MCOURS.COM
Le N°1 du cours et exercices sur Internet





Auteur :VanLancker Luc Rue des Brasseurs, 22 7700 Mouscron
BelgiumVanlancker.Luc@ccim.be

# *BREF PRÉAMBULE*

Ce support de formation, présenté comme un tutorial, est destiné à lâapprentissage du langage Javascript qui ajoute un peu de piment dans les pages Html.

L'apprentissage d'un langage de programmation, fut-il aussi simpliste que Javascript (c'est pourtant bien ce que prétendent certains!!!), implique la connaissance d'une nébuleuse d'éléments avant de pouvoir mettre en î uvre ceux-ci. Pour des raisons pédagogiques, nous avons conçu ce tutorial pour une lecture à deux niveaux :

- *un niveau débutant* qui rassemble les notions de base de Javascript.
- *un niveau avancé* (noté + ) pour aller un peu plus loin dans ces concepts (sans prétendre cependant à l'expertise).

# CHAPITRE 1 : JAVASCRIPT

**Javascript est un langage de scripts**

**incorporé aux balises Html**

**qui permet d'améliorer la présentation**

**et l'interactivité des pages Web.**

Javascript est donc une extension du code Html des pages Web. Les scripts, *qui s'ajoutent ici aux balises Html*, peuvent en quelque sorte être comparés aux macros d'un traitement de texte.

Ces scripts vont être gérés et exécutés par le browser lui-même sans devoir faire appel aux ressources du serveur. Ces instructions seront donc traitées en direct et surtout sans retard par le navigateur.

Javascript a été initialement développé par Netscape et s'appelait alors LiveScript. Adopté à la fin de l'année 1995, par la firme Sun (qui a aussi développé Java), il prit alors son nom de Javascript.

Javascript n'est donc pas propre aux navigateurs de Netscape (bien que cette firme en soit un fervent défenseur). Microsoft l'a d'ailleurs aussi adopté à partir de son Internet Explorer 3. On le retrouve, de façon améliorée, dans Explorer 4.

Les versions de Javascript se sont succédées avec les différentes versions de Netscape : Javascript pour Netscape 2, Javascript 1.1 pour Netscape 3 et Javascript 1.2 pour Netscape 4. Ce qui n'est pas sans poser certains problèmes de compatibilité, selon le browser utilisé, des pages comportant du code Javascript. Mais consolons nous en constatant qu'avec MSIE 3.0 ou 4.0 et la famille Netscape, une très large majorité d'internautes pourra lire les pages comprenant du Javascript.

L'avenir de Javascript est entre les mains des deux grands navigateurs du Web et en partie lié à la guerre que se livrent Microsoft et Netscape. On s'accorde à prédire un avenir prometteur à ce langage surtout de par son indépendance vis à vis des ressources du serveur.

# CHAPITRE 2 : JAVASCRIPT N'EST PAS JAVA

Il importe de savoir que Javascript est totalement différent de Java. Bien que les deux soient utilisés pour créer des pages Web évoluées, bien que les deux reprennent le terme Java (café en américain), nous avons là deux outils informatiques bien différents.

| Javascript | Java |
|---|---|
| Code intégré dans la page Html | Module (applet) distinct de la page Html |
| Code interprété par le browser au moment de l'exécution | Code source compilé avant son exécution |
| Codes de programmation simples mais pour des applications limitées | Langage de programmation beaucoup plus complexe mais plus performant |
| Permet d'accéder aux objets du navigateur | N'accède pas aux objets du navigateur |
| Confidentialité des codes nulle (code source visible) | Sécurité (code source compilé) |

Plus simplement :

- Javascript est plus simple à mettre en oeuvre car c'est du code que vous ajouterez à votre page écrite en Html avec par exemple un simple éditeur de texte comme Notepad. Java pour sa part, nécessite une compilation préalable de votre code.
- Le champ d'application de Javascript est somme toute assez limité alors qu'en Java vous pourrez en principe tout faire.
- Comme votre code Javascript est inclus dans votre page Html, celui-ci est visible et peut être copié par tout le monde (view source). Ce qui pour les entreprises (et les paranoïaques) est assez pénalisant. Par contre, en Java, votre code source est broyé par le compilateur et est ainsi indéchiffrable.
- Même si c'est une appréciation personnelle, les codes Javascript ne ralentissent pas le chargement de la page alors que l'appel à une applet Java peut demander quelques minutes de patience supplémentaire à votre lecteur.

# CHAPITRE 3 : UN PEU DE THEORIE OBJET

## 3.1 LES OBJETS ET LEUR HIERARCHIE

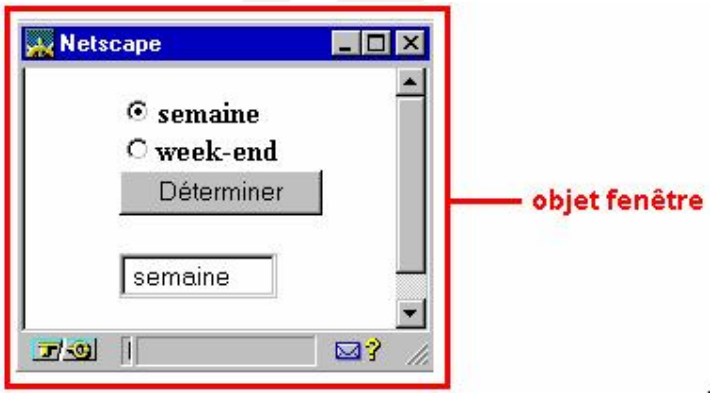
En bon internaute, vous voyez sur votre écran une page Web.

Javascript va diviser cette page en objets et surtout va vous permettre d'accéder à ces objets, d'en retirer des informations et de les manipuler.

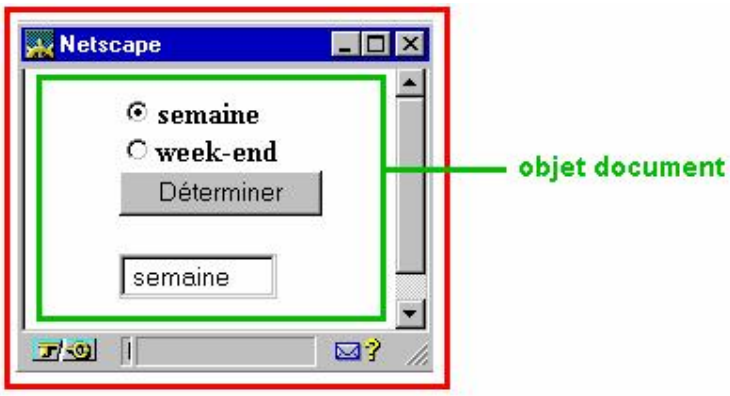
Voyons d'abord une illustration des différents objets qu'une page peut contenir.

Vous avez chargé la page suivante :

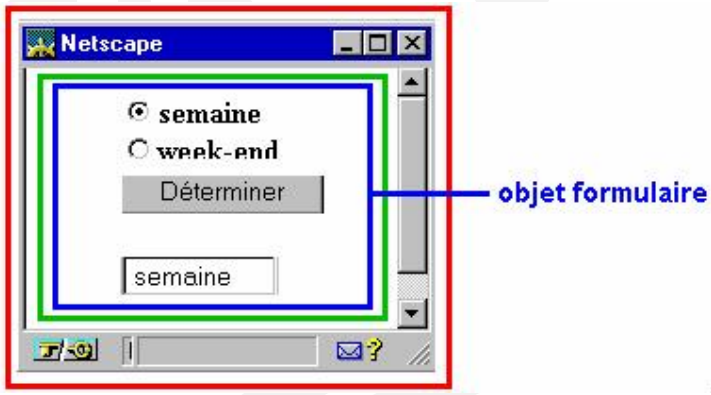

Cette page s'affiche dans une fenêtre. C'est l'objet fenêtre.



