JAVASCRIPT - COURS

Ivan KURZWEG

February 15, 2011

JAVASCRIPT - COURS
by Ivan KURZWEG

Contents

List of Figures

List of Tables

Chapter 1

Le langage Javascript

Ce langage, initialement créé par la société Netscape, permet de rendre une page HTML bien plus interactive, en y insérant du code réagissant, par exemple, aux évènements de l'utilisateur, ou encore à valider les données saisies dans un formulaire HTML. Dans ce premier chapitre, nous allons nous intéresser à la syntaxe du langage, mais aussi à son organisation, en relation avec ce que vous avez vu en cours d'algorithmique. Nous verrons ensuite les différents objets du navigateurs, et les manières de les programmer.

1.1 Ecriture de code Javascript

1.1.1 Format général d'un programme

La particularité du langage Javascript par rapport à d'autres langage de programmation (tels que Java ou C++), est sa faculté d'intégration dans une page HTML. Ainsi, le code écrit dans une page HTML est directement interprété par le navigateur, qui exécute les instructions Javascript.

Un "programme" Javascript est composé d'une série d'intructions que le navigateur va exécuter séquentiellement, c'est à dire l'une après l'autre. Chaque instruction est terminé par un point virgule.

Example 1.1 Hello World

Ce premier exemple nous montre deux instructions en Javascript, la première affichant *"Hello"*, et la deuxième affichant *"World"*.

```
printf("hello");

printf("World");
```

Attention, le langage est dit "*case sensitive*", c'est à dire que le langage est sensible aux majuscules et minuscules.

1.1.2 Intégration dans une page HTML

Comme nous l'avons déjà souligné, le langage Javascript a été crée pour être exécuté sur un navigateur Internet. Il doit donc s'intégrer dans le code HTML. Nous avons à notre disposition une balise HTML particulière :

```
<script language = "javascript">
.....
.....
</script>
```

Il existe trois manière d'intégrer le code Javascript dans le code HTML :

1.1.2.1 Intégration pour exécution directe

Par cette méthode, le code est directement exécuté au chargement de la page. La balise **<script></script>** est placé entre les balises **<body></body>**, et le navigateur exécute le programme, avant même l'affichage de la page HTML.

```
<html>
<head>
<title> .......... </title>
</head>
<body>

<script language="JavaScript">
place du code JavaScript
</script>

</body>
</html>
```

1.1.2.2 Exécution différée

Cette fois ci, le code est d'abord lu par le navigateur, et stocké en mémoire. Il ne s'exécutera que sur demande, c'est à dire lors d'un évènement généré par l'intervention de l'utilisateur.

Nous reviendrons plus tard sur ces évènements et la manière de les gérer.

```
html>
<head>
<title> .......... </title>

<script language="JavaScript">
place du code JavaScript
</script>

</head>
<body>
place du code événement
</body>
</html>
```

1.1.2.3 Insertion de code JavaScript à l'intérieur d'une balise HTML

Nous verrons dans la partie suivante de ce cours que certaines balises HTML acceptent de réagir à des événements, provoqués soit par l'intervention de l'utilisateur, soit par une action du navigateur lui même, comme le chargement de la page HTML (événement **onLoad)** ou la fermeture de la page HTML en cours (événement **onUnload).**

Dans ce cas, le code JavaScript peut être aussi inséré directement au niveau de la balise en question. Le squelette de la page HTML est alors :

```
<html>
<head>
<title> .......... </title>

<script language="JavaScript">
(cet emplacement peut être vide, le code est dans la balise. Mais il est  ↩
    indispensable, car il indique au navigateur qu'il va devoir interpréter du  ↩
    code JavaScript.)
</script>

</head>
<body>

<balise html événement=javascript:place du code événement
```

```
</body>

</html>
```

1.1.3 Les premiers programmes Javascript

Pour l'ensemble des exercices de ce cours, nous utiliserons comme étiteur de code HTML et Javascript XEmacs, et comme navigateur Internet, une version récente de Netscape ou Mozilla. Comme vu dans le cours HTML, nous écrirons directement nos pages HTML dans XEmacs, et nous les ouvrirons par la commande de Netscape "Fichiers -> ouvrir un fichier". Volontairements, nous épurerons au maximum le code HTML, de manière à nous concentrer sur le code Javascript.

