
JAVASCRIPT

# SOMMAIRE

# INTRODUCTION

JavaScript : c'est un langage de Programmation Interprété, structuré et "Orienté" Objet. Il est inspiré du langage « C ». Pour apprendre ce langage vous devez connaître le langage HTML car les scripts JavaScript s'incorporent dans des pages HTML. Ce langage peut être intéressant car au lieu de faire travailler vos serveurs qui hébergent vos pages web pour manipuler des données (vérification de formulaires par exemple), certaines actions peuvent être directement traitées depuis l'ordinateur des internautes pour ainsi éviter de trop ralentir la vitesse de vos serveurs. Si vous connaissez d'autres langages comme le PHP ou le C++, ce langage ne devrait pas être difficile à apprendre car ils sont assez proches.
Le Javascript a été développé par la société Netscape Corporation, qui l'a introduit, pour la première fois dans son navigator 2.0 en 1996. Depuis, le langage a été enrichi, et afin de rendre ce langage accessible à d'autres navigateurs, un procédé de normalisation a été entrepris dès 1996. La norme porte le nom ECMA-262 [ECM99], et on se réfère parfois à javascript sous le nom d'ECMAscript. *ECMA (European Computer Manufacture Association).*
JavaScript connaît plusieurs versions (JavaScript 1.0. JavaScript 1.1 JavaScript 1.2 JavaScript 1.3)
L'intérêt d'un langage comme javascript est de pouvoir contrôler dynamiquement le comportement d'une page Web : on peut par exemple vérifier que le code postal saisi dans la page est correct, faire afficher des menus spéciaux quand la souris approche d'une zone donnée, afficher des bandeaux publicitaires animés, orienter automatiquement le visiteur sur une autre page, …
En résumé, javascript est un langage de scripts qui, incorporé aux balises Html, permet d'améliorer la présentation et l'interactivité des pages Web.

# I. PRESENTATION GENERALE DU LANGAGEJAVASCRIPT

## 1.1. Javascript est utilisé coté client

La charge d'exécuter le code javascript incombe au client, c'est-à-dire à un navigateur Web. Le serveur envoie le code HTML et javascript au client qui l'interprète dès qu'il est chargé. Ce type de fonctionnement s'oppose aux langages orientés serveurs, comme PHP ou ASP.

## 1.2. Javascript est interprété

On distingue habituellement deux modes d'exécution des programmes. Dans le premier mode, dit compilé, le programme source est traduit dans un autre format, directement compréhensible par la machine. La phase de traduction s'appelle la compilation et donne un fichier exécutable (ou binaire).Les avantages de ce mode sont la rapidité d'exécution (les instructions sont directement exécutées par la machine) et la validité syntaxique du programme source (pour que le compilateur puisse traduire le code, il faut que les instructions aient une syntaxique correcte).
Le deuxième mode est dit interprété : un programme « interprète » lit le programme source, et essaye d'exécuter les instructions les unes après les autres. A la différence du mode compilé, il n'y a pas de fichier exécutable. L'avantage de ce mode est que toute personne ayant le programme interprète sur son ordinateur, peut exécuter le programme. L'inconvénient est la lenteur due à l'exécution et à l'interprétation du programme.
Javascript est un langage interprété. Il faut donc un programme interprète appelée moteur de script pour exécuter des programmes écrits dans ce langage. Les navigateurs les plus répandus à l'heure actuelle sur le marché, Netscape et Internet Explorer incorporent de tels interprètes.