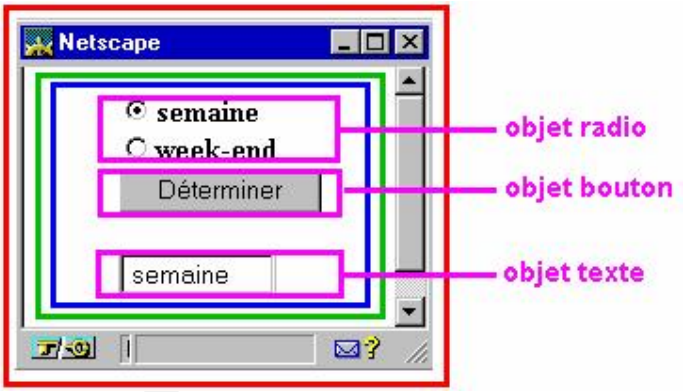

Dans cette fenêtre, il y a un document Html. C'est l'objet **document**. Autrement dit (et c'est là que l'on voit apparaître la notion de la hiérarchie des objets Javascript), l'objet fenêtre contient l'objet document.




Dans ce document, on trouve un formulaire au sens Html. C'est l'objet formulaire. Autrement dit, l'objet fenêtre contient un objet document qui lui contient un objet formulaire.




Dans ce document, on trouve trois objets. Des boutons radio, un bouton classique et une zone de texte. Ce sont respectivement l'objet radio, l'objet bouton, l'objet texte. Autrement dit l'objet fenêtre contient l'objet document qui contient l'objet formulaire qui contient à son tour l'objet radio, l'objet fenêtre contient l'objet document qui contient l'objet formulaire qui contient à son tour l'objet bouton et l'objet fenêtre contient l'objet document qui contient l'objet formulaire qui contient à son tour l'objet texte.




La hiérarchie des objets de cet exemple est donc

radio

fenêtre document formulaire bouton

texte

Pour accéder à un objet (vous l'avez peut-être déjà deviné), il faudra donner le chemin complet de l'objet en allant du contenant le plus extérieur à l'objet à l'objet référencé.

Soit par exemple pour le bouton radio "semaine" : (window).document.form.radio[0].

Nous avons mis l'objet window entre parenthèses car comme il occupe la première place dans la hiérarchie, il est repris par défaut par Javascript et devient donc facultatif.

Et enfin pour les puristes, Javascript n'est pas à proprement parler un langage orienté objet tel que C++ ou Java. On dira plutôt que Javascript est un langage basé sur les objets.

## 3.2 LES PROPRIETES DES OBJETS

Une propriété est un attribut, une caractéristique, une description de l'objet. Par exemple, l'objet volant d'une voiture a comme propriétés qu'il peut être en bois ou en cuir. L'objet livre a comme propriétés son auteur, sa maison d'édition, son titre, son numéro ISBN, etc.

De même les objets Javascript ont des propriétés personnalisées. Dans le cas des boutons radio, une de ses propriétés est, par exemple, sa sélection ou sa non-sélection (checked en anglais).

En Javascript, pour accéder aux propriétés, on utilise la syntaxe :

**nom_de_l'objet.nom_de_la_propriété**

Dans le cas du bouton radio "semaine", pour tester la propriété de sélection, on écrira

**document.form.radio[0].checked**

# CHAPITRE 4 : VOS OUTILS POUR LE JAVASCRIPT

Pour apprendre et exploiter le Javascript, il vous faut :

1. un browser qui reconnaît le Javascript.
2. une solide connaissance du Html
3. un simple éditeur de texte

## 4.1 UN BROWSER COMPATIBLE JAVASCRIPT

Uniquement Netscape et Microsoft vous proposent des navigateurs Javascript "enabled". Pour Microsoft à partir de MSIE Explorer 3.0 et Netscape à partir de Netscape Navigator 2.0.

Par contre, il faut être attentif aux versions de Javascript exploitées par ces browers.

| | |
|---|---|
| Netscape 2.0 | Javascript (baptisé à posteriori 1.0) |
| Netscape 3.0 | Javascript 1.1 |
| Netscape 4.0<br><br>(Communicator) | Javascript 1.2 |
| Explorer 3.0 | Quelque chose qui ressemble à du Javascript 1.0 |
| Explorer 4.0 | Javascript 1.2 |

Il faut bien admettre que Javascript est plutôt l'affaire de Netscape et que vous courrez au devant d'une collection d'ennuis en utilisant Explorer 3 pour le Javascript.

## 4.2 UN SOLIDE BAGAGE EN HTML

Comme le code du Javascript vient s'ajouter au "code" du langage Html, une connaissance approfondie des balises ou tags Html est souhaitable sinon indispensable. Ainsi les utilisateurs d'éditeurs Html "whsiwyg" ou autres "publishers" Html risquent de devoir retourner à leurs chères études.

## 4.3 UN BON EDITEUR DE TEXTE

Une page Html n'est que du texte. Le code Javascript n'est lui aussi que du texte. Quoi de plus simple qu'un éditeur de ... texte comme le Notepad de Windows pour inclure votre Javascript dans votre page Html. Un éditeur Html de la première génération (un bon vieil éditeur qui fait encore apparaître les balises), comme HTML Notepad, fait également bien l'affaire.

# CHAPITRE 5 : LE JAVASCRIPT MINIMUM

## 5.1 LA BALISE <SCRIPT>

De ce qui précède, vous savez déjà que votre script vient s'ajouter à votre page Web.

Le langage Html utilise des tags ou balises pour "dire" au browser d'afficher une portion de texte en gras, en italique, etc.

Dans la logique du langage Html, il faut donc signaler au browser par une balise, que ce qui suit est un script et que c'est du Javascript (et non du VBScript). C'est la balise

```
<SCRIPT LANGUAGE="Javascript">.
```

De même, il faudra informer le browser de la fin du script.

C'est la balise

```
</SCRIPT>.
```

## 5.2 LES COMMENTAIRES

Il vous sera peut-être utile d'inclure des commentaires personnels dans vos codes Javascript. C'est même vivement recommandé comme pour tous les langages de programmation (mais qui le fait vraiment ?).

Javascript utilise les conventions utilisées en C et C++ soit

```
// commentaire
```

Tout ce qui est écrit entre le // et la fin de la ligne sera ignoré.

Il sera aussi possible d'inclure des commentaires sur plusieurs lignes avec le code

```
/* commentaire */
```

sur plusieurs lignes

*Ne confondez pas les commentaires Javascript et les commentaires Html (pour rappel <!-- ...-->).*

## *5.3 MASQUER LE SCRIPT POUR LES ANCIENS BROWSERS*

Les browsers qui ne comprennent pas le Javascript (et il y en a encore) ignorent la balise <script> et vont essayer d'afficher le code du script sans pouvoir l'exécuter. Pour éviter l'affichage peu esthétique de ses inscriptions cabalistiques, on utilisera les balises de commentaire du langage Html <!-- ... -->.