1.1.3.1 Insertion de code Javascript à exécution directe

Sous XEmacs, écrire le code suivant, et enregistrer la page sous le nom "`Exercice 1.3.1`, dans le répertoire `Javascript/Exercices/`. Vous ouvrirez ensuite la page sous Netscape, en observant soigneusement l'interprétation du code et le comportement du navigateur.

```
<html>
<head>
<title>Exercice javascript 1.3.1</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<body bgcolor="#FFFFFF" text="#000000">
<script language = "javascript">
alert("Le message d'alerte s'affiche avant même le contenu de la page.\nCliquez ↩
    sur OK pour que la page s'affiche");
</script>
<h1>Exercice 1.3.1 : Code javascript a ex&eacute;cution directe</h1>
<p>Le script Javascript a &eacute;t&eacute; ex&eacute;cut&eacute; avant l' ↩
    affichage
  de la page.
</body>
</html>
```

Nous voyons ici une seconde instructions Javascript, l'instruction **alert**, qui permet d'ouvrir une boite de dialogue sur un navigateur. Nous aurons l'occasion de revenir sur cette instructions.

A retenir : quand le code Javascript est placé juste après la balise ***<body>****, il est exécuté avant l'affichage de la page.*

1.1.3.2 Insertion de code javascript à exécution différée

Selon le même principe que dans l'exercice précédent, écrire et affiché la page suivante, enregistrée sous le nom `Exercice 1.3.2`.

```
<html>
<head>
<title>Exercice javascript 1.3.2</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<body bgcolor="#FFFFFF" text="#000000">
<script language = "javascript">
function affiche()
{
   alert("Le message d'alerte s'affiche suite à un évènement.\nCliquez sur OK ↩
       pour poursuivre");
}
</script>
<h1>Exercice 1.3.2 : Code javascript a ex&eacute;cution diff&eacute;r&eacute;e</ ↩
    h1>
<p>Le script Javascript est ex&eacute;cut&eacute; suite &agrave; l'arriv&eacute;e
  d'un &eacute;v&egrave;nement dans la page, ici quand l'utilisateur clique sur
```

```
  le bouton ci-dessous.
<form>
<center>
<input type = "button" name="evenement" value = "Cliquez ici pour générer un  ↩
    évènement" onclick= "javascript:affiche()"
</center>
</form>
</body>
</html>
```

Le code contient cette fois ci une fonction Javascript. Nous reviendrons sur ce principe de programmation, mais nous pouvons déjà remarquer que les instructions contenues dans cette fonction (l'instruction **alert** en l'ocurence) sont exécuté lors de l'appel à `affiche()`. Cet appel de la fonction est réalisé quand l'évènement **onclick** apparait sur le bouton du formulaire.

A retenir : pour une exécution de code Javascript différée, il est nécessaire d'englober le bloc d'instruction dans une fonction, qui est appelée depuis le code HTML.

1.1.3.3 Insertion du code Javascript à l'intérieur d'une balise HTML

Toujours sur le même principe, écrire la page HTML `exercice1-3-3.html` contenant le code suivant :

```
<html>
<head>
<title>Exercice javascript 1.3.3</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<script language = "javascript">

</script>
</head>
<body bgcolor="#FFFFFF" text="#000000">

<h1>Exercice 1.3.3 : Code javascript ins&eacute;r&eacute; dans une balise HTML</ ↩
    h1>
<p>Le script Javascript est ex&eacute;cut&eacute; suite &agrave; l'arriv&eacute;e
  d'un &eacute;v&egrave;nement dans la page, ici quand l'utilisateur clique sur
  le bouton ci-dessous.
<form>
<center>
<input type = "button" name="evenement" value = "Cliquez ici pour générer un  ↩
    évènement" onclick= "
javascript:alert('Ce est affiché suite à un évènement.\nCliquez sur OK pour  ↩
    poursuivre');"
</center>
</form>
</body>
</html>
```

Cette fois-ci, l'instruction **alert()** est directement insérée dans le code HTML, au niveau de l'attribut **onclick** de la balise button. Il est bien sûr possible de cumuler plusieurs instructions Javascript à la suite, en les séparant par des points - virgules. Les balises (au contenu vide) **<script></script>** placées en entête de la page permettent de préciser au névigateur qu'il aura du code à interpréter.

*A retenir : dans le cas d'une insertion de code à l'intérieur d'une balise HTML, on place les instructions précédées du mot-clef **javascript:** directement dans la balise, séparées de points-virgules. Il est nécessaire de préciser au navigateur qu'il y aura du code à interpréter dans la page, en plaçant en entête de document les balises **<script language = javascript></script>**.*

1.1.4 Insertion de code par appel de module externe

Dans les paragraphes précédent, nous avons inséré le code Javascript directement dans la page HTML. C'est la méthode la plus simple et la plus fréquemment utilisée dans la création de sites Internets.