*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) |

## 1.3. Le javascript minimum

### 1.3.1. La balise < SCRIPT >

De ce qui précède, vous savez déjà que le script vient s'ajouter à votre page Web. 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.
La balise à utiliser est:

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

Ou

```
<script language="javascript" type="text/javascript">
```

(Pour le XHTML)

Ou encore

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

L'attribut du langage (javascript dans la première ligne) est supporté à partir du Netscape 2.0 et Internet explorer 3.0. Pour des versions plus récentes du navigateur, l'attribut peut aussi être javascript 1.1 (Netscape 3.0) ou javascript 1.2 (Netscape 4.0).
De même, il faudra informer le browser de la fin du script avec la balise :

```
</script>.
```

### 1.3.2. Les commentaires javascript

Il vous sera peut-être utile d'inclure des commentaires personnels dans vos codes javascript. Javascript utilise les conventions utilisées en C et C++ soit :

```
//Commentaire sur une ligne
```

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.

```
/* commentaire sur plusieurs lignes */
```

*Ne confonder pas les commentaires javascript et les commentaires Html (pour rappel* **<!-- … -->**
*Il faut veiller à ne pas imbriquer des commentaires, au risque de provoquer une erreur lors de l'exécution du code!*

### 1.3.3. 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'affiche 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 code javascript ressemblera à ceci :

```
<SCRIPT LANGUAGE = "javascript">
<!-- Masquer le script pour les anciens browsers
…
Instructions javascript
…
//cesser de masquer le script -->
</SCRIPT>
```

## 1.4. Implémentation du code javascript ?

Le code javascript s'insère le plus souvent dans la page HTML elle-même ; Mais on peut aussi le stocker dans un fichier externe et faire appel à ce fichier.
Il existe quatre manières d'insérer du code javascript dans une page HTML :

### 1.4.1. Insertion pour exécution directe

On l'appelle exécution directe, car le code s'exécute automatiquement lors du chargement de la page HTML dans le navigateur. Le code javascript est placé dans le corps même de la page HTML, entre les balises <body> ……. et ……. </body>.

```
<html>
<head>
```

```
…
</head>
<body>

<script language=" javascript">
//Place du code javascript
</script>

</ body>
<html>
```

### 1.4.2. Exécution différée

Le code est d'abord lu par le navigateur, stocké en mémoire, pour ne s'exécuter que sur demande expresse. Le code javascript est placé dans l'en-tête de la page HTML, entre les balises <head> ……. Et </head>. Dans ce cas, le code s'exécutera seulement lors d'un évènement généré par intervention de l'utilisateur. Il faut bien sûr appeler le code correspondant à cet évènement dans le corps du document HTML. Le squelette de la page HTML est alors :

```
<html>
<head>
<title>……</title>
<script language=" javascript">
// Place du code javascript
</ script>
</head>
<body>

//Place du code évènement

</ body>
</html>
```

### 1.4.3. Insertion du code javascript à l'intérieur d'une balise HTML

Certaines balises HTML acceptent de réagir à des évènements soit provoqués par l'intervention de l'utilisateur, soit provoqués par une action du navigateur lui-même, comme le chargement de la page HTML (évènement onload) .

Dans ce cas, le code javascript peut être aussi inséré directement au niveau de la balise en question.

Le squelette de la page HTML est alors :

```
<html>
<head>
<title>……</title>
</head>
<body>
<balise évènement="javascript : place du code évènement ;">
</ body>
</html>
```

**Exemple :**

```
<INPUT TYPE = "button" NAME = "evenement" VALUE = "Cliquez ici pour
exécuter le code contenu dans la balise " onClick= "javascript : alert
('bonjour') ;">
```

### 1.4.4. Insertion du code javascript par appel de module externe

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:

```
<SCRIPT LANGUAGE="javascript" SRC="URL/nom_fichier.js"> …</ script>
```

L'intérêt de cette méthode est de simplifier la maintenance des sites faisant appel à des modules javascript communs à plusieurs pages HTML. En effet, au lieu de modifier toutes les pages contenant le code en question, il n'y a qu'à modifier le code du module externe pour que toutes les pages faisant appel à lui bénéficient de la modification sans risque d'erreur. L'inconvénient est que l'appel au code externe génère une requête supplémentaire vers le serveur, et encombre le réseau.