Votre premier Javasript ressemblera à ceci :

```
<SCRIPT LANGUAGE="javascript">

<!-- Masquer le script pour les anciens browsers

...

programme Javascript

...

// Cesser de masquer le script -->

</SCRIPT>
```

## *5.4 OU INCLURE LE CODE EN JAVASCRIPT ?*

Le principe est simple. Il suffit de respecter les deux principes suivants :

- n'importe où.

- mais là où il le faut.

Le browser traite votre page Html de haut en bas (y compris vos ajoutes en Javascript). Par conséquent, toute instruction ne pourra être exécutée que si le browser possède à ce moment précis tous les éléments nécessaires à son exécution. Ceux-ci doivent donc être déclarés avant ou au plus tard lors de l'instruction.

Pour s'assurer que le programme script est chargé dans la page et prêt à fonctionner à toute intervention de votre visiteur (il y a des impatients) on prendra l'habitude de déclarer systématiquement (lorsque cela sera possible) un maximum d'éléments dans les balises d'en-tête soit entre <HEAD> et </HEAD> et avant la balise <BODY>. Ce sera le cas par exemple pour les fonctions.

Rien n'interdit de mettre plusieurs scripts dans une même page Html.

## 5.5 UNE PREMIERE INSTRUCTION JAVASCRIPT

Sans vraiment entrer dans les détails, voyons une première instruction Javascript (en fait une méthode de l'objet window) soit l'instruction alert().

```
alert("votre texte");
```

Cette instruction affiche un message (dans le cas présent votre texte entre les guillemets) dans une boite de dialogue pourvue d'un bouton OK. Pour continuer dans la page, le lecteur devra cliquer ce bouton.

Vous remarquerez des points-virgules à la fin de chaque instruction Javascript (ce qui n'est pas sans rappeler le C et le C++).

## 5.6 VOTRE PREMIERE PAGE HTML AVEC DU JAVASCRIPT

| | |
|---|---|
| <HTML> | Html normal |
| <HEAD> | ... |
| <TITLE>Mon premier Javascript</TITLE> | ... |
| </HEAD> | ... |
| <BODY> | ... |
| Bla-bla en Html | ... |
| <SCRIPT LANGUAGE="Javascript"> | Début du script |
| <!-- | Masquer le script |
| alert("votre texte"); | Script |
| //--> | Fin de masquer |
| </SCRIPT> | Fin du script |
| Suite bla-bla en Html | Html normal |
| </BODY> | ... |
| </HTML> | ... |

## 5.7 REMARQUES

Javascript est « case sensitive ». Ainsi il faudra écrire alert() et non Alert(). Pour l'écriture des instructions Javascript, on utilisera l'alphabet ASCII classique (à 128 caractères) comme en Html. Les caractères accentués comme é ou à ne peuvent être employés que dans les chaînes de caractères c.-à-d. dans votre texte de notre exemple.

Les guillemets " et l'apostrophe ' font partie intégrante du langage Javascript. On peut utiliser l'une ou l'autre forme à condition de ne pas les mélanger. Ainsi alert("...') donnera un message d'erreur. Si vous souhaiter utiliser des guillemets dans vos chaînes de caractères, tapez \" ou \' pour les différencier vis à vis du compilateur.

## +5.8 VERSIONS DU LANGAGE JAVASCRIPT

Avec les différentes versions déjà existantes (Javascript 1.0, Javascript 1.1 et Javascript 1.2), on peut imaginer des scripts adaptés aux différentes versions mais surtout aux différents navigateurs .

```
<SCRIPT LANGUAGE="Javascript">

// programme pour Netscape 2 et Explorer 3

var version="1.0";

</SCRIPT>


<SCRIPT LANGUAGE="Javascript1.1">

// programme pour Netcape 3 et Explorer 4

var version=1.1;

</SCRIPT>


<SCRIPT LANGUAGE="Javascript1.2">

// programme pour Netscape 4

var version=1.2;

</SCRIPT>


<SCRIPT LANGUAGE="Javascript">

document.write('Votre browser supporte le Javascript ' + version);

</SCRIPT>
```

### +5.9 EXTENSION .JS POUR SCRIPTS EXTERNES

Il est possible d'utiliser des fichiers externes pour les programmes Javascript. On peut ainsi stocker les scripts dans des fichiers distincts (avec l'extension .js) et les appeler à partir d'un fichier Html. Le concepteur peut de cette manière se constituer une bibliothèque de script et les appeler à la manière des #include du C ou C++. La balise devient

```
<SCRIPT LANGUAGE='javascript' SRC='http://site.com/javascript.js'></SCRIPT>
```

### +5.10 TOUJOURS DES COMMENTAIRES

Outre les annotations personnelles, les commentaires peuvent vous être d'une utilité certaine en phase de débogage d'un script pour isoler (sans effacer) une ligne suspecte.

Pour les esprits compliqués, notons que les commentaires ne peuvent être imbriqués sous peine de message d'erreur. La formulation suivante est donc à éviter :

/* script réalisé ce jour /* jour mois */

et testé par nos soins*/

### +5.11 ALERT() ... ROUGE

Joujou des débutants en Javascript, cette petite fenêtre est à utiliser avec parcimonie pour attirer l'attention du lecteur pour des choses vraiment importantes. Et puis, elles ne sont vraiment pas destinées à raconter sa vie. Javascript met à votre disposition la possibilité de créer de nouvelles fenêtres de la dimension de votre choix qui apparaissent un peu comme les popup des fichiers d'aide. Nous les étudierons plus loin dans l'objet Window.

Alert() est une méthode de l'objet Window. Pour se conformer à la notation classique nom_de_l'objet.nom_de_la_propriété, on aurait pu noter window.alert(). Window venant en tête des objets Javascript, celui-ci est repris par défaut par l'interpréteur et devient en quelque sorte facultatif.

Si vous souhaitez que votre texte de la fenêtre alert() s'inscrive sur plusieurs lignes, il faudra utiliser le caractère spécial /n pour créer une nouvelle ligne.

# CHAPITRE 6 : AFFICHER DU TEXTE

## 6.1 METHODE DE L'OBJET DOCUMENT

Rappelez-vous... Nous avions montré que ce qui apparaît sur votre écran, peut être "découpé" en objets et que Javascript allait vous donner la possibilité d'accéder à ces objets (Un peu de théorie objet). La page Html qui s'affiche dans la fenêtre du browser est un objet de type document.

A chaque objet Javascript, le concepteur du langage a prévu un ensemble de méthodes (ou fonctions dédiées à cet objet) qui lui sont propres. A la méthode document, Javascript a dédié la méthode "écrire dans le document", c'est la méthode write().

L'appel de la méthode se fait selon la notation :

```
nom_de_l'objet.nom_de_la_méthode
```

Pour appeler la méthode write() du document, on notera

```
document.write();
```

## 6.2 LA METHODE WRITE()

La syntaxe est assez simple soit

```
write("votre texte");
```

On peut aussi écrire une variable, soit la variable resultat,

```
write(resultat);
```

Pour associer du texte (chaînes de caractères) et des variables, on utilise l'instruction

```
 write("Le résultat est " + resultat);
```

On peut utiliser les balises Html pour agrémenter ce texte

```
write("<B>Le résultat est</B>" + resultat); ou

write ("<B>" + "Le résultat est " + "</B>" + resultat)
```

### 6.3 EXEMPLE (CLASSIQUE !)

On va écrire du texte en Html et en Javascript.

```
<HTML>
<BODY>
<H1>Ceci est du Html</H1>
<SCRIPT LANGUAGE="Javascript">
<!--
document.write("<H1>Et ceci du Javascript</H1>");
//-->
</SCRIPT>
</BODY>
</HTML>
```

Ce qui donnera comme résultat :

# Ceci est du Html

Et ceci du Javascript

### + 6.4 L'INSTRUCTION WRITELN()

La méthode writeln() est fort proche de write() à ceci près qu'elle ajoute un retour chariot à la fin des caractères affichés par l'instruction. Ce qui n'a aucun effet en Html. Pour faire fonctionner write() Il faut l'inclure dans des balises <PRE>.

```
<PRE>
<SCRIPT LANGUAGE="Javascript">
<--
document.writeln("Ligne 1");
document.writeln("Ligne 2");
//-->
</SCRIPT>
</PRE>
```

## +6.5 DE LA BELLE ECRITURE EN JAVASCRIPT ...

### 6.5.1 variable.big();

L'emploi de .big() affichera la variable comme si elle était comprise entre les balises Html <BIG></BIG>.

