JavaScript

1. Présentation

Si le langage Java impose le téléchargement de classes, c'est à dire de programmes précompilés depuis le serveur, JavaScript provoque l'exécution de programmes non compilés mais interprétés et contenus dans le corps de la page HTML.
Alors que Java fut proposé par SUN et ensuite adopté par Netscape 2.0 , Javascript n'est compris que par Netscape et par Internet Explorer V3.0.
A la fin de l'année 1995, Netscape représentait 80% du marché des navigateurs, à la fin de l'année 1996 cette part sera vraisemblablement de 50%, mais sur PC on peut penser que 90% des navigateurs comprendront JavaScript. Certains suggèrent d'utiliser JavaScript pour ce qui est simple, Java pour ce qui est simple ou compliqué.
Pour passer des considérations marketing aux considérations techniques, revenons sur le fait que JavaScript est interprété là où Java est compilé. Ceci a plusieurs implications :

- le code Java est protégé des actes de copie frauduleuse,
- le code JavaScipt est moins *typé* donc moins robuste mais plus accessible à des non informaticiens.
- JavaScript est accessible à des auteurs de pages HTML alors que Java est réservé à un public plus professionnel.

Les principales sources d'information sur JavaScript sont aujourd'hui :

- la FAQ [www.freqgrafx.com/411/jsfaq.html] JavaScript (en anglais)
- la newsgroup comp.lang.javascript
- la page Yahoo relative à JavaScript

Ajoutons [www.essex1.com/people/timothy/js-index.htm] qui contient un grand nombre d'exemples de programmes en JavaScript.

2. Les mots du langage

Les lignes commençant par // sont des commentaires destinées à éclairer le code.
Plusieurs lignes peuvent être mises en commentaire si elles sont encadrées par les symboles /* ..*/.
Ajoutons enfin que le langage JavaScript fait la différence entre les majuscules et les minuscules.
Java comprend les types de données suivants :

- les **nombres** : 2, 2.90, 314E-2 . Ils peuvent être en base 10, 16 (0x.. ou 0X..) ou 8 (commençant par 0). Les réels flottants peuvent contenir un point décimal, un exposant (E)
- les **booléen** : *true* ou *false*
- les **chaînes de caractères** : "coucou" ou 'coucou'. Les caractères spéciaux peuvent être utilisés dans les chaînes (\b backspace, \f form feed , \n new line , \r CR, \t tabulation). Le caractère \ permet d'insérer une double quote dans une chaîne.
- les **tableaux** : tableau[0], tableau[1]
- les **tableaux associatifs** : X["Poitier"] , Y["Poitier"] qui ont donné tant de plaisir aux Perliens et TCLiens!

La typage des données étant assez faible, une variable déclarée dans un type peut être utilisée dans un autre type sans que de message d'erreur n'apparaisse. Netscape risque de ne pas supporter et de provoquer une erreur, mais personne ne vous dira exactement pourquoi.
JavaScript convertit automatiquement les entiers en chaînes de caractères, ce qui vous permet de concaténer des entiers avec des chaînes de caractères pour donner finalement une chaîne.
Par exemple

```
y = 50 + "FHT"
```

assigne à **y** la chaîne "50 FHT".
La déclaration

```
var nom = valeur
```

permet de déclarer une variable *nom* initialisée avec la valeur correspondante et du type de la valeur.
Les tableaux, jusqu'à la version 1.1, doivent être construits par invocation du mot clé new en donnant la taille du tableau; à partir de la version 1.2 ils peuvent être défini de façon littérale sous la forme :
Tableau = [element1, element2, element3]

2.1 Types prédéfinis

String
Le fait d'assigner une chaîne de caractères à une variable déclare la variable de type String.

```
var chaine = "Bonjour vous!"
```

créé un objet appelé *chaine*. Cet objet peut être :

- mis en majuscule par la procedure toUpperCase (par exemple : chaine.toUpperCase ())
- être un texte HTML avec tous les atributs possibles

Math
Les objets mathématiques permettent de gérer les constantes et les fonctions.
Par exemple :

- **Math.PI** désigne le nombre PI.
- **Math.sin(1.50)** est le sinus de l'angle 1.50 exprimé en radian.

