

Cours de Javascript façon Langage C

Site: Espadon
Cours: JAVASCRIPT
Livre: Cours de Javascript façon Langage C
Imprimé par: Thomas Stegen
Date: mercredi 23 mars 2016, 16:17

Table des matières

1 Introduction

2 Théorie Objet

3 Où écrire

4 Les variables

5 Les conditions

6 Les fonctions

7 Les messages

8 Les Formulaires

9 Les Evènements

10 Les Opérateurs

11 Les Objets

11.1 L'objet Windows

11.2 L'Objet String

11.3 L'Objet Math

11.4 L'Objet Date

11.5 L'Objet Array

11.6 L'Objet Frames

1 Introduction

COURS DE JAVASCRIPT

**Javascript est un langage
de scripts
qui incorporé aux balises
HTML,
permet d'agréments la
présentation
et l'interactivité des pages
Web**

Javascript est 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 à l'époque LiveScript. Adopté à la fin de l'année 1995, par la firme Sun (qui a aussi développé Java), il prit alors son nom actuel 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.

JAVASCRIPT N'EST PAS JAVA!!!

Il importe de savoir que Javascript est **totalelement 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)

- 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.

2 Théorie Objet

LES OBJETS ET LEUR HIERARCHIE

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

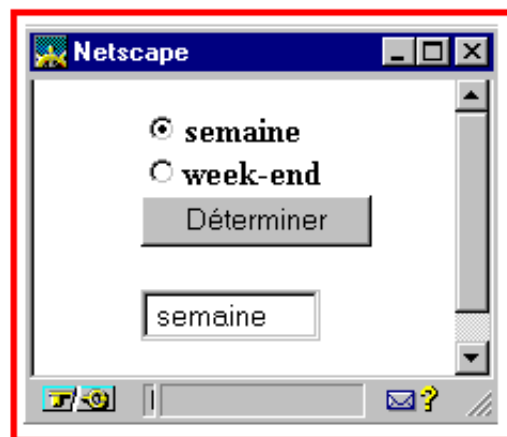
Javascript va diviser cette page en objets et surtout va vous permettre d'accéder à ces objets, d'en retirer des informations et de les manipuler.

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

Vous avez chargé la page suivante :

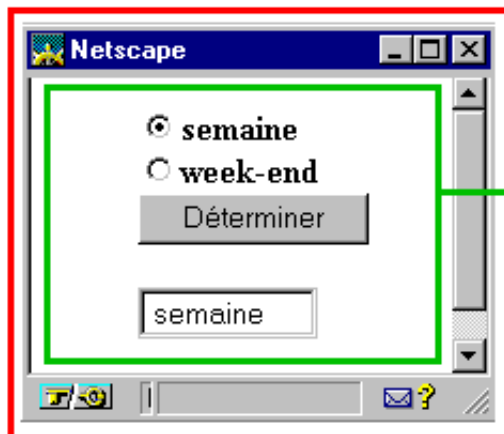


Cette page s'affiche dans une fenêtre. C'est l'**objet fenêtre**.



objet fenêtre

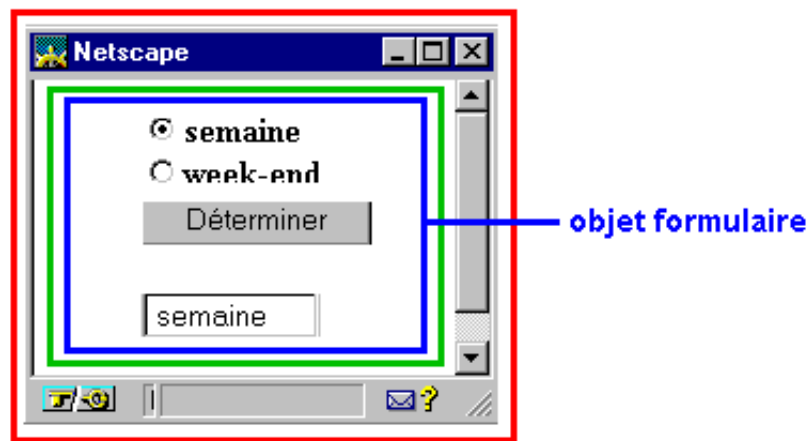
Dans cette fenêtre, il y a un document Html. C'est l'**objet document**.



objet document

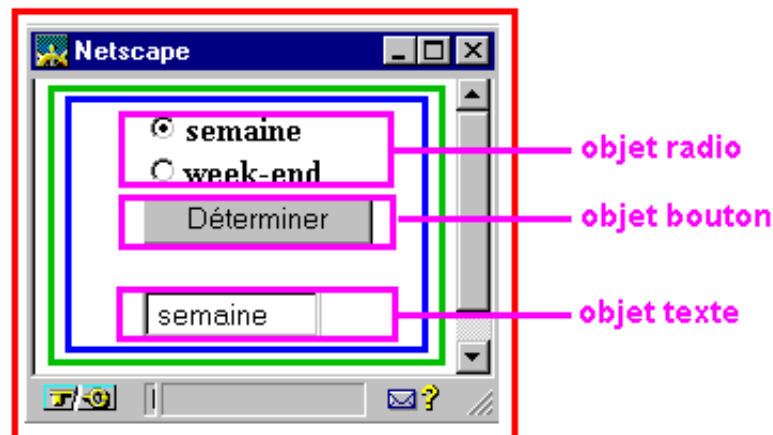
Autrement dit (et c'est là que l'on voit apparaître la notion de la hiérarchie des objets Javascript), l'objet fenêtre contient l'objet document.

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.

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

3 Où écrire

DU JAVASCRIPT D'ACCORD MAIS OU???

Avant d'étudier en détail la syntaxe de Javascript, précisons tout d'abord comment le code Javascript est inséré au code HTML. Pour cela, nous allons nous appuyer sur un premier document HTML enrichi avec des commandes Javascript.

Cet exemple a pour seul but de vous montrer comment le code Javascript s'intègre au sein du code HTML. Ne vous inquiétez donc pas si vous ne comprenez pas vraiment le sens des commandes Javascript.

<pre><HTML> <HEAD> <SCRIPT LANGUAGE="JavaScript1.2"> function fin() { window.clo se() } </SCRIPT> </HEAD> <BODY> <SCRIPT LANGUAGE="JAVASCRIPT1.2"> document.write('Vous pouvez fermer cette fenêtre en cliquant ici') </SCRIPT> ici
 ou en passant la souris au dessus de cela </BODY> </HTML></pre>	<p>=== > Déclaration d'une fonction</p> <p>=== > Execution d'une commande lors du chargement du document.</p> <p>=== > URL avec pseudo-protocole Javascript</p> <p>=== > Nouvel attribut de gestion des événements utilisateurs</p>
---	---

Ce document reflète les trois méthodes possibles pour insérer du JavaScript dans un document HTML:

- La première consiste à utiliser les balises **<SCRIPT>** et **</SCRIPT>**
- la seconde à utiliser le **pseudo-protocole Javascript dans une URL.**
- La dernière à utiliser un des nouveaux **attributs de balise pour la gestion d'événements.**

4 Les variables

UTILISER DES VARIABLES

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!

LA DECLARATION DES VARIABLES

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.

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 false 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.

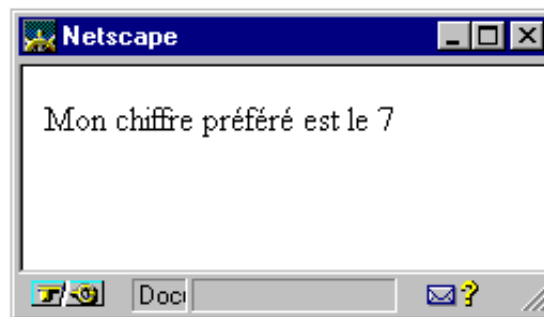
Exercice

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 :



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**

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.

5 Les conditions

LES INSTRUCTIONS CONDITIONNELLES

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.

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'incrémentement *i++* (*i* vaut alors 1) et les instructions s'exécutent.

A la fin de l'exécution des instructions, on revient au compteur. La variable *i* (qui vaut 1) 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.

L'EXPRESSION 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 !

L'EXPRESSION 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
```

L'EXPRESSION 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 ajouté 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
```

6 Les fonctions

LES FONCTIONS

DEFINITIONS

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.

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.

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) que votre fonction soit bien définie avant d'être appelée.

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>.

