l'URL
target	où afficher la page de cette URL

Méthodes aucune

A.14 Math

Propriétés

`E`	constante d'Euler (~ 2.718)
`LN2`	logarithme de 2 (~ 0.693)
`LN10`	logarithme de 10 (~ 2.302)
`LOG2E`	Returns the base 2 logarithm of e, about 1.442
`LOG10E`	Returns the base 10 logarithm of e, about 0.434
`PI`	valeur de Pi (~ 3.14159)
`SQRT1_2`	racine carrée de $\frac{1}{2}$ (~ 0.707)
`SQRT2`	racine carrée de 2 (~ 1.414)

Méthodes

abs(x)	rend la valeur absolue de x
acos(x)	rends arc cosinus de x
asin(x)	rends arc sinus de x
atan(x)	rend l'arc tangente de x
ceil(x)	rend l'entier immédiatement supérieur ou égal à x
cos(x)	rend le cosinus de x
exp(x)	rend e^x
floor(x)	rend l'entier immédiatement inférieur ou égal à x
log(x)	rend le logarithme de x
max(x,y)	rend le maximum de x et y
min(x,y)	rend le minium de x et y
pow(x,y)	rend x^y
random()	rend un nombre aléatoire entre 0 et 1
round(x)	rend l'entier le plus proche de x
sin(x)	rend le sinus de x
sqrt(x)	rend $\sqrt{x}$
tan(x)	rend la tangente de x

A.15 navigator

Cet objet permet d'obtenir des informations sur le navigateur visitant la page.

Propriétés

appCodeName	rend le nom de code du navigateur (Mozilla)
appName	rend le nom commercial du navigateur
appVersion	rend le numéro de version du navigateur
userAgent	rend le nom complet du navigateur

Méthodes aucune

A.16 password

L'objet password créé un champ texte comme l'objet text, mais les caractères saisis apparaissent comme des astérisques, rendant la saise confidentielle aux yeux d'autres personnes que l'utilisateur.

Syntaxe

```
<INPUT TYPE="PASSWORD" NAME="NAME" VALUE="VALUE" SIZE="SIZE">
```

`SIZE` est la largeurdu champ en nombre de caractères.

Propriétés

defaultValue	valeur initiale du champ
name	nom du champ
value	valeur courante du champ

Méthodes

blur()	Enlève le curseur du champ
focus()	Amène le curseur dans le champ
select()	Sélectionne le texte saisi

A.17 radio

Les objets radios sont des boutons ne permettant a l'utilisateur de ne choisir qu'une option parmi plusieurs.

Syntaxe `<INPUT TYPE="radio" NAME="Name" VALUE="Value" {CHECKED}>` où `VALUE` est le texte apparaissant sur le bouton. En spécifiant l'attribut `CHECKED` alors le bouton est pré-sélectionné au chargement du formulaire.

Propriétés

checked	indique si le bouton à été sélectionné
defaultChecked	indique quel bouton doit être pré-selectionné
length	donne le nombre de boutons
object	
name	permet de donner un nom au(x) bouton(s)
value	permet de donner une valeur au(x) bouton(s)

Méthodes click()

Evènements onBlur, onClick, onFocus

A.18 reset

Le bouton reset permet de réinitialiser un formulaire.

Syntaxe `<INPUT TYPE="RESET" NAME="NAME" VALUE="VALUE">` où `VALUE` est le texte apparaissant sur le bouton.

Propriétés

name	Le nom du bouton
value	La chaine associée au bouton

Méthodes click()

Evènements onBlur, onClick, onFocus

A.19 select

Syntaxe

```
<SELECT NAME="NAME" SIZE="SIZE" {MULTIPLE}>
<OPTION VALUE="option"> Texte </SELECT>
```

où `SIZE` fixe le nombre d'options avant de dérouler le menu (défaut 1), `MULTIPLE`, s'il est précisé fait du menu une liste à choix multiple. L'attribut `<OPTION VALUE="option">` ajoute une option de menu à chaque fois qu'il est ajouté. Il faut donc un attribut `OPTION` pour chaque option du menu.