Date
L'objet *Date* permet de manipuler date et heure dans votre application.
La date est le temps écoulé en milliseconde depuis le 1 er janvier 1970 et la création d'une nouvelle date se fait par une déclaration du type :

```
Nom = new Date(parametres)
```

Les valeurs que peut prendre le paramètre sont :

- **rien** : donne la date et l'heure courante
- **une chaîne de caractères** sous la forme *month day, year hour:min:sec* (December 25, 1995 13:30:00)
- **des entiers** : 95, 11, 25 , 13, 30 , 00

Les valeurs omises sont initialisées à la valeur zéro. Il est aussi possible de transformer des valeurs en date :

- **setxxx** : pour transformer des entiers en date

- **getxxx** : pour transformer en date et heure des objets *date*
- **toxxx** : pour retourner une chaîne de caractères correspondant à l'objet *date*

Par exemple **MaDate = new Date** ("Novembre 24, 1961") permettra d'utiliser MaDate.**getDay ()** pour retrouver la valeur 24.

Pour leurs valeurs numériques, les secondes et les minutes vont de 0 à 59, les heures de 0 à 23, les jours de la semaine de 0 à 6; les jours du mois de 1 à 31, les mois de 0 à 11 et les années sont décomptées depuis 1900.

2.2 Les variables

Les variables commencent par des lettres ou par le caractère souligné, dans le reste de l'identifiant des valeurs décimales sont autorisées.

Les noms en minuscule ou en majuscule sont différents.

2.3 Les expressions

Les expressions du langage JavaScript sont de trois ordres :

- arithmétiques : opérations sur les entiers, les réels.
- sur les chaînes de caractères
- booléennes ou logiques

Les expressions conditionnelles

(condition) ? val1 : val2 : évalue la condition et exécute *val1* si vrai, ou *val2* si faux.

Exemple : message **= (** fin = true **) ?** "bonjour" **:** "au revoir"

3. Opérateurs

Généralités

Comme en C ou en Java les opérateurs suivants sont valides : + (adition), - (soustraction) , * (multiplication) , ++ (incrément) , -- (décrément) , / (division), % (modulo), & (ET bit à bit), | (OU bit à bit), ^ (XOR bit à bit), << (décalage de bits vers la gauche), >> (décalage de bits vers la droite),>>> (idem mais en mettant les nouveaux digits à 0), && (ET logique entre deux booléens), || (OU logique entre deux booléens), ! (négation d'un booléen).

Les comparaisons sont faites par les opérateurs :

== (test dégalité) , > , >=, <, <=, != (test d'inégalité)

par exemple if (valeur >= 3){ } signifiera que l'on fera l'action entre {} si la variable valeur est plus grande ou égale à 3

Les opérateurs peuvent être assignés à l'affectation :

x += 4 équivaut à x = x + 4

x++ signifie x+=1 c'est à dire x = x +1 et x-- signifie x-=1 c'est à dire x=x-1

Les opérateurs sur les chaînes de caractères :

+ concatène deux chaînes += ajoute la chaîne de droite à la chaîne de gauche. La précédence des opérateurs est celle du langage C à savoir :

- ,
- = += -= *= /= %= <:<= >>= >>>= &= ^= |=
- ?:
- ||
- &&
- |
- ^
- &
- == !=
- < <= > >=
- << >> >>>
- + -
- * / %
- ! ~ - ++ --
- call, member () [] .

A partir de la version 3 de Nestcape, JavaScript permet d'évaluer des expressions dans les instructions de contrôle.

Ceci se fait en mettant le caractère **&** devant l'expression et en la terminant par un **;**. L'expression ainsi évaluée peut être mis entre accolades pour s'assurer de la portée des opérateurs.

Par exemple, "&{100*10};" vaudra 1000. Ces expressions évaluées peuvent être utilisées à droite d'un test de condition avec les opérateurs ==, < ou >.

4. Instructions de contrôle

Les instructions for

L'instruction **for** permet de répeter une séquence d'instructions tant qu'une condition est vraie; elle peut prendre deux formes :

- La condition de boucle, prend une valeur initiale, une valeur de fin et un pas d'incrément.

La syntaxe est la suivante :

```
for     ([expression     initiale];
[condition]; [expression finale]) {
  instructions
}
```

Exemple :

```
for ( i = 0; i < 255; i++) {
  fonction(n)
}
```

La boucle se répète pour un ensemble de valeurs de i =0 à 254 par incémente de i

```
for (var in obj )in   {
instructions
}
```

Les intruction break et continue

L'instruction break permet de sortir de la boucle ou du bloc en cours. L'instruction *continue* termine le bloc en cours pour se positionner sur l'itération suivante.

if ... else .. [else]

L'intruction de condition permet d'effectuer une séquence d'instruction si la condition exprimée est vraie.

La partie *else* qui est optionnelle permer d'effectuer la séquence suivante si la condition est fausse.