Les quatre instructions Javascript suivantes sont équivalentes :

```
str="Something"; (str est une variable)

document.write("<BIG>"+str+"</BIG>");

document.write('<BIG>Something</BIG>');

document.write(str.big());

document.write("Something".big());
```

### 6.5.2 variable.small();

L'emploi de .small() affichera la variable comme si elle était comprise entre les balises Html <SMALL> </SMALL>.

Les quatre instructions Javascript suivantes sont équivalentes :

```
str="Something";

document.write("<SMALL>"+str +"</SMALL>");

document.write("<SMALL>Something" +"</SMALL>");

document.write(str.small());

document.write("Something".small());
```

### 6.5.3 variable.blink();

L'emploi de .blink() affichera la variable comme si elle était comprise entre les balises Html <BLINK></BLINK>. Pour rappel, cette balise (qui est par ailleurs vite ennuyeuse) n'est valable que sous Netscape 3 et plus.

Les quatre instructions Javascript suivantes sont équivalentes :

```
str="Something";

document.write('<BLINK>'+str+'</BLINK>');

document.write("<BLINK>Something</BLINK>");

document.write(str.blink());

document.write("Something".blink());
```

### 6.5.4 variable.bold();

L'emploi de .bold() affichera la variable comme si elle était comprise entre les balises Html <B></B>.

Les quatre instructions Javascript suivantes sont équivalentes :

```
str="Some words";

document.write("<B>"+str+"</B>");

document.write("<B>Some words</B>");

document.write(str.bold());

document.write("Some words".bold());
```

### 6.5.5 variable.fixed();

L'emploi de .fixed() affichera la variable comme si elle était comprise entre les balises Html <TT></TT>.

Les quatre instructions Javascript suivantes sont équivalentes :

```
str="Something";

document.write("<TT>"+str+"</TT>");

document.write("<TT>Something</TT>");

document.write(str.fixed());

document.write("Something".fixed());
```

### 6.5.6 variable.italics();

L'emploi de .italics() affichera la variable comme si elle était comprise entre les balises Html <I></I>.

Les quatre instructions Javascript suivantes sont équivalentes :

```
str="Something";

document.write("<I>"+str+"</I>");

document.write("<I>Something</I>");

document.write(str.italics());

document.write("Some word".italics());
```

### 6.5.7 variable.fontcolor(color );

L'emploi de .fontcolor(color) affichera la variable comme si elle était comprise entre les balises Html <FONT COLOR="color"> </FONT>.

Les quatre instructions Javascript suivantes sont équivalentes :

```
str1="Some words";

str2="red";

document.write("<FONT COLOR='red'>" +str1+"</FONT>");

document.write("<FONT COLOR='red'>" +"Something</FONT>");

document.write(str1.fontcolor(str2));

document.write(str1.fontcolor("red"));
```

### 6.5.8 variable.fontsize(x);

L'emploi de .fontsize(x) affichera la variable comme si elle était comprise entre les balises Html <FONT SIZE="x"></FONT> où x est un nombre de 1 à 7 ou exprimé en plus ou en moins par rapport à 0 par exemple -2, -1, +1, +2.

Les quatre instructions Javascript suivantes sont équivalentes :

```
str="Something";

x=3;

document.write("<FONT SIZE=3>" +str+"</FONT>");

document.write("<FONT SIZE=3>" +"Something</FONT>");

document.write(str.fontsize(3));

document.write(str.fontsize(x));
```

### 6.5.9 variable.strike();

L'emploi de .strike() affichera la variable comme si elle était comprise entre les balises Html

<STRIKE></STRIKE>.

Les quatre instructions Javascript suivantes sont équivalentes :

```
str="Something";

document.write("<STRIKE>"+str +"</STRIKE>");

document.write("<STRIKE>Something" +"</STRIKE>");

document.write(str.strike());

document.write("Something".strike());
```

### 6.5.10 variable.sub();

L'emploi de .sub() affichera la variable comme si elle était comprise entre les balises Html <SUB> </SUB>.

Les quatre instructions Javascript suivantes sont équivalentes :

```
str="Something";

document.write("<SUB>"+str+"</SUB>");

document.write("<SUB>Something" +"</SUB>");

document.write(str.sub());

document.write("Something".sub());
```

### 6.5.11 variable.sup();

L'emploi de .sup() affichera la variable comme si elle était comprise entre les balises Html <SUP></SUB>.

Les quatre instructions Javascript suivantes sont équivalentes :

```
str="Something";

document.write("<SUP>"+str+"</SUP>");

document.write("<SUP>Something</SUP>");

document.write(str.sup());

document.write("Something".sup());
```

## +6.6 LES INSTRUCTIONS DE FORMATAGE DE DOCUMENT

Rappelons tout d'abord que ce qui suit est optionnel et que vous pouvez utiliser l'instruction document.write() de façon tout à fait classique.

Soit document.write("<BODY BGCOLOR="#FFFFFF");

### 6.6.1document.bgColor

Cette instruction permet de spécifier la couleur d'arrière-plan d'un objet document. On peut employer le nom ou la valeur RGB de la couleur.

```
document.bgColor="white";

document.bgColor="#FFFFFF";
```

### 6.6.2 document.fgColor

Cette instruction permet de spécifier la couleur d'avant-plan (texte) d'un objet document. On peut employer le nom ou la valeur RGB de la couleur.

```
document.fgColor="black";

document.fgColor="#000000";
```

### 6.6.3 document.alinkColor

Cette instruction permet de spécifier la couleur d'un lien actif (après le clic de la souris mais avant de quitter le lien) d'un objet document. On peut employer le nom ou la valeur RGB de la couleur.

```
document.alinkColor="white";

document.alinkColor="#FFFFFF";
```

### 6.6.4 document.linkColor

Cette instruction permet de spécifier la couleur d'un hyperlien d'un objet document. On peut employer le nom ou la valeur RGB de la couleur.

```
document.linkColor="white";

document.linkColor="#FFFFFF";
```

### 6.6.5 document.vlinkColor

Cette instruction permet de spécifier la couleur d'un hyperlien déjà visité d'un objet document. On peut employer le nom ou la valeur RGB de la couleur.

```
document.linkColor="white";

document.linkColor="#FFFFFF";
```

# CHAPITRE 7 : UTILISER DES VARIABLES

## 7.1 LES VARIABLES EN JAVASCRIPT

Les variables contiennent des données qui peuvent être modifiées lors de l'exécution d'un programme. On y fait référence par le nom de cette variable.