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>
```

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 boîte 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");
```

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")
```

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))**.

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.

7 Les messages

LES BOITES DE DIALOGUE OU DE MESSAGES

Javascript met à votre disposition 3 boîtes de message

- **alert()**
- **prompt()**
- **confirm()**

Ce sont toutes trois des méthodes de l'objet **window**.

LA METHODE ALERT()

La méthode **alert()** doit, à ce stade de votre étude, vous être familière car nous l'avons déjà souvent utilisée.

La méthode **alert()** affiche une boîte de dialogue dans laquelle est reproduite la valeur (variable et/ou chaîne de caractères) de l'argument qui lui a été fourni. Cette boîte bloque le programme en cours tant que l'utilisateur n'aura pas cliqué sur "OK".

Alert() sera aussi très utile pour vous aider à débbuger les scripts.

Sa syntaxe est :

```
alert(variable);  
alert("chaîne de caractères");  
alert(variable + "chaîne de caractères");
```



(Cliquez sur l'image pour la démo)

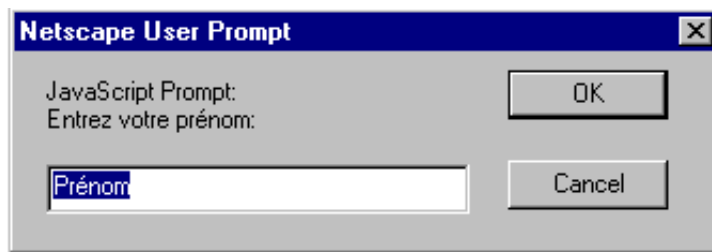
Si vous souhaitez écrire sur plusieurs lignes, il faudra utiliser le signe `\n`.

LA METHODE PROMPT()

Dans le même style que la méthode **alert()**, Javascript vous propose une autre boîte de dialogue, dans le cas présent appelée boîte d'invite, qui est composée d'un champ comportant une entrée à compléter par l'utilisateur. Cette entrée possède aussi une valeur par défaut.

La syntaxe est :

```
prompt("texte de la boîte d'invite", "valeur par défaut");
```



(Cliquez sur l'image pour la démonstration)

Cet exemple s'écrit

```
prompt("Entrez votre prénom:", "Prénom");
```

En cliquant sur OK, la méthode renvoie la valeur tapée par l'utilisateur ou la réponse proposée par défaut. Si l'utilisateur clique sur Annuler ou Cancel, la valeur null est alors renvoyée.

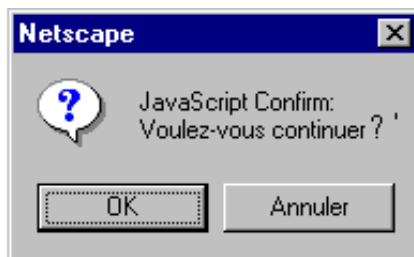
Prompt() est parfois utilisé pour saisir des données fournies par l'utilisateur. Selon certaines sources, le texte ne doit cependant pas dépasser 45 caractères sous Netscape et 38 sous Explorer 3.0.

LA METHODE CONFIRM()

Cette méthode affiche 2 boutons "OK" et "Annuler". En cliquant sur OK, **confirm()** renvoie la valeur true et bien entendu false si on a cliqué sur Annuler. Ce qui peut permettre, par exemple, de choisir une option dans un programme.

La syntaxe de l'exemple est :

```
confirm("Voulez-vous continuer ?")
```



(Cliquez sur l'image pour la démonstration)

UNE MINUTERIE

Javascript met à votre disposition une minuterie (ou plus précisément un compteur à rebours) qui permettra de déclencher une fonction après un laps de temps déterminé.

La syntaxe de mise en route du temporisateur est :

```
nom_du_compteur = setTimeout("fonction_appelée()", temps en milliseconde)
```

Ainsi, **setTimeout("demarrer()", 5000)** va lancer la fonction **demarrer()** après 5 secondes.

Pour arrêter le temporisateur avant l'expiration du délai fixé, il y a :

```
clearTimeout(nom_du_compteur)
```

Prenons un exemple (d'une utilité douteuse mais disons...qu'il est didactique) :

En cliquant sur le bouton "Mise en route", vous activez un compteur qui va afficher une boîte d'alerte après 2 secondes. Si vous avez un peu de réflexe, vous pouvez arrêter prématurément le processus en cliquant sur le bouton "Stop test".

Le script se présente comme suit :

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="Javascript">
<!--
function start()
{
comp=(setTimeout("alert('Vos réflexes sont très lents !')",2000));
}
// -->
</SCRIPT>
</HEAD>
<BODY>
<FORM>
<INPUT TYPE="button" VALUE="Mise en route" onClick="start()">
<INPUT TYPE="button" VALUE=" Stop test " onClick="clearTimeout(comp)">
</FORM>
</BODY>
</HTML>
```

Avec l'instruction **comp=(setTimeout("alert('Vos réflexes sont très lents !')",2000))**, on initialise un compteur, appelé comp par la méthode **setTimeout()**. Ce compteur va déclencher la fonction **start()** après un délai de 2000 millisecondes soit 2 secondes.

Avec **clearTimeout(comp)**, on va arrêter avant terme le compteur dont le nom est **comp**.

L'EMPLOI DE THIS

Pour désigner l'objet en cours, Javascript met à votre disposition le mot-clé **this**. Cette écriture raccourcie est souvent utilisée (sans risque de confusion) en remplacement du chemin complet de l'objet dans un formulaire. Un exemple vous éclairera mieux qu'un long discours.

Soit un script avec un formulaire :

```
<FORM NAME="form3">
<INPUT TYPE="radio" NAME="choix" VALUE="1">Choix numéro 1<BR>
<INPUT TYPE="radio" NAME="choix" VALUE="2">Choix numéro 2<BR>
<INPUT TYPE="radio" NAME="choix" VALUE="3">Choix numéro 3<BR>
<INPUT TYPE="button"NAME="but" VALUE="Quel et votre choix ?" onClick="choixprop(form
3)">
</FORM>
```

Au lieu d'employer **choixprop(form3)**, on aurait pu utiliser **choixprop(this.form)** et éviter ainsi toute confusion avec les autres noms de formulaires. Dans cet exemple, **this.form** désigne le formulaire **form3** complet. Par contre, **choixprop(this)** n'aurait désigné que l'élément de type bouton du formulaire **form3**.

Pour être complet, **this** est utilisé aussi pour créer une ou plusieurs propriétés d'un objet. Ainsi, pour créer un objet livre avec les propriétés auteur, éditeur et prix cette opération peut être effectuée à l'aide de la fonction :

```
function livre(auteur, editeur, prix)
{
  this.auteur = auteur;
  this.editeur = editeur;
  this. prix = prix;
}
```

8 Les Formulaires

LES FORMULAIRES

GENERALITES

Avec Javascript, les formulaires Html prennent une toute autre dimension. N'oublions pas qu'en Javascript, on peut accéder à chaque élément d'un formulaire pour, par exemple, y aller lire ou écrire une valeur, noter un choix auquel on pourra associer un gestionnaire d'événement... Tous ces éléments renforceront grandement les capacités interactives de vos pages.

Mettons au point le vocabulaire que nous utiliserons. Un formulaire est l'élément Html déclaré par les balises **<FORM>****</FORM>**. Un formulaire contient un ou plusieurs éléments que nous appellerons des contrôles (widgets). Ces contrôles sont notés par exemple par la balise **<INPUT TYPE= ...>**.

DECLARATION D'UN FORMULAIRE

La déclaration d'un formulaire se fait par les balises **<FORM>** et **</FORM>**. Il faut noter qu'en Javascript, l'attribut **NAME="nom_du_formulaire"** a toute son importance pour désigner le chemin complet des éléments. En outre, les attributs **ACTION** et **METHOD** sont facultatifs pour autant que vous ne faites pas appel au serveur.

Une erreur classique en Javascript est, emporté par son élan, d'oublier de déclarer la fin du formulaire **</FORM>** après avoir incorporé un contrôle.

LE CONTROLE LIGNE DE TEXTE

La zone de texte est l'élément d'entrée/sortie par excellence de Javascript. La syntaxe Html est

```
<INPUT TYPE="text" NAME="nom" SIZE=x MAXLENGTH=y>
```

pour un champ de saisie d'une seule ligne, de longueur x et de longueur maximale de y.
L'objet text possède trois propriétés :

Propriété	Description
name	indique le nom du contrôle par lequel on pourra accéder.
defaultvalue	indique la valeur par défaut qui sera affichée dans la zone de texte.
value	indique la valeur en cours de la zone de texte. Soit celle tapée par l'utilisateur ou si celui-ci n'a rien tapé, la valeur par défaut.

LIRE UNE VALEUR DANS UNE ZONE DE TEXTE

Voici une zone de texte. Entrez une valeur et appuyer sur le bouton pour contrôler celle-ci.

Le script complet est celui-ci :


```

<HTML>
<HEAD>
<SCRIPT LANGUAGE="javascript">
  function controle(formulaire)
  {
    var test = formulaire.input.value;
    // ou sans passer d'arguments à la fonction
    // var test = document.form1.input.value;
    alert("Vous avez tapé : " + test);
  }
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="form1">
  <INPUT TYPE="text" NAME="input" VALUE=""><BR>
  <INPUT TYPE="button" NAME="bouton" VALUE="Contrôler" onClick="controle(form1)">
</FORM>
</BODY>
</HTML>

```

Lorsqu'on clique le bouton "contrôler", Javascript appelle la fonction **controle()** à laquelle on passe le formulaire dont le nom est **form1** comme argument.

Cette fonction **controle()** définie dans les balises **<HEAD>** prend sous la variable test, la valeur de la zone de texte. Pour accéder à cette valeur, on note le chemin complet de celle-ci (voir le chapitre "Un peu de théorie objet"). Soit dans le document présent, il y a l'objet formulaire appelé **form1** qui contient le contrôle de texte nommé **input** et qui a comme propriété l'élément de valeur **value**. Ce qui donne **document.form1.input.value**.

ECRIRE UNE VALEUR DANS UNE ZONE DE TEXTE

Entrez une valeur quelconque dans la zone de texte d'entrée. Appuyer sur le bouton pour afficher cette valeur dans la zone de texte de sortie.