Syntaxe

```
if  (condition) {
  instructions
} [else {
  instructions
```

```
}]
```

Exemple

```
if ( prix == 0  ) {
  x++
} else {
  result = result + prix
}
```

while

L'instruction *while* permet de répéter un bloc d'instructions tant qu'une expression est vraie.

Syntaxe :

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

5. Les fonctions

Les **fonctions** en JavaScript comme dans tous les autres langages désignent des suites d'instruction répétitives, qui sont éxécutés plusieurs fois mais écrites une seule fois.

La fonction est définie sous la forme : ***nom*** *(param1, param2,...;) { ...}*

Il est de bon ton de définir toutes les fonctions à l'intérieur du couple de balises **<HEAD> </HEAD>**

La fonction sera donc déclarée entre un couple de balise **<SCRIPT> </SCRIPT>** par un appel du genre *nom (param1, param2)*.

Une fonction peut appeler des arguments de tout type, ainsi que des fonctions et pour les fous de l'algorythmique les fonctions peuvent être appelées récursivement.

Une fonction retourne une valeur quand elle rencontre le mot clé *return* suivie de la valeur retournée qui peut être de tout type. Les fonctions suivantes permettent de manipuler les chaînes de caractères :

- **eval** convertit une chaîne de caractères en valeur entière
- **parseInt** convertit une chaîne de caractères en valeur entière dans la base spécifiée
- **parseFloat** convertit une chaîne de caractères en valeur réelle flottante

A partir de la version 1.2 du langage, une fonction peut être appelée récursivement.

JavaScript est orienté objet

Comme son grand frère Java, JavaScript est orienté objet, ce qui revient à dire en première approximation que tout objet possède des propriétés, qu'à tout objet se voient associées des méthodes, sorte de fonctions propres à ces objets.

Certains objets sont pré-définis mais vous pouvez créer un arsenal de nouveaux objets dérivés avec leur propres méthodes.

Si un objet se nomme *Objet*, une propriété de cet objet sera notée *Objet.propriete*

Une propriété a un type et peut donc être manipulée selon ce type comme le serait un entier ou une chaîne de caractères.

Ainsi femme.jambe designera la (sublime) propriété *jambe* d'un objet *femme* (qui n'a rien à voir avec une femme objet).

Les méthodes sont des fonctions qui ont une portée locale sur un objet et non pas génarale sur toute une page HTML.

Une méthode se définit comme une fonction : *object.nom = nomfontion*

Le mot *this* comme dans tous les langages orientés objets permet de référencer l'objet courant.

Pour créer un nouvel objet, il suffit de :

- définir le type de l'objet en écrivant une fonction
- créer une instance de l'objet par l'utilisation du mot *new*

Par exemple :

```
function homme(jambe, bras, tete ) {
  this.jambe
  this.bras
  this.tete
}
```

La création l'objet *homme* se fait par

```
adam = new homme (1.20,0.70,0.50)
```

Bien évidemment, vous pouvez changer la taille du bras de l'objet *homme* par la syntaxe

```
adam.bras = .90
```

Il est possible (mais pas forcément heureux) d'ajouter une nouvelle propriété à un objet *homme* par allocation dynamique d'une valeur à une propriété non encore existante :

```
adam.torse=0.70
```

Enfin une méthode peut être définie par analogie à une fonction déjà existante comme suit :

```
this.creerHomme = creerFemme ;
```

L'appel à la fonction se fait par

```
homme.creerHomme()
```

6. JavaScript et HTML

JavaScript peut être implanté dans une page HTML de deux façons :

- par la balise SCRIPT
- en utilisant les événements
- en mettant le code dans un fichier séparé uniquement à partir de la version 3 de Nestcape.

En utilisant la balise SCRIPT

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

On le voit, le choix du langage ne se limite pas à JavaScript et reste ouvert à d'autres langages comme Visual Basic, que supporte Internet Explorer.

Le code placé à l'intérieur d'un couple de balises *SCRIPT* est évalué après le chargement de la page.

Les fonctions sont définies dans les couples de balises *SCRIPT* dans la partie entête (HEAD) de la page HTML ou même dans le corps de la page, ce qui n'est cependant pas conseillé.

La balise **SCRIPT** peut prendre un argument : **LANGAGE="JavaScript1.1"** qui indique que le code JAVASCRIPT est de version 1.1, actuellement compris par les versions de Netscape 3.x et supérieure et par les versions Microsoft Internet Explorer 4 et supérieures.