Un nom de variable doit commencer par une lettre (alphabet ASCII) ou le signe_ et se composer de lettres, de chiffres et des caractères _ et $ (à l'exclusion du blanc). Le nombre de caractères n'est pas précisé. Pour rappel Javascript est sensible à la case. Attention donc aux majuscules et minuscules!

## 7.2 LA DECLARATION DE VARIABLE

Les variables peuvent se déclarer de deux façons :

- soit de façon explicite. On dit à Javascript que ceci est une variable. La commande qui permet de déclarer une variable est le mot var. Par exemple :

```
var Numero = 1
var Prenom = "Luc"
```

- soit de façon implicite. On écrit directement le nom de la variable suivi de la valeur que l'on lui attribue et Javascript s'en accommode. Par exemple :

```
Numero = 1
Prenom = "Luc"
```

Attention! Malgré cette apparente facilité, la façon dont on déclare la variable aura une grande importance pour la "visibilité" de la variable dans le programme Javascript. Voir à ce sujet, la distinction entre variable locale et variable globale dans le Javascript avancé de ce chapitre.

Pour la clarté de votre script et votre facilité, on ne peut que conseiller d'utiliser à chaque fois le mot var pour déclarer une variable.

## 7.3 LES DONNEES SOUS JAVASCRIPT

Javascript utilise 4 types de données :

| Type | Description |
|---|---|
| Des nombres | Tout nombre entier ou avec virgule tel que 22 ou 3.1416 |
| Des chaînes de caractères | Toute suite de caractères comprise entre guillemets telle que "suite de caractères" |
| Des booléens | Les mots true pour vrai et fase pour faux |
| Le mot null | Mot spécial qui représente pas de valeur |

Notons aussi que contrairement au langage C ou C++, Il ne faut pas déclarer le type de données d'une variable. On n'a donc pas besoin de int, float, double, char et autres long en Javascript.

## 7.4 EXERCICE

Nous allons employer la méthode write() pour afficher des variables. On définit une variable appelée texte qui contient une chaîne de caractères "Mon chiffre préféré est " et une autre appelée variable qui est initialisée à 7.

```
<HTML>
<BODY>
<SCRIPT LANGUAGE="Javascript">
<!--
var texte = "Mon chiffre préféré est le "
var variable = 7
document.write(texte + variable);
//-->
</SCRIPT>
</BODY>
</HTML>
```

Le résultat se présente comme suit :

Mon chiffre préféré est le 7

## +7.5 LES NOMS RESERVES

Les mots de la liste ci-après ne peuvent être utilisés pour des noms de fonctions et de variables .Certains de ces mots sont des mots clés Javascript, d'autres ont été réservés par Netscape pour un futur usage éventuel.

| | |
|---|---|
| A | abstract |
| B | boolean break byte |
| C | case catch char class const continue |
| D | default do double |
| E | else extends |
| F | false final finally float for function |
| G | goto |
| I | if implements import in instanceof int interface |
| L | long |
| N | native new null |
| P | package private protected public |
| R | return |
| S | short static super switch synchronized |
| T | this throw throws transient true try |
| V | var void |
| W | while with |

## +7.6 VARIABLES GLOBALES ET VARIABLES LOCALES

Les variables déclarées tout au début du script, en dehors et avant toutes fonctions (voir plus loin...), seront toujours globales, qu'elles soient déclarées avec var ou de façon contextuelle. On pourra donc les exploiter partout dans le script.

Dans une fonction, une variable déclarée par le mot clé var aura une portée limitée à cette seule fonction. On ne pourra donc pas l'exploiter ailleurs dans le script. D'où son nom de locale. Par contre, toujours dans une fonction, si la variable est déclarée contextuellement (sans utiliser le mot var), sa portée sera globale.

Nous reviendrons sur tout ceci dans l'étude des fonctions.

# CHAPITRE 8 : LES OPERATEURS

Les variables, c'est bien mais encore faut-il pouvoir les manipuler ou les évaluer. Voyons (et ce n'est peut-être pas le chapitre le plus marrant de ce tutorial) les différents opérateurs mis à notre disposition par Javascript.

## 8.1 LES OPERATEURS DE CALCUL

Dans les exemples, la valeur initiale de x sera toujours égale à 11.

| Signe | Nom | Signification | Exemple | Résultat |
|---|---|---|---|---|
| + | plus | addition | x + 3 | 14 |
| - | moins | soustraction | x - 3 | 8 |
| * | multiplié par | multiplication | x*2 | 22 |
| / | divisé | par division | x /2 | 5.5 |
| % | modulo | reste de la division par | x%5 | 1 |
| = | a la valeur | affectation | x=5 | 5 |

## 8.2 LES OPERATEURS DE COMPARAISON

| Signe | Nom | Exemple | Résultat |
|---|---|---|---|
| | | | |
| == | égal | x==11 | true |
| < | inférieur | x<11 | false |
| <= | inférieur ou égal | x<=11 | true |
| > | supérieur | x>11 | false |
| =< | supérieur ou égal | x>=11 | true |
| != | différent | x!=11 | false |

Important. On confond souvent le = et le == (deux signes =). Le = est un opérateur d'attribution de valeur tandis que le == est un opérateur de comparaison. Cette confusion est une source classique d'erreur de programmation.

## 8.3 LES OPERATEURS ASSOCIATIFS

On appelle ainsi les opérateurs qui réalisent un calcul dans lequel une variable intervient des deux côtés du signe = (ce sont donc en quelque sorte également des opérateurs d'attribution).

Dans les exemples suivants x vaut toujours 11 et y aura comme valeur 5.

| Signe | Description | Exemple | Signification | Résultat |
|---|---|---|---|---|
| += | plus égal | x += y | x = x + y | 16 |
| -= | moins égal | x -= y | x = x - y | 6 |
| *= | multiplié égal | x *= y | x = x * y | 55 |
| /= | divisé égal | x /= y | x = x / y | 2.2 |

## 8.4 LES OPERATEURS LOGIQUES

Aussi appelés opérateurs booléens, ses opérateurs servent à vérifier deux ou plusieurs conditions.

| Signe | Nom | Exemple | Signification |
|---|---|---|---|
| && | et | (condition1) && (condition2) | condition1 et condition2 |
| \|\| | ou | (condition1) \|\| (condition2) | condition1 ou condition2 |

## 8.5 LES OPERATEURS D'INCREMENTATION

Ces opérateurs vont augmenter ou diminuer la valeur de la variable d'une unité. Ce qui sera fort utile, par exemple, pour mettre en place des boucles.

Dans les exemples x vaut 3.

| Signe | Description | Exemple | Signification | Résultat |
|---|---|---|---|---|
| x++ | incrémentation<br>(x++ est le même que x=x+1) | y = x++ | 3 puis plus 1 | 4 |
| x-- | décrémentation<br>(x-- est le même que x=x-1) | y= x-- | 3 puis moins 1 | 2 |

## +8.6 LA PRIORITE DES OPERATEURS JAVASCRIPT

Les opérateurs s'effectuent dans l'ordre suivant de priorité (du degré de priorité le plus faible ou degré de priorité le plus élevé).

Dans le cas d'opérateurs de priorité égale, de gauche à droite.

| Opération | Opérateur |
|---|---|
| , | virgule ou séparateur de liste |
| = += -= *= /= %= | affectation |
| ? : | opérateur conditionnel |
| \|\| | ou logique |
| && | et logique |
| == != | égalité |
| < <= >= > | relationnel |
| + - | addition soustraction |
| * / | multiplier diviser |
| ! - ++ -- | unaire |
| ( ) | parenthèses |

# CHAPITRE 9 : LES FONCTIONS

## 9.1 DEFINITION

Une fonction est un groupe de ligne(s) de code de programmation destiné à exécuter une tâche bien spécifique et que l'on pourra, si besoin est, utiliser à plusieurs reprises. De plus, l'usage des fonctions améliorera grandement la lisibilité de votre script.