Cependant, cette méthode trouve ses limites dans la réutilisation et la maintenance des fonctions. En effet, si vous utilisez souvent une même série d'intructions Javascript dans plusieurs pages HTML, et

que vous souhaitez les modifier, vous devrez reprendre une par une l'ensemble des pages ... ce qui peut rapidement devenir fastidieux.

Une solution à ces problèmes est de placer des fonctions Javascript dans un module externe. Ce fichier contiendra le code source, et l'insertion dans la page HTML deviendra :

```
<script src = "URL du module externe">
..
..
</script>
```

Ces balises indiquent au navigateur non plus la nature du code qu'il doit utiliser, mais l'emplacement auquel il trouvera ce code. Ainsi, en modifiant le contenu du module externe, on modifiera le comportement de l'ensemble des pages qui s'y réfèrent.

1.1.4.1 Rédaction du module externe

Le module externe doit être écrit au format texte simple, et portant l'extension .txt. Il doit être placé à son adresse d'appel. Il contient le code source Javascript, généralement écrit sous forme de fonctions.

Travail à faire : créer un module externe nommé **`mod_jvs1.txt`**, et le placer dans le répertoire `/javascript/exercices/modules/`. Y insérer le code de deux fonctions, similaires à la fonction **affiche()** vue dans les exercices précédents :

- **affiche1()** : ouvre une boite de dialogue affichant "Execution de la fonction 1"
  affiche2() : ouvre une boite de dialogue affichant "Execution de la fonction 2"

1.1.4.2 Appel du module externe

Nous disposons donc désormais des fonctions **affiche1()** et **affiche2()**, dont les code sont décrits dans le module externes. Nous souhaitons utiliser ces fonctions dans une page HTML.

Travail à faire : selon le modèle vu dans le paragraphe 1.3.2, écrire la page HTML `exercice-1.4.2.html`, contenant deux boutons, exécutant chacun une des deux fonctions **affiche()** contenues dans le module externe `mod_jvs1.txt`.

Note : afin d'indiquer au navigateur où se trouve le module externe, il est nécessaire d'insérer en entête du document la balise suivante, au contenu vide.

```
<script src = "URL du module externe">
</script>
```

"URL du module externe" contiendra le chemin relatif du module externe par rapport à la page HTML, à savoir ici : `/modules/mod_jvs1.txt`. Par cette commande, le navigateur chargera en mémoire le contenu du module externe, et pourra exécuter les fonctions qui y sont contenues.

1.2 Structure du langage

Nous avons vu dans les paragraphes précédents quelques aperçus du langage Javascript, comme les fonctions, les instructions ou les blocs. Dans cette partie du cours, nous allons nous intéresser plus précisément à la syntaxe générale du langage, et à quelques unes de ses instructions et fonctions clefs.

1.2.1 Blocs et instructions

Une instructions Javascript peut être vue comme un ordre passé à l'interpréteur du navigateur. Chaque instruction se termine par un point-vrigule.

Un bloc d'instructions est une série d'instructions exécutées séquentiellement, et encadré par des accolades. Les blocs d'instructions permettent de définir par exemple les actions effectuées par une fonction, ou encore la série d'instructions répétées dans une structure itérative, comme vous l'avez vu en algorithmique.

1.2.2 Variables

Vous avez vu en algorithmique la notion de variables, éléments indispensables à la programmation, permettant de travailler sur des données informatiques.

1.2.2.1 Déclaration et affectation de variables

Toute variable utilisée dans un programme javascript doit obligatoirement être déclarée, par l'instructions suivante :

```
var mon_nom_de_variable;
```

Concernant les noms de variables, il n'y a pas de limitations particulière, si ce n'est de ne pas utiliser un des mots réservés du langage. Il est également à noter que le langage étant *case sensitive*, les majuscules et minuscules ont leur importance.

Une fois la variable déclarée, il est alors possible de lui affecter une valeur, grâce à l'opérateur d'affectation :

```
mon_nom_de_variable = ma_valeur
```

1.2.2.2 Type de variables

Comme nous l'avons souligné en introduction, le langage Javascript nous permet de déclarer les variables sans les typer, c'est-à-dire sans spécifier le type de données qu'elles contiendront. Le type de la variable dépendra dans ce cas-là de la nature de la valeur qui lui sera affectée en premier. Par exemple, le programme suivant forcera la variable a être de type entier :

```
var mon_age;
mon_age = 30;
```

Une autre solution pour typer les variables directement lors de leur déclaration est de leur affecter directement une valeur :

```
var mon_age = 30, mon_prenom = "ivan", feminin = false;
```