## 1.5. Typologie

Javascript est case sensitive ; c'est-à-dire, qu'il fait la distinction entre les majuscules et les minuscules.

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 doivent être utilisés au sein d'une chaîne ou remplacés par des codes spéciaux.

Les guillemets (") et l'apostrophe ( ' )font partir intégrante du langage javascript et servent à délimiter les chaînes de caractère. Pour les utiliser dans une chaîne de caractère il faut utiliser le caractère d'échappement **** (\" ou\')

Comme tout langage informatique, javascript possède ses mots réservés. Ce sont ceux que l'interpréteur sait reconnaître, et grâce auxquels il fait ce qu'on lui dit de faire.

Chaque instruction javascript se termine par un point–virgule( ;). Mais, en réalité, ce point-virgule est optionnel et n'est obligatoire que lorsque vous écrivez plusieurs instructions sur une même ligne.

Comme dans le cadre du html, la saisie du code javascript peut se faire avec par exemple un simple éditeur de texte comme le bloc notes de Windows.

# II. ENTREE / SORTIE

## 2.1. Afficher du texte en javascript

Pour écrire du texte dans une page web, javascript fournit deux méthodes : la méthode **write()** et la méthode **writeln().**

### 2.1.1. La méthode write ( )

Pour utiliser cette méthode, la syntaxe est : **document.Write ("texte")** ; ou plus simplement **: write ("texte")** ;
On peut aussi écrire une variable, par exemple, la variable résultat : write (resultat) ;
Pour associer du texte (chaînes de caractères) et des variables, on utilise l'opérateur de concaténation des chaines (+) : write ("le résultat est" + resultat) ;
On peut utiliser les balises html pour agrémenter ce texte :
document.write ("<B> le résultat est </B>" + resultat) ; ou
document.write ("<B>" + "le résultat est"+ "</B>" + resultat) ;

**Exemple :**

```
<HTML>
<HEAD>
<TITLE> Exemple </TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE=" javascript" type="text/javascript">
<!--
var filiere ="INFORMATIQUE" ;
var annee = 5 ;
document.write("Je suis étudiant en "+ filiere +" "+ annee ) ;
// -->
</SCRIPT>
</BODY>
</HTML>
```

### 2.1.2. La méthode 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 writeln () il faut l'inclure dans des balises <PRE> </PRE>

**Exemple**

```
<PRE>
<SCRIPT LANGUAGE=" javascript">
< - -
document.writeln ("ligne 1")
document.writeln ("ligne 2")
// - -
</ SCRIPT>
</ PRE>
```

Autrement dit l'emploi de writeln ( ) est anecdotique et on utilise simplement la balise <BR> avec la méthode write ( ).

## 2.2. Les boites de dialogue ou de message

Javascript met à votre disposition 3 boites de messages :
**alert()**
**prompt()**
**confirm()**
Ce sont toutes trois des méthodes de l'objet window (la fenêtre active)

### 2.2.1. La méthode alert ( )

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

**syntaxe:**

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

Si vous souhaitez écrire sur plusieurs lignes, il faut utiliser la séquence d'échappement **\n**.

### 2.2.2. La méthode prompt( )

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

**syntaxe:**

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

En cliquant sur OK, la méthode renvoie la valeur tapée par l'utilisateur ou la réponse proposée par défaut. Si l'utilisateur clique sur Annule 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.*

### 2.2.3. La méthode confirm ( )

Cette méthode affiche 2 boutons OK et Annuler. En cliquant sur OK, elle 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.