Zone de texte d'entrée

Zone de texte de sortie

Voici le code :

```

<HTML>
<HEAD>
<SCRIPT LANGUAGE="javascript">
function afficher(formulaire)
{
var testin =formulaire.input.value;
formulaire.output.value=testin;
// ou sans passer d'arguments à la fonction
// var testin =document. form2.input.value;
// document.form2.output.value=testin;
}
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="form2">
<INPUT TYPE="text" NAME="input" VALUE="">
Zone de texte d'entrée <BR>
<INPUT TYPE="button" NAME="bouton" VALUE="Afficher" onClick="afficher(form2)"><BR>
<INPUT TYPE="text" NAME="output" VALUE="">
Zone de texte de sortie
</FORM>
</BODY>
</HTML>

```

Lorsqu'on clique le bouton "Afficher", Javascript appelle la fonction **afficher()** à laquelle on passe le formulaire dont le nom est cette fois **form2** comme argument.

Cette fonction **afficher()** définie dans les balises **<HEAD>** prend sous la variable **testin**, la valeur de la zone de texte d'entrée. A l'instruction suivante, on dit à Javascript que la valeur de la zone de texte **output** comprise dans le formulaire nommé **form2** est celle de la variable **testin**. A nouveau, on a utilisé le chemin complet pour arriver à la propriété valeur de l'objet souhaité soit en Javascript **document.form2.output.value**.

LES BOUTONS RADIO

Les boutons radio sont utilisés pour noter un choix, et seulement un seul, parmi un ensemble de propositions.

Propriété	Description
name	indique le nom du contrôle. Tous les boutons portent le même nom.
index	l'index ou le rang du bouton radio en commençant par 0.
checked	indique l'état en cours de l'élément radio
defaultchecked	indique l'état du bouton sélectionné par défaut.
value	indique la valeur de l'élément radio.

Prenons un exemple :

```

<HTML>
<HEAD>
<SCRIPT language="javascript">
function choixprop(formulaire)
{
if (formulaire.choix[0].checked) { alert("Vous avez choisi la proposition " + formulair
e.choix[0].value) };
if (formulaire.choix[1].checked) { alert("Vous avez choisi la proposition " + formulair
e.choix[1].value) };
if (formulaire.choix[2].checked) { alert("Vous avez choisi la proposition " + formulair
e.choix[2].value) };
// ou sans passer d'arguments à la fonction
// if (document.form3.choix[0].checked) { alert("Vous avez choisi la proposition " + do
cument.form3.choix[0].value) };
// if (document.form3.choix[1].checked) { alert("Vous avez choisi la proposition " + do
cument.form3.choix[1].value) };
// if (document.form3.choix[2].checked) { alert("Vous avez choisi la proposition " + do
cument.form3.choix[2].value) };
}
</SCRIPT>
</HEAD>
<BODY>
Entrez votre choix :
<FORM NAME="form3">
<INPUT TYPE="radio" NAME="choix" VALUE="1">Choix numéro 1<BR>
<INPUT TYPE="radio" NAME="choix" VALUE="2">Choix numéro 2<BR>
<INPUT TYPE="radio" NAME="choix" VALUE="3">Choix numéro 3<BR>
<INPUT TYPE="button"NAME="but" VALUE="Quel et votre choix ?" onClick="choixprop(form
3)">
</FORM>
</BODY>
</HTML>

```

PS: Ce programme a été écrit avec un souci didactique. On pourrait l'écrire avec des codes plus compacts.

Entrez votre choix :

☐ Choix numéro 1

☐ Choix numéro 2

☐ Choix numéro 3

Dans le formulaire nommé **form3**, on déclare trois boutons **radio**. Notez que l'on utilise le même nom pour les trois boutons. Vient ensuite un bouton qui déclenche la fonction **choixprop()**.

Cette fonction teste quel bouton radio est coché. On accède aux boutons sous forme d'indice par rapport au nom des boutons radio soit **choix[0]**, **choix[1]**, **choix[2]**. On teste la propriété **checked** du bouton par **if(form3.choix[x].checked)**. Dans l'affirmative, une boîte d'alerte s'affiche. Ce message reprend la valeur attachée à chaque bouton par le chemin **form3.choix[x].value**.

LES BOUTONS A COCHER (CHECKBOX)

Les boutons case à cocher sont utilisés pour noter un ou plusieurs choix (pour rappel avec les boutons radio un seul choix) parmi un ensemble de propositions. A part cela, sa syntaxe et son usage est tout à fait semblable aux boutons radio sauf en ce qui concerne l'attribut name.

Propriété	Description
name	indique le nom du contrôle. Toutes les cases à cocher portent un nom différent.
checked	indique l'état en cours de l'élément case à cocher.
defaultchecked	indique l'état du bouton sélectionné par défaut.
value	indique la valeur de l'élément case à cocher.

Entrez votre choix :

Il faut sélectionner les numéros 1,2 et 4 pour avoir la bonne réponse.

<input type="checkbox"/> Choix numéro 1
<input type="checkbox"/> Choix numéro 2
<input type="checkbox"/> Choix numéro 3
<input type="checkbox"/> Choix numéro 4

```

<HTML>
<HEAD>
<script language="javascript">
function reponse(formulaire)
{
if ( (formulaire.check1.checked) == true
    && (formulaire.check2.checked) == true
    && (formulaire.check3.checked) == false
    && (formulaire.check4.checked) == true)
// ou sans passer d'arguments à la fonction
//if ( (document.form4.check1.checked) == true
//    && (document.form4.check2.checked) == true
//    && (document.form4.check3.checked) == false
//    && (document.form4.check4.checked) == true)
{
alert("C'est la bonne réponse! ")
}
else
{
alert("Désolé, continuez à chercher.")}
}
</SCRIPT>
</HEAD>
<BODY>
Entrez votre choix :
<FORM NAME="form4">
<INPUT TYPE="CHECKBOX" NAME="check1" VALUE="1">Choix numéro 1<BR>
<INPUT TYPE="CHECKBOX" NAME="check2" VALUE="2">Choix numéro 2<BR>
<INPUT TYPE="CHECKBOX" NAME="check3" VALUE="3">Choix numéro 3<BR>
<INPUT TYPE="CHECKBOX" NAME="check4" VALUE="4">Choix numéro 4<BR>
<INPUT TYPE="button"NAME="but" VALUE="Corriger"
    onClick="reponse(form4)">
</FORM>
</BODY>
</HTML>

```

Dans le formulaire nommé **form4**, on déclare quatre cases à cocher. Notez que l'on utilise un nom différent pour les quatre boutons. Vient ensuite un bouton qui déclenche la fonction **reponse()**. Cette fonction teste quelles cases à cocher sont sélectionnées. Pour avoir la bonne réponse, il faut que les cases 1, 2 et 4 soient cochées. On accède aux cases en utilisant chaque fois leur nom. On teste la propriété **checked** du bouton par (**form4.nom_de_la_case.checked**). Dans l'affirmative (&& pour et logique), une boîte d'alerte s'affiche pour la bonne réponse. Dans la négative, une autre boîte d'alerte vous invite à recommencer.

LISTE DE SÉLECTION

Le contrôle liste de sélection vous permet de proposer diverses options sous la forme d'une liste déroulante dans laquelle l'utilisateur peut cliquer pour faire son choix. Ce choix reste alors affiché. La boîte de la liste est créée par la balise **<SELECT>** et les éléments de la liste par un ou plusieurs tags **<OPTION>**. La balise **</SELECT>** termine la liste.

Propriété	Description
name	indique le nom de la liste déroulante.
length	indique le nombre d'éléments de la liste. S'il est indiqué dans le tag <SELECT> , tous les éléments de la liste seront affichés. Si vous ne l'indiquez pas un seul apparaîtra dans la boîte de la liste déroulante.
selectedIndex	indique le rang à partir de 0 de l'élément de la liste qui a été sélectionné par l'utilisateur.
defaultselected	indique l'élément de la liste sélectionné par défaut. C'est lui qui apparaît alors dans la petite boîte.

Un petit exemple comme d'habitude :

Entrez votre choix :

```

<HTML>
<HEAD>
<script language="javascript">
function liste(formulaire)
{
alert("L'\élément " + (formulaire.list.selectedIndex + 1));
// ou sans passer d'arguments à la fonction
// alert("L'\élément " + (document.form5.list.selectedIndex + 1));
}
</SCRIPT>
</HEAD>
<BODY>
Entrez votre choix :
<FORM NAME="form5">
<SELECT NAME="list">
<OPTION VALUE="1">Elément 1
<OPTION VALUE="2">Elément 2
<OPTION VALUE="3">Elément 3
</SELECT>
<INPUT TYPE="button"NAME="b" VALUE="Quel est l'élément retenu?" onClick="liste(form5)">
</FORM>
</BODY>
</HTML>

```

Dans le formulaire nommé **form5**, on déclare une liste de sélection par la balise **<SELECT>** **</SELECT>**. Entre ses deux balises, on déclare les différents éléments de la liste par autant de tags **<OPTION>**. Vient ensuite un bouton qui déclenche la fonction **liste()**.

Cette fonction teste quelle option a été sélectionnée. Le chemin complet de l'élément sélectionné est **form5.nomdelaliste.selectedIndex**. Comme l'index commence à 0, il ne faut pas oublier d'ajouter 1 pour retrouver le juste rang de l'élément.

LE CONTROLE TEXTAREA

L'objet textarea est une zone de texte de plusieurs lignes.