En Javascript, il existe deux types de fonctions :

- les fonctions propres à Javascript. On les appelle des "méthodes". Elles sont associées à un objet bien particulier comme c'était le cas de la méthode Alert() avec l'objet window.
- les fonctions écrites par vous-même pour les besoins de votre script. C'est à celles-là que nous nous intéressons maintenant.

## 9.2 DECLARATION DES FONCTIONS

Pour déclarer ou définir une fonction, on utilise le mot (réservé) function.

La syntaxe d'une déclaration de fonction est la suivante :

```
function nom_de_la_fonction(arguments)
{
... code des instructions ...
}
```

Le nom de la fonction suit les mêmes règles que celles qui régissent le nom de variables (nombre de caractères indéfini, commencer par une lettre, peuvent inclure des chiffres...). Pour rappel, Javascript est sensible à la case. Ainsi fonction() ne sera pas égal à Fonction(). En outre, Tous les noms des fonctions dans un script doivent être uniques.

La mention des arguments est facultative mais dans ce cas les parenthèses doivent rester. C'est d'ailleurs grâce à ces parenthèses que l'interpréteur Javascript distingue les variables des fonctions. Nous reviendrons plus en détail sur les arguments et autres paramètres dans la partie Javascript avancé.

Lorsque une accolade est ouverte, elle doit impérativement, sous peine de message d'erreur, être refermée. Prenez la bonne habitude de fermer directement vos accolades et d'écrire votre code entre elles.

Le fait de définir une fonction n'entraîne pas l'exécution des commandes qui la composent. Ce n'est que lors de l'appel de la fonction que le code de programme est exécuté.

## 9.3 L'APPEL D'UNE FONCTION

L'appel d'une fonction se fait le plus simplement du monde par le nom de la fonction (avec les parenthèses).

Soit par exemple `nom_de_la_fonction();`

Il faudra veiller en toute logique (car l'interpréteur lit votre script de haut vers le bas) à ce que votre fonction soit bien définie avant d'être appelée.

## 9.4 LES FONCTIONS DANS <HEAD>...</HEAD>

Il est donc prudent ou judicieux de placer toutes les déclarations de fonction dans l'en-tête de votre page c.-à-d .dans la balise <HEAD> ...</HEAD>. Vous serez ainsi assuré que vos fonctions seront déjà prises en compte par l'interpréteur avant qu'elles soient appelées dans le <BODY>.

## 9.5 EXEMPLE

Dans cet exemple, on définit dans les balises HEAD, une fonction appelée message() qui affiche le texte "Bienvenue à ma page". cette fonction sera appelée au chargement de la page voir onLoad=.... dans le tag <BODY>.

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="Javascript">
<--
function message() {
document.write("Bienvenue à ma page");
}
//-->
</SCRIPT>
</HEAD>
<BODY onLoad="message()">
</BODY>
</HTML>
```

## +9.6 PASSER UNE VALEUR A UNE FONCTION

On peut passer des valeurs ou paramètres aux fonctions Javascript. La valeur ainsi passée sera utilisée par la fonction.

Pour passer un paramètre à une fonction, on fournit un nom d'une variable dans la déclaration de la fonction.

Un exemple un peu simplet pour comprendre. J'écris une fonction qui affiche une boite d'alerte dont le texte peut changer.

Dans la déclaration de la fonction, on écrit :

```
function Exemple(Texte)
{
alert(texte);
}
```

Le nom de la variable est Texte et est définie comme un paramètre de la fonction.

Dans l'appel de la fonction, on lui fournit le texte :

```
Exemple("Salut à tous");
```

MCOURS.COM
Le N°1 du cours et exercices sur Internet

## +9.7 PASSER PLUSIEURS VALEURS A UNE FONCTION

On peut passer plusieurs paramètres à une fonction. Comme c'est souvent le cas en Javascript, on sépare les paramètres par des virgules.

```
function nom_de_la_fonction(arg1, arg2, arg3)
{
... code des instructions ...
}
```

Notre premier exemple devient pour la déclaration de fonction :

```
function Exemplebis(Texte1, Texte2){...}
```

et pour l'appel de la fonction :

```
Exemplebis("Salut à tous", "Signé Luc")
```

## +9.8 RETOURNER UNE VALEUR

Le principe est simple (la pratique parfois moins). Pour renvoyer un résultat, il suffit d'écrire le mot clé return suivi de l'expression à renvoyer. Notez qu'il ne faut pas entourer l'expression de parenthèses. Par exemple :

```
function cube(nombre)
{
var cube = nombre*nombre*nombre
return cube;
}
```

Précisons que l'instruction return est facultative et qu'on peut trouver plusieurs return dans une même fonction.

Pour exploiter cette valeur de la variable retournée par la fonction, on utilise une formulation du type document.write(cube(5)).

## +9.9 VARIABLES LOCALES ET VARIABLES GLOBALES

Avec les fonctions, le bon usage des variables locales et globales prend toute son importance.

Une variable déclarée dans une fonction par le mot clé var aura une portée limitée à cette seule fonction. On ne pourra donc pas l'exploiter ailleurs dans le script. On l'appelle donc variable locale.

```
function cube(nombre)
{
var cube = nombre*nombre*nombre
}
```

Ainsi la variable cube dans cet exemple est une variable locale. Si vous y faites référence ailleurs dans le script, cette variable sera inconnue pour l'interpréteur Javascript (message d'erreur).

Si la variable est déclarée contextuellement (sans utiliser le mot var), sa portée sera globale -- et pour être tout à fait précis, une fois que la fonction aura été exécutée--.

```
function cube(nombre)
{
cube = nombre*nombre*nombre
}
```

La variable cube déclarée contextuellement sera ici une variable globale.

Les variables déclarées tout au début du script, en dehors et avant toutes fonctions, seront toujours globales, qu'elles soient déclarées avec var ou de façon contextuelle.

```
<SCRIPT LANGUAGE="javascript">
var cube=1
function cube(nombre)
{
var cube = nombre*nombre*nombre
}
</SCRIPT>
```

La variable cube sera bien globale.

Pour la facilité de gestion des variables, on ne peut que conseiller de les déclarer en début de script (comme dans la plupart des langages de programmation). Cette habitude vous met à l'abri de certaines complications.

# CHAPITRE 10 : LES EVENEMENTS

## 10.1 GENERALITES

Avec les événements et surtout leur gestion, nous abordons le côté "*magique*" de Javascript.

En Html classique, il y a un événement que vous connaissez bien. C'est le clic de la souris sur un lien pour vous transporter sur une autre page Web. Hélas, c'est à peu près le seul. Heureusement, Javascript va en ajouter une bonne dizaine, pour votre plus grand plaisir.

Les événements Javascript, associés aux fonctions, aux méthodes et aux formulaires, ouvrent grand la porte pour une réelle *interactivité* de vos pages.

## 10.2 LES EVENEMENTS

Passons en revue différents événements implémentés en Javascript.

| Description | Evénement |
|---|---|
| Lorsque l'utilisateur clique sur un bouton, un lien ou tout autre élément. | Clik |
| Lorsque la page est chargée par le browser ou le navigateur. | Load |
| Lorsque l'utilisateur quitte la page. | Unload |
| Lorsque l'utilisateur place le pointeur de la souris sur un lien ou tout autre élément. | MouseOver |
| Lorsque le pointeur de la souris quitte un lien ou tout autre élément.<br><br>Attention : Javascript 1.1 (donc pas sous MSIE 3.0 et Netscape 2). | MouseOut |
| Lorsque un élément de formulaire a le focus c-à-d devient la zone d'entrée active. | Focus |
| Lorsque un élément de formulaire perd le focus c-à-d que l'utilisateur clique hors du champs et que la zone d'entrée n'est plus active. | Blur |
| Lorsque la valeur d'un champ de formulaire est modifiée. | Change |
| Lorsque l'utilisateur sélectionne un champ dans un élément de formulaire. | Select |
| Lorsque l'utilisateur clique sur le bouton Submit pour envoyer un formulaire. | Submit |

## 10.3 LES GESTIONNAIRES D'EVENEMENTS

Pour être efficace, il faut qu'à ces événements soient associées les actions prévues par vous. C'est le rôle des gestionnaires d'événements. La syntaxe est