**syntaxe :**

```
confirm ("voulez-vous continuer ?") ;
```

**Exemple**

```
<HTML>
<HEAD><TITLE> utilisation des boites de dialogue </TITLE></HEAD>
<BODY>

<SCRIPT LANGUAGE=" javascript" type="text/javascript">
<!--
nom=window.prompt("votre nom SVP ?", "") ;
if(confirm("votre nom est bien " + nom +"?????"))
{
alert("vous avez confirmé la saisie de votre nom");
document.write ("bonjour " + nom +"<BR> Bienvenue au cours de Javascript
! ! !") ;
```

```
}
else
{
alert("vous n'avez pas confirmé la saisie de votre nom");
document.write ("bonjour  inconnu <BR> Présentez-vous ! ! !") ;

}

//-->
</SCRIPT>

</ BODY>
</HTML>
```

# III. LES VARIABLES

## 3.1. Déclaration

Une variable est une case mémoire à laquelle on a donné un nom. Les variables contiennent des données qui, peuvent être modifiées lors de l'exécution d'un programme. 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'exception du blanc). Pour rappel javascript est sensible à la case. Attention donc aux majuscules et minuscules !
L'opération qui permet d'attribuer une valeur à une variable s'appelle affectation.
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-clé **var**.

**Exemple :**

```
var compteur ;
var numero=1 ;
var prenom = "Nabeledi" ;
```

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.

**Exemple :**

```
Numero = 1 ;
Nom='Nabson' ;
Prenom = "bill" ;
```

*Notons qu'on peut initialiser les variables (y inscrire une valeur initiale).*

## 3.2. Les mots défendus

Lorsque vous nommez une variable en javascript, vous devez faire attention de ne pas utilisez les mots réservés suivant comme nom de variable javascript :

| | | |
|---|---|---|
| abstract | Boolean | Break |
| byte | Char | Continue |
| default | Delete | Do |
| double | else | Export |
| false | Final | Float |
| for | Function | Goto |
| if | implements | Import |
| instanceof | In | Int |
| interface | Long | Native |
| new | Null | Package |
| private | Protected | Public |
| return | Short | Static |
| switch | synchronised | This |
| throws | Transient | True |
| typeof | Var | Void |
| while | With | |

## 3.3. Les types de données

Un type de données est un ensemble de valeurs et un ensemble d'opérateurs sur ces valeurs définis pour assurer la cohérence des instructions.

**Javascript utilise trois (3) principaux types simples de données :**

| Type | Description |
|---|---|
| Nombres | Tout nombre entiers avec virgule tel que 22 ou 3.1416 |
| Chaînes de caractères | Toute suite de caractères comprise entre les guillemets ou quotes telle que "suite de caractères" |
| Booléens | Les mots true pour vrai et false pour faux |

**Remarque**

Il existe un type particulier : **null**, qui ne possède qu'une seule valeur : null !
Une variable non encore affectée est indéfinie (ce qui est différent de la valeur null) et sera type **undefined** .

*Notons aussi que contrairement au langage Cou C++, la déclaration du type de données d'une variable n'est pas nécessaire. On n'a donc pas besoin de int, float, char etc.*
*En javascript. Il n'y a pas de distinction entre entiers et réels.*

# IV. LES OPERATEURS ET LES EXPRESSIONS

Une expression est une combinaison de variables et d'opérateurs. On distingue des expressions arithmétiques, logiques et les expressions de chaîne.
Voyons les différents opérateurs mis à notre disposition par javascript pour construire ces expressions.

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

## 4.1. Les opérateurs arithmétiques

| Signe | Nom | Signification | Exemple | résultat |
|---|---|---|---|---|
| + | Plus | Addition | x+3 | 14 |
| - | Moins | Soustraction | x-3 | 8 |
| * | Multiplié | 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=11 | |

## 4.2. Les opérateurs de comparaison

| Signe | Nom | Exemple | Résultat |
|---|---|---|---|
| = = | Egal | 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 les signes = et = = (deux signes=). Le signe = est un opérateur d'affectation tandis que = = est un opérateur de comparaison. Cette confusion est une source classique d'erreur de programmation.*

## 4.3. Les opérateurs associatifs

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

*Dans les exemples suivants x vaut toujours 11 et il 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 |