La syntaxe Html est :

```
<FORM>
<TEXTAREA NAME="nom" ROWS=x COLS=y>
texte par défaut
</TEXTAREA>
</FORM>
```

où **ROWS=x** représente le nombre de lignes et **COLS=y** représente le nombre de colonnes.

texte par défaut

L'objet textarea possède plusieurs propriétés :

Propriété	Description
name	indique le nom du contrôle par lequel on pourra accéder.
defaultValue	indique la valeur par défaut qui sera affichée dans la zone de texte.
value	indique la valeur en cours de la zone de texte. Soit celle tapée par l'utilisateur ou si celui-ci n'a rien tapé, la valeur par défaut.

A ces propriétés, il faut ajouter **onFocus()**, **onBlur()**, **onSelect()** et **onChange()**.

En Javascript, on utilisera `\r\n` pour passer à la ligne.

Comme par exemple dans l'expression **document.Form.Text.value = 'Check\r\nthis\r\nout'**.

LE CONTROLE RESET

Le contrôle **Reset** permet d'annuler les modifications apportées aux contrôles d'un formulaire et de restaurer les valeurs par défaut.

la syntaxe Html est :

```
<INPUT TYPE="reset" NAME="nom" VALUE="texte">
```

où **VALUE** donne le texte du bouton.

texte par défaut...

```
<FORM NAME="form6">
<TEXTAREA Name="texte" ROWS=3 COLS=30>Texte par défaut
</TEXTAREA>
<INPUT TYPE="reset" NAME="res" VALUE="Reset de la zone de texte ci-dessus">
</FORM>
```

Une seule méthode est associée au contrôle **Reset**, c'est la méthode **onClick()**. Ce qui peut servir, par exemple, pour faire afficher une autre valeur que celle par défaut.

LE CONTROLE SUBMIT

Le contrôle a la tâche spécifique de transmettre toutes les informations contenues dans le formulaire à l'URL désignée dans l'attribut **ACTION** du tag **<FORM>**.

la syntaxe Html est :

```
<INPUT TYPE="submit" NAME="nom" VALUE="texte">
```

où **VALUE** donne le texte du bouton.

Une seule méthode est associée au contrôle **Submit**, c'est la méthode **onClick()**.

LE CONTROLE HIDDEN (CACHE)

Le contrôle **Hidden** permet d'entrer dans le script des éléments (généralement des données) qui n'apparaîtront pas à l'écran. Ces éléments sont donc cachés. D'où son nom.

la syntaxe Html est :

```
<INPUT TYPE="hidden" NAME="nom" VALUE= "les données cachées">
```

ENVOIE D'UN FORMULAIRE PAR EMAIL

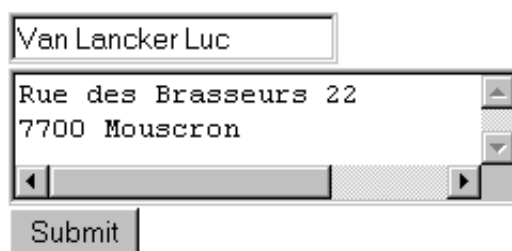
Uniquement Netscape !!!

A force de jouer avec des formulaires, il peut vous prendre l'envie de garder cette source d'information. Mais comment faire? Javascript, et a fortiori le Html, ne permet pas d'écrire dans un fichier. Ensuite, le contrôle Submit est surtout destiné à des CGI ce qui entraîne (encore) un codage spécial à maîtriser. D'autant que pour nous simples et présumés incompetents internautes, la plupart des providers ne permettra pas d'héberger une CGI faite par un amateur pour des raisons (tout à fait compréhensibles) de sécurité. Il ne reste plus que l'unique solution de l'envoi d'un formulaire via le courrier électronique.

La syntaxe est :

```
<FORM METHOD="post" ACTION="mailto:votre_adresse_Email">  
<INPUT TYPE="text" NAME="nom">  
<TEXTAREA NAME="adresse" ROWS=2 COLS=35>  
</TEXTAREA>  
<INPUT TYPE="submit" VALUE="Submit">  
</FORM>
```

Ce qui donne :



Van Lancker Luc

Rue des Brasseurs 22
7700 Mouscron

Submit

Vous recevrez dans notre boîte de réception, un truc bizarre du genre :

nom=Van+Lancker+Luc&adresse=Rue+des+Brasseurs+2217OD%OA7700+Mouscron.

où on retrouve les champs nom= et adresse=, où les champs sont séparés par le signe &, où les espaces sont remplacés par le signe + et 17%OD%OA correspond à un passage à la ligne.

Attention ! Ceci ne marche que sous Netscape et pas sous Microsoft Explorer 3.0 ...
Avec Explorer, le mailto ouvre le programme de Mail mais n'envoie rien du tout.

9 Les Evènements

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.

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.	Click
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. 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

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 boîte d'alerte avec le message indiqué.

onClick

Événement 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.

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>
```

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.

A titre d'illustration, passez avec le pointeur de la souris sur le mot voyelles (sans cliquer sur le lien).

Voyelles

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

Notons que 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.

Nous parlons plus longuement de **onmouseover** et de **onmouseout** à la fin de ce chapitre.

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".

Si vous cliquez dans la zone de texte, vous effectuez un focus

Le code est :

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

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.

Si après avoir cliqué et/ou écrit dans la zone de texte, vous cliquez ailleurs dans le document, vous produisez un événement Blur.

Le code est :

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

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.

Modifiez ce texte...

Le code est :

```
<FORM>
<INPUT TYPE=text onChange="alert('Vous avez fait un Change')" size=25 Value="Modifiez c
e texte...">
</FORM>
```

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.

Selectionnez ce texte...

Le code est :

```
<FORM>
<INPUT TYPE=text onSelect="alert('Vous avez fait un Change')" size=25 Value="Selectionn
ez ce texte...">
</FORM>
```

GESTIONNAIRES D'EVENEMENTS DISPONIBLES EN JAVASCRIPT

Il nous semble utile de présenter la liste des objets auxquels correspondent des gestionnaires d'événement bien déterminé.

Objets	Gestionnaires d'événement disponibles
Fenêtre	onLoad, onUnload
Lien hypertexte	onClick, onMouseOver, onMouseOut
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 sélection	onBlur, onChange, onFocus
Bouton Submit	onClick
Bouton Reset	onClick

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.

MESSAGE IMPORTANT...

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>
```

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 l'exemple suivant

Message Important

dont le code est :

```
<A HREF="" onMouseOver="alert('Coucou')" onMouseOut="alert('Au revoir')">message impor
tant</A>
```

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

PROBLEME, ET SI ON CLIQUE QUAND MEME!!!

Vous avez codé votre instruction **onmouseover** avec le lien fictif ``, 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'intégralité 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

MESSAGE IMPORTANT...

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

CHANGEMENT D'IMAGES

Avec le gestionnaire d'événement **onmouseover**, on peut prévoir qu'après le survol d'une 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 jeune()
{
document.images["photo_ben"].src="ben.jpg"
}
function vieux()
{
document.images["photo_ben"].src="ben_vieux.jpg"
}
</SCRIPT>
</HEAD>
<BODY>
<A HREF="#" onmouseover="vieux();" onmouseout="jeune();">
<IMG SRC="ben.jpg" name="photo_ben" 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.

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 ?

10 Les Opérateurs

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 cours) les différents opérateurs mis à notre disposition par Javascript.

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

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.

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

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

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++	incréméntation(x++ est le même que x=x+1)	y = x++	3 puis plus 1	4
x--	décréméntation (x-- est le même que x=x-1)	y= x--	3 puis moins 1	2

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.

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

11 Les Objets

Les différents objets

11.1 L'objet Windows

PROPRIETES ET METHODES DE L'OBJET WINDOW

Certaines propriétés et méthodes de l'objet window ne vous sont pas inconnues :

- celles des boîtes de dialogue. Soit **alert()**, **confirm()**, et **prompt()**,
- et celles du ou des minuteries. Soit **setTimeout()** et **clearTimeout()**.

Une autre série, ayant trait aux frames, fait l'objet d'un chapitre consacré à ce sujet :

- ce sont **frames[]**, **length**, **parent**, **opener** et **top**.

Une série a trait à la barre d'état qui est abordée ci-après :

- ce sont **status** et **defaultStatus**.

Une série pour l'ouverture et la fermeture d'une fenêtre :

- ce sont **open()** et **close()**.

Et enfin **self** qui renvoie à la fenêtre en cours.

UTILISATION DE LA BARRE D'ETAT

Avec Javascript, la barre d'état (petite bande située au bas de la fenêtre du browser et qui vous informe sur l'état des transferts et des connections) peut être utilisée pour afficher des messages de votre cru. Comme je suis myope comme une taupe, ce n'est pas ma partie préférée du Javascript mais c'est une opinion des plus subjective.

Les propriétés mises en oeuvre sont :

Propriété	Description
status	valeur du texte affiché dans la barre d'état de la fenêtre.
defaultStatus	valeur par défaut qui s'affiche dans la barre d'état.

Généralement, cet événement est mis en oeuvre par un **onmouseover()** sur un lien hypertexte.

En voici un exemple :

```
<HTML>
<BODY>
<A HREF="#" onmouseover="self.status='Votre texte'; return true;"> A voir ici </A>
</BODY>
</HTML>
```

A voir ici
(simple survol)

Il est indispensable d'ajouter **return true;**

OUVERTURE ET FERMETURE DE FENETRE (THEORIE)

Les méthodes mises en oeuvre sont :



Méthodes	Description
open()	ouvre une nouvelle fenêtre.
close()	ferme la fenêtre en cours.