L'instruction précédente nous permet de passer en revue les trois types de variables principaux : numérique (**Number**), chaine de caractère (**String**) et booleen (**Boolean**) :

- **Number** : accepte tout nombre (y compris à virgule)
- **String** : accepte toute chaine de caractère, sans limitation de longeur
- **Booelan** : prend les valeur **True** (vrai) ou **false** (faux)

1.2.2.3 Portée d'une variable

Une fois la variable déclarée, elle est directement utilisable dans le bloc d'instructions qui la contient, et dans tous ceux qui lui sont inférieurs. Ainsi, une variable déclarée juste après la balise <script> sera utilisable dans tout le programme, y compris dans les fonctions qui le compose. Par contre, une variable déclarée dans une fonction ne sera pas accessible depuis une autre fonction. Nous reviendrons sur cet aspect important de la portée des variables dans le paragraphe consacré aux fonctions.

1.2.3 Opérateurs

1.2.3.1 Opérateurs arithmétiques

Les opérateurs arithmétiques s'appliquent sur une ou deux opérandes de types **Number**.

Les opérateur d'incrémentation et de décrémentation correspondant aux instructions suivantes :

```
i++ équivaut à i=i+1
i-- équivaut à i= i-1
```

1.2.3.2 Opérateurs de comparaison

Les opérateurs de comparaison s'appliquent à différents types de données, et renvoient un booléen en fonction du résultat de la comparaison.

Table 1.1 Opérateurs arithmétiques

Opération	Signe
Addition	+
Soustraction	-
Multiplication	*
Division	/
Modulo	%
Negation	-
Incrément	++
Décrément	--

Table 1.2 Opérateurs de comparaison

Egalité	==
Différence	!=
Inférieur, supérieur	< ; >
Inférieur ou égal, supérieur ou égal	<= ; >=

1.2.3.3 Opérateurs logique

Ces opérateurs s'appliquent à des booléens, et permettent d'évaluer des expressions logiques :

1.2.4 Structures conditionnelles

Vous avez vu en algorithmique les structures conditionnelles. La traduction de l'instruction algorithmique **si ..alors ... sinon ... finsi** se traduit en Javascript par :

```
if (condition)
{
    liste d'instructions;
}
  else
{
    liste d'instructions;
}
```

La "condition" doit être une expression booléenne évaluable. Si elle est VRAI, alors la première série d'instructions est exécutée, si elle est FAUSSE, alors c'est la seconde série qui est exécutée.

1.2.5 Structures répétitives

Les structures itératives ou répétitives permettent d'exécuter plusieurs fois un bloc d'instructions.

1.2.5.1 Instruction for

La boucle **For** est à rapprocher de l'instruction **pour** vu en algorithmique. Elle permet de répéter un nombre de fois défini le bloc d'instructions situé juste après :

```
for (initialisation;test;incrément)
{
   liste d'instructions;
}
```

Table 1.3 Opérateurs logiques

ET	&&
OU	||
NON	!

La boucle **For** utilise donc un compteur, initialisé à une certaine valeur, valeur modifiée à chaque exécution du bloc d'instructions, jusqu'au changement de valeur de la condition. Le compteur étant une variable, il ne faut bien sûr pas omettre de la déclarer.

Example 1.2 Hello World "puissance 10"

```
var i;
for (i=1;i<=10;i++)
{
  printf("Hello world "+i+" <br>");
}
```

Cet exemple va donc écrire 10 fois les mots "`hello world`", suivi des chiffres 1,2,3 .. jusqu'à 10. Au premier passage dans la boucle, **i** se voit affecté la valeur 1. A chaque passage dans la boucle, le programme écrit la phrase, et incrémente **i** de 1. Quand **i** prend la valeur 11, la conditionnelle est fausse, et le programme saute directement à l'instruction suivante.

1.2.5.2 Instruction While

La boucle Javascript **while** est à rapprocher de la boucle **tantque** vue en algorithmique. Elle permet d'exécuter un bloc d'instructions tant qu'une expression booléenne est vraie.

```
while (condition)
{
   liste d'instructions;
}
```

1.2.6 Fonctions

1.2.6.1 Déclaration d'une fonction

Nous avons étudié les fonctions en algorithmique, et aperçu quelques exemples dans les paragraphes précédents de leur application en Javascript.

Une fonction en Javascript doit être déclarée selon le modèle suivant :

```
function nom_fonction (para1, para1, ...)
{
   Instruction;
   Instruction;
   ...
   return resultat;
}
```

Function est un mot réservé du langage Javascript. **Para1, para2,** ... contiennent des paramètres qui vont pouvoir être utilisés dans la fonction. Enfin, le mot réservé **return** permet de renvoyer une valeur à l'instruction appelante. Quand ce mot n'est pas présent dans le corps de la fonction, la fonction se comporte alors comme une procédure.