Propriétés	length	Met à jour le nombre d'options
	name	Met à jour le nom du menu
	options	Tableau des différentes options du menu
	selectedIndex	Rend le numéro de l'option choisie

Méthodes : blur(), et focus()

Evènements onBlur, onChange, onFocus

A.20 string

Propriétés length

Méthodes	big	Affiche la chaine dans une grande police
	blink	Affiche la chaine en clignotant
	bold	Affiche la chaine en caractères gras
	charAt	Rend le caractère à une certaine position dans la chaine
	fixed	Affiche la chaine dans une police à taille fixe
	fontcolor	La couleur de fond d'affichage de la chaine
	fontsize	La taille de la police d'affichage de la chaine
	indexOf	Rend la première occurence d'une sous-chaine dans la chaine
	italics	Affiche la chaine en italique
	lastIndexOf	Cherche la dernière occurence d'une valeur dans une chaine
	link	affiche La chaine comme un lien
	small	Affiche la chaine en petite police
	strike	Affiche la chaine avec une barre en travers
	sub	Affiche la chaine en mode indice (subscript)
	substring	Référence une sous-chaine
	sup	Affiche la chaine en mode exposant (superscript)
	toLowerCase	Change la chaine en minuscules
	toUpperCase	Change la chaine en majuscules

A.21 submit

L'objet submit est le bouton permettant de valider un formulaire.

Syntaxe `<INPUT TYPE="SUBMIT" NAME="NAME" VALUE="Text">` où VALUE est le texte affiché sur le bouton.

Propriétés name, et value

Méthodes click()

Evènements onClick

A.22 text

Un objet text est un champ de saisie sur une ligne permettant de saisir toute chaine alphanumérique.

Syntaxe `<INPUT TYPE="text" NAME="name" VALUE="value" SIZE="size">` où `VALUE` est le texte affiché initialement dans le champ, et `SIZE` est la largeur du champ en nombre de caractères.

Propriétés defaultValue, name, form, type, value

Méthodes focus, blur, and select

Evènements onBlur, onChange, onFocus, onSelect

A.23 textarea

L'objet textarea permet à l'utilisateur de saisir plusieurs lignes de textes dans un formulaire.

Syntaxe

```
<TEXTAREA NAME="name" ROWS=rows COLS=cols
 WRAP="off/virtual/physical">Text</TEXTAREA>
```

où `rows` spécifie le nombre de lignes du champ, et `cols` le nombre de caractères par ligne.

Propriétés defaultValue, name, et select

Méthodes focus, blur, et select

Evènements onBlur, onChange, onFocus, onKeyDown, onKeyPress, onKeyUp, onSelect

A.24 window

Syntaxe `w = window.open("URL", "NAME" {,"Options"})`.

où `URL` est l'adresse de la page à ouvrir, `NAME` est le nom de la fenêtre, et Options sont des paramètres optionels comme la taille de la fenêtre, etc.

Propriétés	defaultStatus	Le message affiché par défaut dans la barre de statut
	document	Permet d'accéder au document courant
	frames	Le tableau contenant les cadres de la fenêtre
	frame	La fenêtre avec ascenseur utilisée pour un tableau
	length	Le nombre de cadre dans la fenêtre
	location	L'URL du document courant
	name	Le nom de la fenêtre
	parent	La fenêtre englobante
	self	Référence à la fenêtre elle-même
	status	Le message de la barre de statut
	top	Référence à la fenêtre mère
	window	Référence à la fenêtre courante

Méthodes	alert()	Créé une boite de dialogue de type message+OK
	close()	Ferme le document
	confirm()	Créé une boite de dialogue de type message+OK/Cancel
	open()	Ouvre une nouvelle fenêtre
	prompt()	Créé une boite de dialogue de type message+zone de saisie+OK/Cancel
	setTimeout()	Exécute le code JavaScript après un certain nombre de millisecondes
	clearTimeout()	Annule l'effet de la commande setTimeout()