La syntaxe est :

[window.]open("URL","nom_de_la_fenêtre","caractéristiques_de_la_fenêtre")

où **URL** est l'URL de la page que l'on désire afficher dans la nouvelle fenêtre. Si on ne désire pas afficher un fichier htm existant, on mettra simplement "".

où **caractéristiques_de_la_fenêtre** est une liste de certaines ou de toutes les caractéristiques de fenêtre suivantes que l'on note à la suite, séparées par des virgules et sans espaces ni passage à la ligne.

Caractéristique	Description
toolbar=yes ou no	Affichage de la barre d'outils
location=yes ou non	Affichage de champ d'adresse (ou de localisation)
directories=yes ou no	Affichage des boutons d'accès rapide
status=yes ou no	Affichage de la barre d'état
menubar=yes ou no	Affichage de la barre de menus
scrollbars=yes ou no	Affichage des barres de défilement.
(scrollbars=no	fonctionne mal sous Explorer 3.0)
resizable=yes ou no	Dimensions de la fenêtre modifiables
width=x en pixels	Largeur de la fenêtre en pixels
height=y en pixels	Hauteur de la fenêtre en pixels

On peut aussi utiliser 1 ou 0 au lieu de yes ou no.

Remarques :

Cette nouvelle fenêtre va s'afficher un peu n'importe où sur votre écran. Vous ne pouvez pas décider de l'endroit exact où elle peut apparaître. Cependant sous Netscape 4 c.-à -d. sous Javascript 1.2 , ce petit "plus" est possible.

Sous Microsoft Explorer 3, l'apparition de la nouvelle fenêtre se fait après une grimace du browser (il ouvre temporairement une nouvelle fenêtre du browser).

L'usage des nouvelles fenêtres est assez sympathique en Javascript pour afficher des informations complémentaires sans surcharger la page (ou fenêtre) de départ. Cependant, aussi longtemps que l'utilisateur ne ferme pas ces nouvelles fenêtres, celles-ci restent ouvertes (lapalissade). Le pire est lorsqu'on les minimise. Pour peu qu'on utilise souvent cette technique, le navigateur se retrouve avec plusieurs dizaines de fenêtres ouvertes ce qui fait désordre, ralentit le système et peut finir par le planter.

Veillez donc à toujours faire fermer ces nouvelles fenêtres.

EXEMPLE 1: OUVERTURE PAR UN BOUTON (CODE DANS LE **onClick()**)

Nous allons ouvrir une petite fenêtre qui affiche le fichier test.htm avec un bouton dans la page.

Fichier test.htm :

```
<HTML>
<BODY>
<H1>Ceci est un test<HI>
<FORM>
<INPUT TYPE="button" value= " Continuer " onClick="self.close()">
</FORM>
</BODY>
</HTML>
```

où **self.close()** fermera la fenêtre courante, c.-à -d. la nouvelle fenêtre.

Dans la page de départ :

```
<FORM>
<INPUT TYPE ="button" value="Ouvrir une nouvelle fenêtre"
onClick="open('test.htm', 'new', 'width=300,height=150,toolbar=no,location=no,
directories=no,status=no,menubar=no,scrollbars=no,resizable=no')">
(sans espaces ni passage à la ligne)
</FORM>
```

OUVERTURE PAR UN BOUTON (AVEC APPEL DE FONCTION)

Dans la page de départ :

```
<SCRIPT LANGUAGE="javascript">
<!--
function new_window() {
xyz="open('test.htm', 'new', 'width=300,height=150,toolbar=no,location=no,
directories=no,status=no,menubar=no,scrollbars=no,resizable=no')"
// sans espaces ni passage à la ligne
}
// -->
</SCRIPT>
<FORM>
<INPUT TYPE ="button" value="Ouvrir une nouvelle fenêtre" onClick="new_window()">
</FORM>
```

FERMETURE AUTOMATIQUE APRES X SECONDES

Avec ce script, sans intervention de l'utilisateur, la nouvelle fenêtre se ferme de façon automatique après 4 secondes. En cliquant sur le bouton, l'utilisateur interrompt prématurément le compteur et ferme la fenêtre. Avec ce système, on est certain que la nouvelle fenêtre sera fermée.

La page test.htm devient testc.htm

```
<HTML>
<BODY onLoad='compt=setTimeout("self.close();",4000) '>
<H1>Ceci est un test</H1>
<FORM>
<INPUT TYPE="button" value=" Continuer " onClick="clearTimeout(compt);self.close();">
</FORM>
</BODY>
</HTML>
```

Dans la page de départ :

```
<FORM>
<INPUT TYPE ="button" value="Ouvrir une nouvelle fenêtre"
onClick="open('testc.htm', 'new', 'width=300,height=150,toolbar=no,location=no,
directories=no,status=no,menubar=no,scrollbars=no,resizable=no')">
(sans espaces ni passage à la ligne)
</FORM>
```

OUVERTURE EN CLIQUANT SUR UN LIEN OU UNE IMAGE

On ajoute simplement le "onClick=open..." à la balise <A> du lien ou de l'image.

Dans la page de départ, on a :

```
<A HREF="#" onClick="open('testc.htm', 'width=300,height=150,toolbar=no,location=no,directories=no,status=no,menubar=no,scrollbar=no,resizable=no')">lien de test</A> (sans espaces ni passage à la ligne)
```

lien de test
(cliquez)

La fermeture automatique est particulièrement utile ici car si l'utilisateur clique quand même, la nouvelle fenêtre disparaît derrière la fenêtre de départ et risque donc de ne pas être fermée.<:P>

OUVERTURE PAR SURVOL DU LIEN (javascript 1.0)

On utilise ici **onmouseover**. Pour rappel, Javascript 1.0 donc compatible Explorer 3.

Dans la page de départ, on a :

```
<A HREF="#" onmouseover="open('testc.htm', 'width=300,height=150,toolbar=no,location=no,directories=no,status=no,menubar=no,scrollbar=no,resizable=no')">lien de test</A> (sans espaces ni passage à la ligne)
```

lien de test
(simple survol)

OUVERTURE PAR SURVOL DU LIEN ET FERMETURE EN QUITTANT LE LIEN (javascript 1.1)

On utilise ici **onmouseover** et **onmouseout**. Pour rappel, **onmouseover** est du Javascript 1.1 et ne fonctionne donc pas sous Explorer 3.0.

Dans la page de départ, on a :

```
<A HREF="#" onmouseover="open('test.htm', 'width=300,height=150,toolbar=no,location=no,directories=no,status=no,menubar=no,scrollbars=no,resizable=no')" onmouseout="self.close()">lien de test</A>
```

(sans espaces ni passage à la ligne)

ECRIRE DANS LA NOUVELLE FENETRE

On passe par l'ouverture d'une nouvelle fenêtre par l'appel d'une fonction.

Dans la page de départ :

```
<SCRIPT LANGUAGE="Javascript">
<!--
function opnw(){ msg=window.open("", "", "width=300,height=50,toolbar=no,location=no,directories=no,
status=no,menubar=no,scrollbars=no,resizable=no");
msg.document.write('<HTML> <BODY>' +
'<CENTER><H1>Ceci est un test<H1></CENTER>' +
'</BODY></HTML>')
// sans espaces ni passage à la ligne
}
// -->
</SCRIPT>
```

Selon l'auteur, avec cette forme d'écriture, il n'y a pas moyen de fermer la fenêtre par un bouton (Help...)

11.2 L'Objet String

GENERALITES

Revenons à l'objet String pour nous intéresser à la manipulation des caractères si utile pour l'aspect programmation de Javascript.

On signale dans la littérature une limitation de la longueur des strings à 50/80 caractères. Cette limitation du compilateur Javascript peut toujours être contournée par l'emploi de signes + et la concaténation.

Instruction	Description
length	C'est un entier qui indique la longueur de la chaîne de caractères.
charAt()	Méthode qui permet d'accéder à un caractère isolé d'une chaîne.
indexOf()	Méthode qui renvoie la position d'une chaîne partielle à partir d'une position déterminée. (en commençant au début de la chaîne principale soit en position 0).
lastIndexOf()	Méthode qui renvoie la position d'une chaîne partielle à partir d'une position déterminée. (en commençant à la fin soit en position length moins 1).
substring(x,y)	Méthode qui renvoie un string partiel situé entre la position x et la position y-1.
toLowerCase()	Transforme toutes les lettres en minuscules.
toUpperCase()	Transforme toutes les lettres en Majuscules.

La propriété length

La propriété **length** retourne un entier qui indique le nombre d'éléments dans une chaîne de caractères. Si la chaîne est vide (" "), le nombre est zéro.

La syntaxe est simple :

```
x=variable.length;  
x=("chaîne de caractères").length;
```

La propriété **length** ne sert pas que pour les Strings, mais aussi pour connaître la longueur ou le nombre d'éléments :

- **de formulaires** . Combien a-t-il de formulaires différents ?
- **de boutons radio**. Combien a-t-il de boutons radio dans un groupe ?
- **de cases à cocher**. Combien a-t-il de cases à cocher dans un groupe ?
- **d'options**. Combien a-t-il d'options dans un Select ?
- **de frames**. Combien a-t-il de frames "enfants" ?
- **d'ancres, de liens, etc.**

La méthode CharAt()

Il faut d'abord bien noter que les caractères sont comptés de gauche à droite et que la position du premier caractère est 0. La position du dernier caractère est donc la longueur (length) de la chaîne de caractère moins 1;

chaîne : Javascript (longueur = 10)

|||||||

position : 0123456789 (longueur - 1)

Si la position que vous indiquer est inférieure à zéro ou plus grande que la longueur moins 1, Javascript retourne une chaîne vide.