1.2.6.2 Appel d'une fonction

Un appel à une fonction est donc un ordre donné au navigateur pour exécuter des lignes de codes placées hors du bloc de code appelant. L'appel se fait de deux manières :

- soit directement en donnant le nom de la fonction et les arguments qu'elle accepte quand la fonction ne retourne pas de valeurs
- soit en affectant le résultat de la fonction à une variable.

1.2.6.3 Exemple de fonction

Cet exemple est tiré du site www.le-webmestre.net

```
function titre(police,couleur,taille,texte){
   document.write("<font color=\"" + couleur + "\" size=\"" + taille + "\" face ↩
       =\"" + police + "\">" + texte + "</font>");
}
titre("Comic Sans MS","red",4,"Je m'éclate à faire des titres en JavaScript !");
document.write("<br>");
titre("Impact,Verdana,sans-ms","#FF8040",5,"Bien sûr, JavaScript ne sert pas qu'à ↩
     faire des titres !");
document.write("<br>");
titre("Arial","black",7,"Mais cela vous montre les fonctions et les arguments !") ↩
    ;
```

1.2.7 Tableaux

Les tableaux permettent de stocker plusieurs valeurs dans une seule variable. La déclaration d'une variable de type tableau se fait en utilisant l'instruction suivante :

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

ou encore en précisant la taille du tableau

```
var mon_tableau = new Array(10);
```

ou encore en spécifiant les valeurs à insérer dans le tableau :

```
var mon_tableau = new Array("valeur1","valeur2", "valeur3", ...)
```

L'accès à un élément du tableau se fait en spécifiant l'indice (le rang) de l'élément :

```
mon_tableau[0]
```

Il est bien sûr possible de donner plusieurs dimensions à un tableau, comme en algorithmique :

```
var mon_tableau[10][10]
```

permet de définir un tableau de deux dimensions.

Enfin, pour obtenir la longueur d'un tableau, nous pouvons utiliser la propriété **length** :

```
mon_tableau.length
```

Chapter 2

Le modèle objet du navigateur

2.1 Le modèle objet du navigateur

Nous avons vu dans les paragraphes précédents la syntaxe du langage Javascript, et quelques unes de ses principales fonctions et instructions. Nous allons maintenant aborder la méthode de programmation du navigateur lui-même. Nous allons en particulier nous intéresser à la notion d'objets et d'évènements.

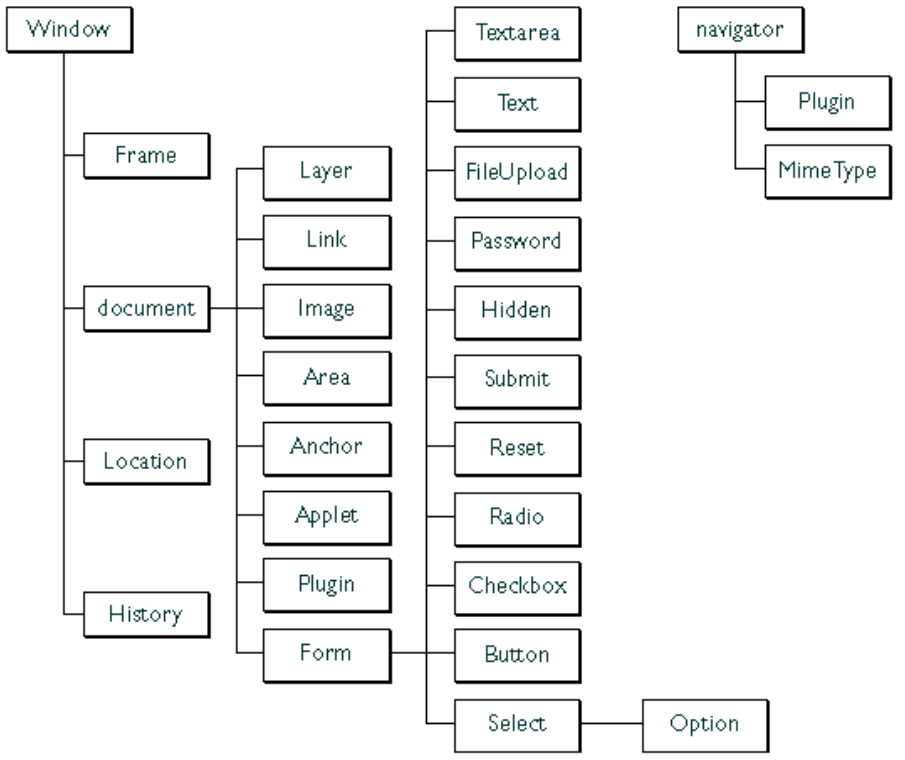
2.1.1 Les différents objets

Un objet est une entité appartenant au monde des navigateurs ou au langage lui-même. Il sont soit prédéfinis, soit créés de toute pièce par le programmeur. Un objet possède des propriétés, qui le caractérisent, et des méthodes, qui lui permettent d'effectuer des actions, et peuvent réagir à des évènements.