L'argument **LANGAGE="JavaScript1.2"** indique que le code JAVASCRIPT est de version 1.2,

actuellement compris par les versions de Netscape 4.x et supérieure.
Pour utiliser dans une page un code JavaScript à la fois compatible 1.0 et 1.1 il suffit de déclarer les procédures deux fois comme le montre l'exemple suivant :

```
<SCRIPT LANGUAGE="JavaScript"> code
</SCRIPT>
<SCRIPT LANGUAGE="JavaScript1.1">
</SCRIPT>
```

Ceci peut être obtenu également par l'astuce suivante

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

Ceci peut se décliner par des variantes du style

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

La variable jsVersion servira alors tout au long des programmes à établir un code compatible avec la version du navigateur.
Notons que le couple de balises **<NOSCRIPT> </NOSCRIPT>** permettent d'encadrer le texte qui sera affiché si JavaScript n'est pas compris par le navigateur.

En utilisant les événements

Les événements sont les résultats d'une action de l'utilisateur, comme par exemple un clic sur l'un des boutons de la souris.
La syntaxe de ces événements est :
<*balise* **eventHandler** ="code JavaScript"> où *balise* est le nom d'une balise et eventHandler est le nom d'un événement.
Par exemple, pour utiliser la fonction *exemple* quand un utilisateur appuie sur un bouton, l'instruction suivante

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

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

- Avec formulaire

```
<FORM>
<INPUT VALUE=événement
événement=alert("événement")>
texte explicatif sur l'événement
[mode d'emploi]
</FORM>
```

- Avec hypertexte

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

- **onClick** l'utilisateur clique sur une zone hypertexte.
  [*cliquez sur onClick*]
- **onLoad** : l'utilisateur charge la page dans le navigateur
- **onSelect** : l'utilisateur sélectionne un élément d'un formulaire
- **onSubmit** : l'utilisateur soumet un formulaire
- **onUnload** : l'utilisateur quitte la page

Le code dans un fichier source

Uniquement à partir de la version 3 de Netscape, il est possible de mettre le code des programmes JavaScript dans un fichier annexe, en utilisant la balise SCRIPT comme suit :

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

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

Dans le HTML

Il est enfin possible d'utiliser le JavaScript pour mettre des variables dans le code HTML, ceci permet de rendre des parties du langage dynamiques.
Une variable JavaScript est définie par la syntaxe suivante : **&variables;** de la même manière que les variables **>** désignaient le symbole >.
Des fonctions peuvent être utilisées dans le corps du champ variable, si elles sont délimitées par les symboles **{** et **}.**

7. Deux ou trois choses avant de commencer

Avant de plonger dans les fonctions et les objets de référence, regardons quelques techniques.
Avec le choix JavaScript vous avez et nous le verrons plus loin, la possibilité de fabriquer du texte HTML, qui ne sera pas forcément bien formaté par le navigateur, il vous faudra pour cela forcer un ré-affichage de la page. Il vous faut savoir également que JavaScript ne produit pas de de documents imprimables jusqu'à la version 3 de Nestcape; gardez cela présent à l'esprit car les pages HTML finissent souvent - et vous savez combien c'est vrai - sur le coin de votre imprimante.
Utilisez les simples quotes dans les portions de code JavaScript, pour réserver les doubles quotes pour les parties purement HTML.
Définissez des fonctions dans la partie entête de votre document.

Les objets de base

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

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

Ainsi pour la page HTML en cours, nous aurons les propriétés suivantes :

- **location.href** = "http://www.imaginet.fr/ime/javascri.htm"
- **document.title** = "Un nouveau guide d'Internet - JavaScript"
- **document.fgColor** = #000000

Ceci représente un avantage extrêmement intéressant : vous allez pouvoir définir en JavaScript une nouvelle valeur pour le fond de votre document ou définir vos formulaires de façon dynamique.

Le schéma suivant, que l'on retrouve souvent dans la littérature, illustre la hiérarchie de toutes les classes présentes dans le Navigateur :

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

Le formulaire dans un document est stocké dans un tableau appelé *forms*, le premier formulaire est noté *forms[0]*, la deuxième est *forms[1]* etc..

Ainsi la valeur du texte entré qui était *document.nom.ungi.value* peut être aussi appelé *document.form[0].ungi.value*.

Ce mécanisme se répète pour tous les éléments comme les radios boutons, les champs texte etc ..

L'objet *window* est un objet important dans le langage JavaScript, il permet l'ouverture et la fermeture de pages dans un nouveau navigateur, d'émettre des fenêtres *Popup* ou des fenêtres de confirmation.

A l'instar des formulaires rangés en tableau form[], les frames sont rangées dans un tableau frames[0], ... frames[n]. Ainsi la première frame d'une page composée est window.frame[0].

Les tableaux prédéfinis

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

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