La syntaxe de **charAt()** est :

```
chaîne_réponse = chaîne_départ.charAt(x);
```

où x est un entier compris entre 0 et la longueur de la chaîne à analyser moins 1.

Notez les exemples suivants :

```
var str="Javascript";           La réponse est "J".
var chr=str.charAt(0);
var chr="Javascript".charAt(0);
ou var chr=charAt(str,0);
ou var chr=charAt("Javascript",0);

var str="Javascript";           La réponse est "t".
var chr=str.charAt(9);
var chr=charAt(str,9);

var str="Javascript";           La réponse est "".
var chr=charAt(str,13);         soit vide.
```

La méthode indexOf()

Cette méthode renvoie la position, soit x, d'un string partiel (lettre unique, groupe de lettres ou mot) dans une chaîne de caractères en commençant à la position indiquée par y. Cela vous permet, par exemple, de voir si une lettre, un groupe de lettres ou un mot existe dans une phrase.

```
variable="chaîne_de_caractères";
var="string_partiel";
x=variable.indexOf(var,y);
```

où y est la position à partir de laquelle la recherche (de gauche vers la droite) doit commencer. Cela peut être tout entier compris entre 0 et la longueur - 1 de la chaîne de caractères à analyser.

Si y n'est pas spécifié, la recherche commencera par défaut à la position 0.

Si le string partiel n'est pas trouvé dans la chaîne de caractères à analyser, la valeur retournée sera égale à -1.

Quelques exemples :

```
variable="Javascript"           x vaut 4  
var="script"  
x=variable.indexOf(var,0);  
variable="VanlanckerLuc&ccim.be"   x vaut -1  
var="@"  
x=variable.indexOf(var);
```

La méthode `lastIndexOf()`

Méthode fort semblable à `indexOf()` sauf que la recherche va cette fois de droite à gauche (en commençant donc par la fin).

La syntaxe est en tous points identique sauf que y signale une position située vers la fin de la chaîne de caractères.

```
x=variable.lastIndexOf(var,y);
```

Les exemples suivants montrent la différence entre `indexOf()` et `lastIndexOf()` :

```
variable="Javascript"  
var="a"  
x=variable.indexOf(var,0); ici x vaut 1 soit la position du premier a.  
x=variable.lastIndexOf(var,9); ici x vaut 3 soit la position du second a.
```

Notons que même lorsqu'on commence à lire de la fin de la chaîne, la position retournée est comptée depuis le début de la chaîne avec le comptage commençant à zéro.

La méthode `substring()`

La méthode `substring()` est du même genre que `indexOf()`, `lastIndexOf()` et `charAt()` que nous venons d'étudier. Elle sera particulièrement utile, par exemple, pour prendre différentes données dans une longue chaîne de caractères.

```
variable = "chaîne de caractères"  
resultat=variable.substring(x,y)
```

où `resultat` est un sous ensemble de la chaîne de caractère (ou de la variable).

Les `x` et `y` sont des entiers compris entre 0 et la longueur moins 1 de la chaîne de caractères.

Si `x` est inférieur à `y`, la valeur retournée commence à la position `x` et se termine à la position `Y-1`.

Si `x` est supérieur à `y`, la valeur retournée commence à la position `y` et se termine à la position `X-1`.

En fait, ceci donne le même résultat et il est équivalent d'écrire par exemple `substring(3,6)` ou `substring(6,3)`.

Si `x` est égal à `y`, `substring()` retourne une chaîne vide (logique, non?)

Vous souhaitez sà rement quelques exemples :

```
Javascript
```

```
|||||||
```

```
0123456789
```

```
str="Javascript";  
str1=str.substring(0,4);  
str2="Javascript".substring(0,4);  
str3=str.substring(6,9);
```

Les résultats sont :

str1="Java"; soit les positions 0,1,2 et 3.

str2="Java"; soit les positions 0,1,2 et 3.

str3="rip"; soit les positions 6,7 et 8

La méthode toLowerCase()

Cette méthode affiche toutes les majuscules d'une chaîne de caractères variable2 en minuscules.

```
variable2="chaîne de caractères";  
variable1=variable2.toLowerCase();
```

Exemple :

```
str="JavaScript";  
str1=str.toLowerCase();  
str2="JavaScript".toLowerCase();
```

Le résultat sera :

str1="javascript";

str2="javascript";

La méthode toUpperCase()

Cette méthode affiche toutes les minuscules d'une chaîne de caractères variable2 en majuscules.

```
variable2="chaîne de caractères";  
variable1=variable2.toUpperCase();
```

Exemple :

```
str="JavaScript";  
str3=str.toUpperCase();  
str4="JavaScript".toUpperCase();
```

Le résultat sera :

str3="JAVASCRIPT";

str4="JAVASCRIPT";

Utilité de toLowerCase() et de toUppercase()

L'utilité de ces 2 méthodes ne vous saute peut être pas aux yeux. Et pourtant, il faut se rappeler que Javascript est case sensitive. Ainsi une recherche sur Euro ne donnera pas le même résultat que sur euro ou EUro.

Ainsi, pour les bases de données, il est prudent de tout convertir en minuscules (ou en majuscules). A fortiori, pour certaines informations des utilisateurs introduites par le biais d'un formulaire.

11.3 L'Objet Math

Pour manipuler les nombres, voici l'objet Math.

abs()

```
x=Math.abs(y);
```

La méthode **abs()** renvoie la valeur absolue (valeur positive) de y. Il supprime en quelque sorte le signe négatif d'un nombre.

```
y = 4;  
x = math.abs(y);  
x = Math.abs(4);  
x = math.abs(-4);  
    ont comme résultat  
x = 4
```

ceil()

```
x=Math.ceil(y);
```

La méthode **ceil()** renvoie l'entier supérieur ou égal à y.

Attention ! Cette fonction n'arrondit pas le nombre.

Comme montré dans l'exemple, si y = 1.01, la valeur de x sera mise à 2.

```
y=1.01;  
x=Math.ceil(y);
```

a comme résultat 2.

floor()

```
x=Math.floor(y);
```

La méthode **floor()** renvoie l'entier inférieur ou égal à y.

Attention ! Cette fonction n'arrondit pas le nombre.

Comme montré dans l'exemple, si y = 1.99, la valeur de x sera mise à 1.

```
y=1.999;  
x=Math.floor(y);  
a comme résultat 1.
```

round()

```
x=Math.round(y);
```

La méthode **round()** arrondit le nombre à l'entier le plus proche.

```
y=20.355;  
x=Math.round(y);
```

a comme résultat

```
x=20;
```

Attention ! Certains calculs réclament une plus grande précision. Pour avoir deux décimales après la virgule, on utilisera la formule :

```
x=(Math.round(y*100))/100;
```

et dans ce cas

```
x=20.36;
```

max()

```
x=Math.max(y,z);
```

La méthode **max(y,z)** renvoie le plus grand des 2 nombres y et z.

```
y=20; z=10;  
x=Math.max(y,z);
```

a comme résultat

```
x=20;
```

min()

```
x=Math.min(y,z);
```

La méthode **min(y,z)** renvoie le plus petit des 2 nombres y et z.

```
y=20; z=10;  
x=Math.min(y,z);
```

a comme résultat

```
x=10;
```

pow()

```
x=Math.pow(y,z);
```

La méthode **pow()** calcule la valeur d'un nombre y à la puissance z.

```
y=2; z=8  
x=Math.pow(y,z);
```

a comme résultat
 2^8 soit 256

random()

```
x=Math.random();
```

La méthode **random()** renvoie la valeur d'un nombre aléatoire choisi entre 0 et 1.
Attention ! Cela ne fonctionne que sous Unix.
A la place, on peut utiliser la fonction suivante qui renvoie un nombre "aléatoire" entre 0 et 9.

```
function srand(){ t=new Date(); r=t.getTime(); p="a"+r; p=p.charAt((p.length-4)); x=p; }
```

sqrt()

```
x=Math.sqrt(y);
```

La méthode **sqrt()** renvoie la racine carrée de y.

```
y=25;  
x=Math.sqrt(y);
```

a comme résultat

```
x=5;
```

parseFloat()

```
x=parseFloat("variable");
```

Cette fonction convertit une chaîne contenant un nombre en une valeur à virgule flottante. Ou si vous préférez, elle retourne les chiffres derrière la virgule d'un nombre.

Attention ! Le résultat risque d'être surprenant si Javascript rencontre autre chose dans la chaîne que des nombres, les signes + et -, le point décimal ou un exposant. S'il trouve un caractère "étranger", La fonction ne prendra en compte que les caractères avant le caractère "étranger".