2.1.1.1 L'organisation des objets du navigateur

L'objet qui nous intéressera particulièrement dans la programmation Javascript est l'objet **Window,** qui modélise la fenêtre du navigateur. Les différents éléments qui peuvent composer une page HTML sont ainsi organisés de manière hiérarchique à partir de cet objet **Window.**

Figure 2.1 Hiérarchie des objets du navigateur



De la figure précédente, on peut par exemple déduire qu'un formulaire **(form)** peut contenir un objet **Text,** un objet **Button,** etc..

2.1.1.2 Les classes de bases de l'objet navigateur

Les objets sont automatiquement instanciés à chaque étape du fonctionnement du navigateur, par exemple lors de l'ouverture d'une fenêtre ou de frames, la création des documents contenus dans chaque fenêtre ou frame, et les divers éléments (formulaires, images, liens ...) contenus dans chaque élément. Les applications JavaScript peuvent alors dialoguer avec ces objets visuels et les manipuler. Le programmeur peut ainsi agir sur l'état du navigateur, de ses fenêtres et des documents et des composants qui y sont inclus.

Quelques uns des objets souvent manipulés :

- **navigator** : c'est le logiciel client dont le nom est noté dans **navigator.appName**
- **window** : l'objet de plus haut niveau créé par le navigateur, c'est sa fenêtre.
- **location** : c'est l'objet URL de la fenêtre courante.
- **history** : ce sont les URL précédemment visitées.
- **document** : il s'agit du document courant, dont les propriétés sontle titre, les couleurs (fond, texte, lien ...), les formulaires, les images etc..

Pour accéder à un objet de la page HTML, nous pouvons utiliser l'attribut name ou mieux encore, l'attribut id que nous aurons bien évidemment appliqués à l'ensemble de nos balises HTML.

Ainsi, l'accès à un champ du formulaire se fera par l'appel suivant :

```
window.document.nom_du_fomulaire.nom_du_champ
```

Nous voyons dans l'instruction précédente qu'il est cependant nécessaire de bien spécifier le "chemin" du formulaire : le formulaire faait bien partie de la fenêtre (**window**), qui contient un document HTML (**document**), qui lui même contient un formulaire, etc ...

2.1.1.3 Propriétés et méthodes

Chaque élément du document HTML est donc accessible via son nom et la hiérarchie des objets depuis le langage Javascript. Il s'agit maintenant de pouvoir manipuler ces objets en utilisant leurs propriétés, leurs méthodes et les évènements auxquels ils réagissent.

Nous avons vu qu'une méthode permet en *programmation objet* d'effectuer des actions sur l'objet, et qu'une propriété est une de ses caractéristiques.

Globalement, on peut manipuler grâce aux propriétés, l'ensemble des attributs de chaque balise HTML. Ainsi, si l'on veut modifier la classe CSS d'un objet de notre document, nous modifierons sa propriété **classname** :

```
window.document.mom_element.classname = "nouvelle_classe";
```

Tous les attributs de toutes les balises sont ainsi manipulables avec le Javascript !

De la même manière, il est possible de manipuler les évènements auxquels sont liés les éléments HTML, en utilisant leurs méthodes associées :

```
window.document.nom_element.methode();
```

Par exemple, l'envoi d'un formulaire, qui correspond à l'évènement HTML **onsubmit,** se fera par l'instruction Javascript :

```
window.document.nom_formulaire.submit();
```

2.1.2 Les événements

Nous avons vu dans les séquences du cours HTML que les balises HTML sont associées à des évènements auxquelles elles peuvent réagir. Ainsi, la balise **<body>** possède deux évènements intrinsèques souvent utilisés, **onload** et **onunload**.

A l'intérieur de ces balises, il est donc possible d'associer du code javascript qui sera executé lorsque l'évènement sera déclenché, soit par l'utilisateur (envoi d'un formulaire par exemple), soit par le navigateur lui-même (chargement d'une page par exemple).