```
onévénement="fonction()"
```

Par exemple, onClick="alert('Vous avez cliqué sur cet élément')".

De façon littéraire, au clic de l'utilisateur, ouvrir une boite d'alerte avec le message indiqué.

### 10.3.1 onclick

Evénement classique en informatique, le clic de la souris.

Le code de ceci est :

```
<FORM>
<INPUT TYPE="button" VALUE="Cliquez ici" onClick="alert('Vous avez bien cliqué ici')">
</FORM>
```

Nous reviendrons en détail sur les formulaires dans le chapitre suivant.

### 10.3.2 onLoad et onUnload

L'événement Load survient lorsque la page a fini de se charger. A l'inverse, Unload survient lorsque l'utilisateur quitte la page.

Les événements onLoad et onUnload sont utilisés sous forme d'attributs de la balise <BODY> ou <FRAMESET>. On peut ainsi écrire un script pour souhaiter la bienvenue à l'ouverture d'une page et un petit mot d'au revoir au moment de quitter celle-ci.

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE='Javascript'>
function bienvenue() {
alert("Bienvenue à cette page");
}
function au_revoir() {
alert("Au revoir");
}
</SCRIPT>
</HEAD>
<BODY onLoad='bienvenue()' onUnload='au_revoir()'>
Html normal
</BODY>
</HTML>
```

### 10.3.4 onmouseOver et onmouseOut

L'événement onmouseOver se produit lorsque le pointeur de la souris passe au dessus (sans cliquer) d'un lien ou d'une image. Cet événement est fort pratique pour, par exemple, afficher des explications soit dans la barre de statut soit avec une petite fenêtre genre infobulle.

L'événement onmouseOut, généralement associé à un onmouseOver, se produit lorsque le pointeur quitte la zone sensible (lien ou image).

Notons qui si onmouseOver est du Javascript 1.0, onmouseOut est du Javascript 1.1.

En clair, onmouseOut ne fonctionne pas avec Netscape 2.0 et Explorer 3.0.

### 10.3.5 onFocus

L'événement onFocus survient lorsqu'un champ de saisie a le focus c.-à-d. quand son emplacement est prêt à recevoir ce que l'utilisateur à l'intention de taper au clavier. C'est souvent la conséquence d'un clic de souris ou de l'usage de la touche "Tab".

### 10.3.6 onBlur

L'événement onBlur a lieu lorsqu'un champ de formulaire perd le focus. Cela se produit quand l'utilisateur ayant terminé la saisie qu'il effectuait dans une case, clique en dehors du champ ou utilise la touche "Tab" pour passer à un champ. Cet événement sera souvent utilisé pour vérifier la saisie d'un formulaire.

Le code est :

```
<FORM>
<INPUT TYPE=text onBlur="alert('Ceci est un Blur')">
</FORM>
```

### 10.3.7 onchange

Cet événement s'apparente à l'événement onBlur mais avec une petite différence. Non seulement la case du formulaire doit avoir perdu le focus mais aussi son contenu doit avoir été modifié par l'utilisateur.

### 10.3.8 onselect

Cet événement se produit lorsque l'utilisateur a sélectionné (mis en surbrillance ou en vidéo inverse) tout ou partie d'une zone de texte dans une zone de type text ou textarea.

## +10.4 GESTIONNAIRES D'EVENEMENT DISPONIBLES EN JAVASCRIPT

Il nous semble utile dans cette partie "avancée" de présenter la liste des objets auxquels correspondent des gestionnaires d'événement bien déterminés.

| Objets | Gestionnaires d'événement disponibles |
|---|---|
| Fenêtre | onLoad, onUnload |
| Lien hypertexte | onClick, onmouseOver, on mouseOut |
| Elément de texte | onBlur, onChange, onFocus, onSelect |
| Elément de zone de texte | onBlur, onChange, onFocus, onSelect |
| Elément bouton | onClick |
| Case à cocher | onClick |
| Bouton Radio | onClick |
| Liste de sélectionon | Blur, onChange, onFocus |
| Bouton Submit | onClick |
| Bouton Reset | onClick |

## +10.5 LA SYNTAXE DE ONMOUSEOVER

Le code du gestionnaire d'événement onmouseOver s'ajoute aux balises de lien :

```
<A HREF="" onmouseOver="action()">lien</A>
```

Ainsi, lorsque l'utilisateur passe avec sa souris sur le lien, la fonction action() est appelée. L'attribut HREF est indispensable. Il peut contenir l'adresse d'une page Web si vous souhaitez que le lien soit actif ou simplement des guillemets si aucun lien actif n'est prévu. Nous reviendrons ci-après sur certains désagréments du codage HREF="".

Voici un exemple. Par le survol du lien "message important", une fenêtre d'alerte s'ouvre.

Le code est :

```
<BODY>
...
<A HREF="" onmouseOver="alert('Coucou')">message important</A>


</BODY>
```

ou si vous préférez utiliser les balises <HEAD>

```
<HTML>
<HEAD>
<SCRIPT language="Javascript">
function message(){
alert("Coucou")
}
</SCRIPT>
</HEAD>
<BODY>
<A HREF="" onmouseOver="message()">message important</A>
</BODY>
</HTML>
```

## +10.6 LA SYNTAXE DE ONMOUSEOUT

Tout à fait similaire à onmouseOver, sauf que l'événement se produit lorsque le pointeur de la souris quitte le lien ou la zone sensible.

Au risque de nous répéter, si onmouseOver est du Javascript 1.0 et sera donc reconnu par tous les browsers, onmouseOut est du Javascript 1.1 et ne sera reconnu que par Netscape 3.0 et plus et Explorer 4.0 et plus (et pas par Netscape 2.0 et Explorer 3.0)

On peut imaginer le code suivant :

```
<A HREF="" onmouseOver="alert('Coucou')" onmouseOut="alert('Au revoir')">message important</A>
```

Les puristes devront donc prévoir une version différente selon les versions Javascript.

## *+10.7 PROBLEME! ET SI ON CLIQUE QUAND MEME...*

Vous avez codé votre instruction onmouseOver avec le lien fictif <A HREF=""... >, vous avez même prévu un petit texte, demandant gentiment à l'utilisateur de ne pas cliquer sur le lien et comme de bien entendu celui-ci clique quand même.

Horreur, le browser affiche alors l'entièreté des répertoires de sa machine ou de votre site). Ce qui est un résultat non désiré et pour le moins imprévu.