Si le premier caractère n'est pas un caractère admis, x sera égal à 0 sous Windows et à "NaN" sur les autres systèmes.

```
str='-.12345';  
str1='$5.50';  
x=parseFloat(str);
```

aura comme résultat


```
x= -.12345;  
x=0 ou "NaN";
```

parseInt()

```
x=parseInt(variable);
```

Retourne la partie entière d'un nombre avec une virgule.

```
str='1.2345';  
x=parseInt(str);  
x=1;
```

eval()

```
x=eval(variable);
```

Cette fonction évalue une chaîne de caractère sous forme de valeur numérique. On peut stocker dans la chaîne des opérations numériques, des opérations de comparaison, des instructions et même des fonctions.

```
str='5 + 10';  
x=eval(str);
```

a comme résultat

```
x=15;
```

On dit dans la littérature que cette fonction **eval()** est une opération majeure de Javascript. Que son emploi n'est pas toujours conseillé pour les débutants. Pire, que cette fonction **eval()** n'est pas supportée par toutes les plate-formes avec toutes les versions des browsers. Prudence donc, vérifiez toujours le résultat renvoyé par **eval()**. Mieux, trouvez un autre tutorial qui sera plus précis sur le sujet !

Les fonctions trigonométriques

Voici (sans commentaires) ces différentes fonctions :

```
x=Math.PI;  
x=Math.sin(y);  
x=Math.asin(y);  
x=Math.cos(y);  
x=Math.acos(y);  
x=Math.tan(y);  
x=Math.atan(y);
```

Les fonctions logarithmiques

Pour les initiés :

```
x=Math.exp(y);  
x=Math.log(y);  
x=Math.LN2;  
x=Math.LN10;  
x=Math.E;  
x=Math.LOG2E;  
x=Math.LOG10E;
```

11.4 L'Objet Date

new Date();

Cette méthode renvoie toutes les informations "date et heure" de l'ordinateur de l'utilisateur.

```
variable=new Date();
```

Ces informations sont enregistrées par Javascript sous le format :

"Fri Dec 17 09:23:30 1998"

Attention ! La date et l'heure dans Javascript commence au 1e janvier 1970. Toute référence à une date antérieure donnera un résultat aléatoire.

La méthode **new date ()** sans arguments renvoie la date et l'heure courante.

Pour introduire une date et une heure déterminée, cela se fera sous la forme suivante :

```
variable=new Date("Jan 1, 2000 00:00:00");
```

Toutes les méthodes suivantes vous faciliterons la tâche pour accéder à un point précis de cette variable (en fait un string) et pour modifier si besoin en est le format d'affichage.

getFullYear()

```
variable_date=new Date();  
an=variable_date.getFullYear();
```

Retourne les deux derniers chiffres de l'année dans **variable_date**. Soit ici 03.
Comme vous n'avez que deux chiffres, il faudra mettre 20 en préfixe soit

```
an="20"+variable_date.getFullYear();
```

getMonth()

```
variable_date=new Date();  
mois=variable_date.getMonth();
```

Retourne le mois dans **variable_date** sous forme d'un entier compris entre 0 et 11 (0 pour janvier, 1 pour février, 2 pour mars, etc.). Soit ici 11 (le mois moins 1).

getDate();

```
variable_date=new Date();  
jourm=variable_date.getDate();
```

Retourne le jour du mois dans **variable_date** sous forme d'un entier compris entre 1 et 31.
Eh oui, ici on commence à 1 au lieu de 0 (pourquoi???).
A ne pas confondre avec **getDay()** qui retourne le jour de la semaine.

getDay();

```
variable_date=new Date();  
jours=variable_date.getDay();
```

Retourne le jour de la semaine dans **variable_date** sous forme d'un entier compris entre 0 et 6 (0 pour dimanche, 1 pour lundi, 2 pour mardi, etc.).

getHours();

```
variable_date=new Date();  
hrs=variable_date.getHours();
```

Retourne l'heure dans **variable_date** sous forme d'un entier compris entre 0 et 23.

getMinutes();

```
variable_date=new Date();  
min=variable_date.getMinutes();
```

Retourne les minutes dans **variable_date** sous forme d'un entier compris entre 0 et 59.

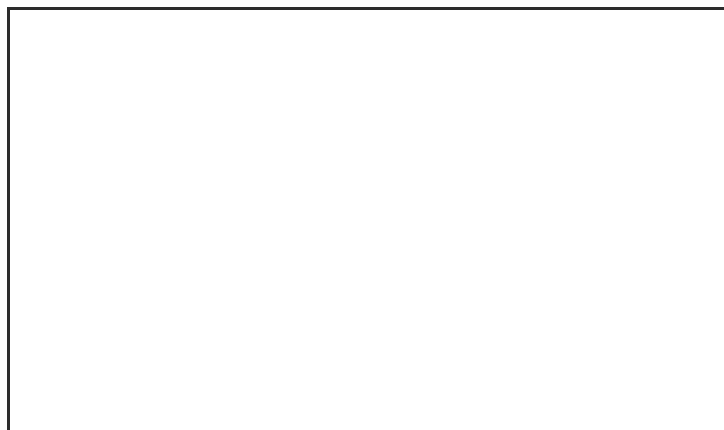
getSeconds();

```
variable_date=new Date();  
sec=variable_date.getSeconds();
```

Retourne les secondes dans **variable_date** sous forme d'un entier compris entre 0 et 59.

Exemple 1 : Un script qui donne simplement l'heure.

25/6/20015 10:00:45
30/9/2009 14:58:07



```

<HTML>
<HEAD>
<SCRIPT LANGUAGE="Javascript">
<!--
function getDt(){
    dt=new Date();
    cal=""+ dt.getDate()+"/"+(dt.getMonth()+1)
    + "/200" +(dt1.getYear()-100);
    hrs=dt.getHours();
    min=dt.getMinutes();
    sec=dt.getSeconds();
    tm=" "+((hrs<10)?"0":"" ) +hrs)+":";
    tm+=((min<10)?"0":"" )+min+":";
    tm+=((sec<10)?"0":"" )+sec+" ";
    document.write(cal+tm);
}
// -->
</SCRIPT>
</HEAD>
<BODY >
<SCRIPT LANGUAGE="Javascript">
<!--
    getDt();
    // -->
</SCRIPT>
</BODY>
</HTML>

```

Exemple 2 : Un script avec une trotteuse.

14:58:12

Vous souhaitez peut-être que votre affichage de l'heure change toutes les secondes. Rappelerez-vous la minuterie **setTimeout**. Il suffit d'ajouter au script un **setTimeout** qui affiche l'heure toutes les secondes. La fonction qui affiche l'heure étant **getDt()**, l'instruction à ajouter est donc **setTimeout("getDt()",1000);** Et le tour est joué.

```

<HTML>
<HEAD>
<SCRIPT LANGUAGE="Javascript">
<!--
    function getDt(){
    dt=new Date();
    hrs=dt.getHours();
    min=dt.getMinutes();
    sec=dt.getSeconds();
    tm=" "+((hrs<10)?"0":"" )+hrs+":";
    tm+=((min<10)?"0":"" )+min+":";
    tm+=((sec<10)?"0":"" )+sec+" ";
    document.horloge.display.value=tm;
    setTimeout("getDt()",1000);
    }
    // -->
</SCRIPT>
</HEAD>
<BODY onLoad="getDt()">
<FORM name="horloge">
<INPUT TYPE="text" NAME="display" SIZE=15 VALUE ="">
</FORM>
</BODY>
</HTML>

```

D'autres propriétés

```
variable.getTime();
```

Retourne l'heure courante dans **variable_date** sous forme d'un entier représentant le nombre de millisecondes écoulées depuis le 1 janvier 1970 00:00:00.

```
variable.getTimezoneOffset();
```

Retourne la différence entre l'heure locale et l'heure GMT (Greenwich, UK Mean Time) sous forme d'un entier représentant le nombre de minutes (et pas en heures).

```
variable.setYear(x);
```

Assigne une année à l'actuelle valeur de **variable_date** sous forme d'un entier supérieur à 1900.
Exemple : variable_date.setYear(98) pour 1998.

```
variable.setMonth(x);
```

Assigne un mois à l'actuelle valeur de **variable_date** sous forme d'un entier compris entre 0 et 11.
Exemple : variable_date.setMonth(1);

```
variable.setDate(x);
```

Assigne un jour du mois à l'actuelle valeur de **variable_date** sous forme d'un entier compris entre 1 et 31.

Exemple : `variable_date.setDate(1);`

```
variable.setHours(x);
```

Assigne une heure à l'actuelle valeur de **variable_date** sous forme d'un entier compris entre 1 et 23.

Exemple : `variable_date.setHours(1);`

```
variable.setMinutes(x);
```

Assigne les minutes à l'actuelle valeur de **variable_date** sous forme d'un entier compris entre 1 et 59.

Exemple : `variable_date.setMinutes(1);`

```
variable.setSeconds(x);
```

Assigne les secondes à l'actuelle valeur de **variable_date** sous forme d'un entier compris entre 1 et 59.

Exemple : `variable_date.setSeconds(0);`

```
variable.setTime(x);
```

Assigne la date souhaitée dans **variable_date** sous forme d'un entier représentant le nombre de millisecondes écoulées depuis le 1 janvier 1970 00:00:00. et la date souhaitée.

```
variable.toGMTString();
```

Retourne la valeur de l'actuelle valeur de **variable_date** en heure GMT (Greenwich Mean Time).

```
variable.toLocaleString()
```

Retourne la valeur de l'actuelle valeur de **variable_date** sous forme de String.

Il me semble qu'on aura plus vite fait avec les **getHours()**, **getMinutes()** and **getSeconds()**.

11.5 L'Objet Array

Généralités

L'objet **Array** (ou tableaux) est une liste d'éléments indexés dans lesquels on pourra ranger (écrire) des données ou aller reprendre ces données (lire).