Combiné à l'utilisation des feuilles de style, les évènements et Javascript permettent également d'obtenir des combinaisons graphiques intéressantes. Les évènements accessibles pour chaque balise HTML sont décrits dans la spécification HTML 4.0.

Voici une liste des principaux évènements intrinsèques :

Table 2.1 Listes des principaux évènements

Nom	Description
onAbort	Quand l'internaute arrête le chargement d'une image
onBlur	Quand un objet HTML n'a plus le focus
onChange	Quand un objet HTML a été modifié et qu'il n'a plus le focus
onClick	Quand l'internaute clique sur un objet HTML
onDbClick	Quand l'internaute double clique sur un objet HTML
onError	Quand le chargement de la page ou d'une image a entrainé une erreur
onFocus	Quand un objet HTML a le focus (cliquer dans un champ par exemple)
onKeyDown	Quand l'internaute appuie sur une touche de son clavier
onKeyPress	Quand une touche du clavier est enfoncée (combinaison de onKeyDown et onKeyUp)
onKeyUp	Quand l'internaute "lâche son doigt" d'une touche alors enfoncée
onLoad	Quand le changement de la page est fini
onMouseDown	Quand l'internaute appuie sur un bouton de la souris
onMouseMove	Quand l'internaute déplace sa souris
onMouseOut	Quand l'internaute enlève sa souris sans cliquer d'un objet HTML
onMouseOver	Quand l'internaute met sa souris sans cliquer sur un objet HTML
onMouseUp	Quand l'internaute "lâche son doigt" d'un bouton de sa souris alors enfoncé
onMove	Quand une fenêtre ou une frame est déplacée
onReset	Quand un formulaire est remis à zéro (avec un bouton [Reset])
onResize	Quand l'internaute redimensionne sa fenêtre de navigateur
onScroll	Quand la position de la barre de défilement est modifiée
onSelect	Quand l'internaute sélectionne du texte dans un objet HTML
onSubmit	Quand le formulaire est envoyé, quand l'internaute clique sur un bouton [Envoyer]
onUnLoad	Quand une page est quittée

2.1.3 Exemples

Example 2.1 Changement de classe d'un lien

```
<a href = "mon_url.html" class = "ma_classe1" on mouseover= "this.classname = ' ↩
    ma_classe2'"
onmouseout = "this.classname='ma_classe1'">mon lien change de forme avec le  ↩
    passage de la souris</a>
```

Example 2.2 Ouverture d'une nouvelle fenêtre

```
<a href = "javascript:window.open('mon_url.html', 'NouvFenetre', 'scrollbars=yes, ↩
    status=yes,width=300,height=300')"">
une nouvelle fenêtre va s'ouvrir : taille 300 x 300</a>
```

Example 2.3 Changement d'image au passage de la souris

```
<A HREF=""  onMouseOver="img.src='mon_img2.jpg "  onMouseOut="img.src='mon_img. ↩
    jpg' ">
  <IMG NAME="img" WIDTH=60 HEIGHT=61 BORDER=0
  SRC="mon_img.jpg"></A>
```

Chapter 3

Application au contrôle de saisie des données

3.1 Validation des données

Dans le cours HTML, nous avons étudié les formulaires, permettant à l'utilisateur de rentrer des données dans une page Web. Pour le bon fonctionnement d'une application, une vérification des données est souvent nécessaire. Par exemple, il vaut mieux vérifier qu'une date est bien valide avant de tenter de l'insérer dans une base de données. Cette validation des données peut se faire de deux manières :

- *du côté du serveur* : les données sont envoyées à travers le réseau, et un programme vérifie qu'elles sont valides. Mais ce mode de traitement nécessite de surcharger le serveur et encombre le réseau.
- *du côté du client* : avant de poster les données du formulaire, il est possible de programmer le navigateur pour qu'il effectue lui-même ces vérifications. C'est donc au niveau de la page contenant le formulaire qu'il faut écrire un programme Javascript réalisant ces opérations de contrôle .

3.1.1 Les objets du formulaires

Nous avons vu précédemment les objets Javascript, et plus particulièrement les objets du navigateur. Nous allons maintenant décrire avec précision certains des objets des formulaires, de manière à effectuer certaines opérations de contrôle et de formatage des données entrées par l'utilisateur.