Pour éviter cela, prenez l'habitude de mettre l'adresse de la page encours ou plus simplement le signe # (pour un ancrage) entre les guillemets de HREF. Ainsi, si le lecteur clique quand même sur le lien, au pire, la page encours sera simplement rechargée et sans perte de temps car elle est déjà dans le cache du navigateur.

Prenez donc l'habitude de mettre le code suivant

```
<A HREF="#" onmouseOver="action()"> lien </A>.
```

## +10.8 CHANGEMENT D'IMAGES

Avec le gestionnaire d'événement onmouseOver, on peut prévoir qu'après le survol d'un image par l'utilisateur, une autre image apparaisse (pour autant qu'elle soit de la même taille).

Le code est relativement simple.

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="Javascript1.1">
function lightUp() {
document.images["homeButton"].src="button_hot.gif"
}
function dimDown() {
document.images["homeButton"].src="button_dim.gif"
}
</SCRIPT>
</HEAD>
<BODY>
<A HREF="#" onmouseOver="lightUp();" onmouseOut="dimDown();">
<IMG SRC="button_dim.gif" name="homeButton" width=100 height=50 border=0>
</A>
</BODY>
</HTML>
```

Compléter toujours en Javascript les attributs width=x height=y de vos images.

Il n'y a pas d'exemple ici pour la compatibilité avec les lecteurs utilisant explorer 3.0 en effet, non seulement onmouseOut mais aussi image[] est du Javascript 1.1.

## +10.9 L'IMAGE INVISIBLE

Ce changement d'image ne vous donne-t-il pas des idées?... Petit futé! Et oui, on peut prévoir une image invisible de la même couleur que l'arrière plan (même transparente). On la place avec malice sur le chemin de la souris de l'utilisateur et son survol peut ,à l'insu de l'utilisateur, déclencher un feu d'artifice d'actions de votre choix. Magique le Javascript ?

# CHAPITRE 11 : LES CONDITIONS

## 11.1 SI MAMAN SI ..." OU L'EXPRESSION IF

A un moment ou à un autre de la programmation, on aura besoin de tester une condition. Ce qui permettra d'exécuter ou non une série d'instructions.

Dans sa formulation la plus simple, l'expression if se présente comme suit

```
if (condition vraie)
{
une ou plusieurs instructions;
}
```

Ainsi, si la condition est vérifiée, les instructions s'exécutent. Si elle ne l'est pas, les instructions ne s'exécutent pas et le programme passe à la commande suivant l'accolade de fermeture.

De façon un peu plus évoluée, il y a l'expression if...else

```
if (condition vraie) {
instructions1;
}
else {
instructions2;
}
```

Si la condition est vérifiée (true), le bloc d'instructions 1 s'exécute. Si elle ne l'est pas (false), le bloc d'instructions 2 s'exécute.

Dans le cas où il n'y a qu'une instruction, les accolades sont facultatives.

Grâce aux opérateurs logiques "et" et "ou", l'expression de test pourra tester une association de conditions. Ainsi if ((condition1) && (condition2)), testera si la condition 1 et la condition 2 est réalisée. Et if ((condition1) || (condition2)), testera si une au moins des conditions est vérifiée.

Pour être complet (et pour ceux qui aiment les écritures concises), il y a aussi :

```
(expression) ? instruction a : instruction b
```

Si l'expression entre parenthèse est vraie, l'instruction a est exécutée.Si l'expression entre parenthèses retourne faux, c'est l'instruction b qui est exécutée.

## 11.2 L'EXPRESSION FOR

L'expression for permet d'exécuter un bloc d'instructions un certain nombre de fois en fonction de la réalisation d'un certain critère. Sa syntaxe est :

```
for (valeur initiale ; condition ; progression) {
instructions;
}
```

Prenons un exemple concret :

```
for (i=0, i<10, i++) {
document.write(i + "<BR>")
}
```

Au premier passage, la variable i, étant initialisée à 0, vaut bien entendu 0. Elle est bien inférieure à 10. Elle est donc incrémentée d'une unité par l'opérateur d'incrémentation i++ (i vaut alors 2) et les instructions s'exécutent.

A la fin de l'exécution des instructions, on revient au compteur. La variable i (qui vaut 2) est encore toujours inférieure à 10. Elle est augmentée de 1 et les instructions sont à nouveau exécutées. Ainsi de suite jusqu'à ce que i vaille 10. La variable i ne remplit plus la condition i<10. La boucle s'interrompt et le programme continue après l'accolade de fermeture.

## +11.3 WHILE

L'instruction while permet d'exécuter une instruction (ou un groupe d'instructions) un certain nombre de fois.

```
while (condition vraie){
continuer à faire quelque chose
}
```

Aussi longtemps que la condition est vérifiée, Javascript continue à exécuter les instructions entre les accolades. Une fois que la condition n'est plus vérifiée, la boucle est interrompue et on continue le script. Prenons un exemple.

```
compt=1;
while (compt<5) {
document.write ("ligne : " + compt + "<BR>");
compt++;
}
document.write("fin de la boucle");
```

Voyons comment fonctionne cet exemple. D'abord la variable qui nous servira de compteur compt est initialisée à 1. La boucle while démarre donc avec la valeur 1 qui est inférieure à 5. La condition est vérifiée. On exécute les instructions des accolades. D'abord, "ligne : 1" est affichée et ensuite le compteur est incrémenté de 1 et prend donc la valeur 2. La condition est encore vérifiée. Les instructions entre les accolades sont exécutées. Et ce jusqu'à l'affichage de la ligne 4. Là, le compteur après l'incrémentation vaut 5. La condition n'étant plus vérifiée, on continue dans le script et c'est alors fin de boucle qui est affiché.

Attention ! Avec ce système de boucle, le risque existe (si la condition est toujours vérifiée), de faire boucler indéfiniment l'instruction. Ce qui à la longue fait misérablement planter le browser !

Ce qui donnerait à l'écran :

ligne : 1

ligne : 2

ligne : 3

ligne : 4

fin de la boucle

### *+11.4 BREAK*

L'instruction break permet d'interrompre prématurément une boucle for ou while.

Pour illustrer ceci, reprenons notre exemple :

```
compt=1;
while (compt<5) {
if (compt == 4)
break;
document.write ("ligne : " + compt + "<BR>");
compt++;
}
document.write("fin de la boucle");
```

Le fonctionnement est semblable à l'exemple précédent sauf lorsque le compteur vaut 4. A ce moment, par le break, on sort de la boucle et "fin de boucle" est affiché.

Ce qui donnerait à l'écran :

ligne : 1

ligne : 2

ligne : 3

fin de la boucle

## +11.5 CONTINUE

L'instruction continue permet de sauter une instruction dans une boucle for ou while et de continuer ensuite le bouclage (sans sortir de celui-ci comme le fait break).

Reprenons notre exemple ;

```
compt=1;
while (compt<5) {
if (compt == 3){
compt++
continue;}
document.write ("ligne : " + compt + "<BR>");
compt++;
}
document.write("fin de la boucle");
```

Ici, la boucle démarre. Lorsque le compteur vaut 3, par l'instruction continue, on saute l'instruction document.write (ligne : 3 n'est pas affichée) et on continue la boucle. Notons qu'on a dû ajouter compt++ avant continue; pour éviter un bouclage infini et un plantage du navigateur (compt restant à 3).

Ce qui fait à l'écran :

ligne : 1

ligne : 2

ligne : 4

fin de la boucle