Attention ! L'objet Array est du Javascript 1.1

Tableau à une dimension

Pour faire un tableau, il faut procéder en deux étapes :

- d'abord construire la structure du tableau. A ce stade, les éléments du tableau sont vides.
- ensuite affecter des valeurs dans les cases ainsi définies.

On commence par définir le tableau :

```
nom_du_tableau = new Array (x);
```

où x est le nombre d'éléments du tableau.

On notera que, "le nombre d'éléments est limité à 255. Cette restriction ne figure pas dans la documentation de Netscape mais elle a été constatée expérimentalement." Source : Javascript de Michel Dreyfus Collection Mégapoché.

Ensuite, on va alimenter la structure ainsi définie :

```
nom_du_tableau[i] = "élément";
```

où i est un nombre compris entre 0 et x moins 1.

Exemple :

un carnet d'adresse avec 3 personnes:

construction du tableau :

```
carnet = new Array(3);
```

ajout des données :

```
carnet[0]="Dupont";  
carnet[1]="Médard";  
carnet[2]="Van Lancker";
```

pour accéder un élément, on emploie :

```
document.write(carnet[2]);
```

On notera ici que les données sont bien visibles au lecteur un peu initié (view source).

Remarques :

- Quand on en aura l'occasion, il sera pratique de charger le tableau avec une boucle. Supposons que l'on ait à charger 50 images. Soit on le fait manuellement, il faut charger **0.gif, 1.gif, 2.gif...** Soit on utilise une boucle du style :

```
function gifs() {
  gif = new Array(50);
  for (var=i;i<50;i++)
  {gif[i] =i+".gif";}
}
```

- Javascript étant un langage peu typé, il n'est pas nécessaire de déclarer le nombre d'élément du tableau (soit x). Javascript prendra comme nombre d'éléments, le nombre i le plus élevé lors de "l'alimentation" de la structure (en fait i + 1). Ainsi la formulation suivante serait aussi correcte pour un tableau à 3 dimensions.

```
carnet = new Array();
carnet[2]="Van Lancker";
```

Propriétés et méthodes

Elément	Description
length	Retourne le nombre d'éléments du tableau.
join()	Regroupe tous les éléments du tableau dans une seule chaîne. Les différents éléments sont séparés par une caractère séparateur spécifié en argument. Par défaut, ce séparateur est une virgule.
reverse()	Inverse l'ordre des éléments (ne les trie pas).
sort()	Retourne les éléments par ordre alphabétique (à condition qu'ils soient de même nature)

Dans le cadre de notre exemple,

```
document.write(carnet.join());
```

donne comme résultat : Médard,Dupont, Van Lancker.

```
document.write(carnet.join("-"));
```

a comme résultat : Médard-Dupont-Van Lancker.

```
document.write(carnet.reverse().join("-"));
```

donne : Van Lancker-Dupont-Médard

Tableau à deux dimensions

On peut créer des tableaux à deux dimensions (et plus encore) par un petit artifice de programmation. On déclare d'abord un tableau à 1 dimension de façon classique :

```
nom_du_tableau = new Array (x);
```

Ensuite, on déclare chaque élément du tableau comme un tableau à 1 dimension :

```
nom_du_tableau[i] = new Array(y);
```

Pour un tableau de 3 sur 3 :

Tarif	T. Small	T. M�dium	T. Large
Chemises	1200	1250	1300
Polos	800	850	900
T-shirts	500	520	540

```
nom = new Array(3);
nom[0] = new Array(3);
nom[1] = new Array(3);
nom[2] = new Array(3);
nom[0][0]="1200"; nom[0][1]="1250"; nom[0][2]="1300";
nom[1][0]="800"; nom[1][1]="850"; nom[1][2]="900";
nom[2][0]="500"; nom[2][1]="520"; nom[2][2]="540";
```

Pour exploiter ces donn es, voici une illustration de ce qui est possible :

Choix de l'article :

Choix de la taille :

Le formulaire s' crit :

```
<FORM name="form" >
<SELECT NAME="liste">
<OPTION>Chemises
<OPTION>Polos
<OPTION>T-shirts
</SELECT>
<SELECT NAME="taille">
<OPTION>T. Small
<OPTION>T. M dium
<OPTION>T. Large
</SELECT>
<INPUT TYPE="button" VALUE="Donner le prix" onClick="affi(this.form)">
<INPUT TYPE="TEXT" NAME="txt">
</FORM>
```

o  la fonction **affi()** se formule comme suit :

```
function affi() {  
    i = document.form.liste.selectedIndex;  
    j= document.form.taille.selectedIndex  
    document.form.txt.value=nom[i][j];  
}
```

11.6 L'Objet Frames

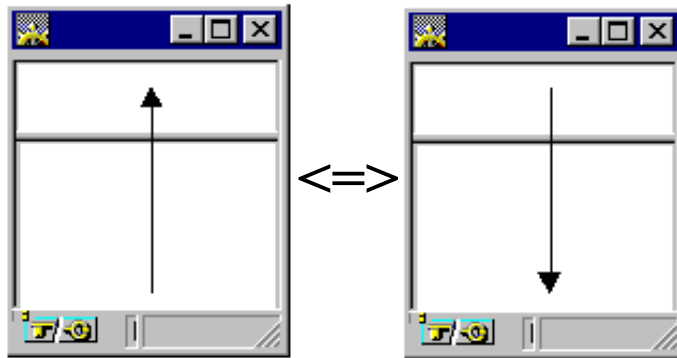
GENERALITES

Tout d'abord, présentons les frames. Utiliser des frames vous permet de diviser la fenêtre affichant votre page HTML en plusieurs cadres (parties distinctes) indépendants les uns des autres. Vous pouvez alors charger des pages différentes dans chaque cadre. Pour la syntaxe Html des frames, vous pouvez vous référer au cours d'Initiation à HTML.

En Javascript, nous nous intéresserons à la capacité de ces cadres à interagir. C'est à dire à la capacité d'échanger des informations entre les frames.

En effet, la philosophie du Html veut que chaque page composant un site soit une entité indépendante. Comment pourrait-on faire alors pour garder des informations tout au long du site ou tout simplement passer des informations d'une page à une autre page. Tout simplement (sic), en utilisant des frames.

Le schéma suivant éclairera le propos :



PROPRIETES

Propriétés	Description
length	Retourne le nombre de frames subordonnés dans un cadre "créateur".
parent	Synonyme pour le frame "créateur".

ECHANGE D'INFORMATIONS ENTRE LES FRAMES

Par l'exemple suivant, nous allons transférer une donnée introduite par l'utilisateur dans une frame, dans une autre frame.

La page "créatrice" des frames

```
<HTML>
<HEAD>
</HEAD>
<FRAMESET ROWS="30%,70%">
<FRAME SRC="enfant1.htm" name="enfant1">
<FRAME SRC="enfant2.htm" name="enfant2">
</FRAMESET>
</HTML>
```

La page "créatrice" contient deux frames subordonnées "**enfant1**" et "**enfant2**".

Le fichier **enfant1.htm**

```
<HTML>
<BODY>
<FORM name="form1">
<INPUT TYPE="TEXT" NAME="en" value=" ">
</FORM>
</BODY>
</HTML>
```

Le fichier **enfant2.htm**

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="Javascript">
<!--
    function affi(form) {
        parent.enfant1.document.form1.en.value=document.form2.out.value
    }
    // -->
</SCRIPT>
</HEAD>
<BODY>
Entrez une valeur et cliquez sur "Envoyer".
<FORM NAME="form2" >
<INPUT TYPE="TEXT" NAME="out">
<INPUT TYPE="button" VALUE="Envoyer" onClick="affi(this.form)">
</FORM>
</BODY>
</HTML>
```

La donnée entrée par l'utilisateur se trouve par le chemin **document.form2.out.value**. Nous allons transférer cette donnée dans la zone de texte de l'autre frame. Pour cela, il faut en spécifier le chemin complet. D'abord la zone de texte se trouve dans la frame subordonnée appelée **enfant1**. Donc le chemin commence par **parent.enfant1**. Dans cette frame se trouve un document qui contient un formulaire (**form1**) qui contient une zone de texte (**en**) qui a comme propriété **value**. Ce qui fait comme chemin **document.form1.en.value**. L'expression complète est bien :

parent.enfant1.document.form1.en.value=document.form2.out.value

EN RESUME: COMMENT FAIRE REFERENCE A UN OBJET DANS LE CAS DE FRAMES

Reprenons l'exemple des deux frames verticales du début du chapitre: Considérons que nous sommes dans le frame enfant1

Deux cas peuvent arriver:

Je veux faire référence à un objet dans le même frame

- **document.form1.en.value**
par défaut, fait référence au document actif soit le frame dans lequel on est.
- **this.document.form1.en.value**
this. fait référence au document actif soit le frame dans lequel on est.
- **window.top.enfant1.document.form1.en.value**
Cette fois ci, on écrit le chemin complet depuis le frame créateur le plus haut dans la hierarchie des frames.

Je veux faire référence à un objet dans un frame différent

- **parent.enfant2.document.form2.out.value**
parents. fait référence au frame créateur du frame dans lequel on est, puis on va dans le frame enfant2 qui est situé en dessous lui dans la hiérarchie des frames
- **window.top.enfant2.document.form2.out.value**
Cette fois ci, on écrit le chemin complet depuis le frame créateur le plus haut dans la hierarchie des frames.