3.1.1.1 Les zones de saisies de texte

Les zones de saisie de texte comporte les propriétés et méthodes suivantes :

- **value** : La valeur du champ (value en HTML)
- **name** : Le nom du champ (name en HTML)
- **size** : La taille du champ (size en HTML)
- **style** : Feuilles de Styles JavaScript (JSSS)
- **focus()** : Méthode qui donne le focus, c'est à dire que le curseur est dans le champ de saisie

3.1.1.2 Les boutons

Une seule méthode intéressante pour les boutons : **onclick()**

3.1.1.3 Les cases à cocher

Les cases à cocher ont :

- **onclick()** : évènement qui intervient quand l'utilisateur clique dessus
- **checked** : true pour valider la case à cocher, false sinon.

3.1.1.4 L'objet form : exemple

```
<html>
   <head>
      <title></title>
      <script language="JavaScript">
         function verif_form(){
            var temoin = 0;
            if(document.formulaire.nom.value == ""){
               alert("Le champ \"Nom\" est vide");                return false;
            }
            if(document.formulaire.prenom.value == ""){
               alert("Le champ \"Prénom\" est vide");
               return false;
            }
            if(document.formulaire.sexe.selectedIndex == 0){
               alert("Vous n'avez pas choisi votre sexe");
               return false;
            }
            for(i=0;i<document.formulaire.loisir.length;i++){
               if(document.formulaire.loisir[i].checked){
                  ++temoin;
               }
            }
            if(temoin != 3){
               alert("Vous devez choisir 3 loisirs !");
               return false;
            }
            return true;
         }
      </script>
   </head>
   <body>
      <form action="monmail@greta.fr"
          method="post" name="formulaire" onSubmit="return verif_form()">
      Votre nom <input type="text" name="nom"><br>
      Votre prénom <input type="text" name="prenom"><br>
      Sexe :
      <select name="sexe">
         <option>Choisissez</option>
         <option value="h">Homme</option>
         <option value="f">Femme</option>
      </select><br>
      Vos loisirs : (3 choix)<br><hr>
      <input type="checkbox" name="loisir" value="javascript">Le JavaScript<br>
      <input type="checkbox" name="loisir" value="couture">La couture<br>
      <input type="checkbox" name="loisir" value="cinéma">Le cinéma<br>
      <input type="checkbox" name="loisir" value="télé">La télévision<br>
      <input type="checkbox" name="loisir" value="marche">La marche<br>
      <input type="checkbox" name="loisir" value="sport">Le sport<br>
      <input type="checkbox" name="loisir" value="lecture">La lecture<br>
      <input type="checkbox" name="loisir" value="cheval">Le cheval<br>
      <input type="checkbox" name="loisir" value="dormir">Dormir<br>
      <hr>
      <input type="submit">
      </form>
   </body>
</html>
```

3.1.2 Traitement des chaines de caractères et des nombres

3.1.2.1 Traitements des chaînes de caractères

Lors de la validation des formulaires, il est souvent nécessaire de traiter les chaînes de caractères. Javascript met à notre disposition quelques méthodes applicables aux string, détaillées dans le manuel Javascript. En voici quelques unes :

- **indexOf(searchValue)** : retourne la place de la première occurence de la chaîne searchValue
- **substring(indexA, indexB)** : retourne la chaîne de caractère comprise entre indexA et indexB
- **toLowerCase(), toUpperCase()** : renvoie la chaîne passée respectivement en minuscule et en majuscule

3.1.2.2 Traitements des nombres

Outre les opérations classiques sur les nombres que nous avons vu en algorithmique, il peut également être intéressant de tester si une variable est bien de type Number, Javascript étant relativement souple sur le typage des variables. Ainsi, la fonction isNan(testValue) nous renvoie la valeur false si testValue n'est pas un nombre, et la valeur true dans le cas contraire.

3.2 Trouver des scripts sur Internet

Vous l'avez vu, Javascript est un langage très complet, permettant de donner du dynamisme aux pages en appliquant la programmation du côté client. Nous avons aussi la possibilité de décharger le serveur d'une partie de son travail, en utilisant le contrôle de saisie des champs des formulaires.

Javascript demande quand même une certaine expérience en programmation, avant de pouvoir réaliser des menus déroulants, des applications graphiques, etc.. mais représente une alternative sérieuse aux solutions propriétaires telles que Flash ou autres Plug-Ins, dont la gratuité n'est pas garantie dans l'avenir.

Un bon moyen de se familiariser avec Javascript est tout simplement d'aller chercher des exemples de réalisation de scripts sur Internet. Un site en particulier recense de nombreux exemples, organisés par catégorie :

http://javascript.internet.com/

Il existe également de nombreuses ressources documentaires sur Javascript, telles que des cours et des tutoriaux. Un bon portail vers des sites sur Javascript, et intégrant lui-même un cours très complet sur ce langage est :

http://www.le-webmestre.net