## 4.4. Les opérateurs logiques

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

| Signe | Nom | Exemple | Signification |
|---|---|---|---|
| && | Et | (condition1) && (condition2) | Condition 1 et condition 2 |
| \|\| | Ou | (Condition1) \|\| (condition2) | Condition 1 ou condition 2 |

## 4.5. Les opérateurs d'incrémentation et de décrémentation

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 suivants 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 |

## 4.6. La priorité des opérateurs javascript

Les opérateurs s'utilisent dans l'ordre suivant de priorité (du degré de priorité le plus faible au 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 |
| = = ! = | Egalité |
| < <= >= > | Comparaison |
| + - | Addition soustraction |
| * / | Multiplier diviser |
| ! - + + - - | Unaire |
| ( ) | Parenthèses |

**Remarque : concaténation de chaînes de caractères**

L'opérateur '+' sert aussi à juxtaposer deux ou plusieurs chaînes caractères pour en former une seule. On parle de concaténation.

**Exemple :**

```
"L'étoile" +" " + "filante" donne          "l'étoile filante".
```

# V. LES STRUCTURES DE CONTROLE

## 5.1. Structures conditionnelles

### 5.1.1. L'instruction if

L'instruction if est la structure de test la plus basique, on la retrouve dans tous les langages (avec une syntaxe différente...). Elle permet d'exécuter une série d'instruction si jamais une condition est réalisée.

**Syntaxe**

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

**Remarques:**

*la condition doit être entre des parenthèses*
*il est possible de définir plusieurs conditions à remplir avec les opérateurs ET et OU (&& et ||)*
*par exemple:*
*if ((condition1)&&(condition2)) teste si les deux conditions sont vraies*
*if ((condition1)||(condition2)) exécutera les instructions si l'une ou l'autre des deux conditions est vraie*
*s'il n'y a qu'une instruction, les accolades ne sont pas nécessaires...*

### 5.1.2. L'instruction if ... else

L'instruction *if* dans sa forme basique ne permet de tester qu'une condition, or la plupart du temps on aimerait pouvoir choisir les instructions à exécuter **en cas de non réalisation de la condition**...

L'expression *if ... else* permet d'exécuter une autre série d'instruction en cas de non réalisation de la condition.

**Syntaxe :**

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

**Une façon plus courte de faire un test**

Il est possible de faire un test avec une structure beaucoup moins lourde grâce à la structure suivante:

```
(condition) ? instruction si vrai : instruction si faux
```

**Remarques:**

*la condition doit être entre des parenthèses*
*Lorsque la condition est vraie, l'instruction de gauche est exécutée*
*Lorsque la condition est fausse, l'instruction de droite est exécutée*

### 5.1.3. Structures conditionnelles imbriquées;

Les instructions conditionnelles peuvent être imbriquées : cela signifie qu'une structure conditionnelle peut elle-même contenir une autre.

**Exemple :**

```
<SCRIPT language = "JavaScript" >
 var age=0;
age=prompt("Donnez votre âge : ","");
 if ( age <= 0  )
 alert("Cela n'a pas de sens !");
 else
    if (age <=13)
    alert("Vous êtes encore bien trop jeune ...")
    else
       if (age <18)
       alert("Désolé, vous êtes encore mineur(e)")
       else
          if (age <25)
          alert("Vous êtes déjà majeur(e) !")
          else alert("Ne vous vieillissez donc pas !");
</SCRIPT>
```

### 5.1.4. switch

Le mot réservé **switch** permet en conjonction avec **case** de mettre en place un sélecteur de cas d'une grande souplesse.
Cette structure remplace avantageusement une structure équivalente construite à partir de if else et if else imbriqués portant sur une même variable.
Le **switch case** est une structure de code qui permet d'exécuter des actions différentes selon le contenu d'une expression ou d'une variable. Le mot clé **switch** est suivi de l'expression ou de la variable entre parenthèses. Le mot clé **case** identifie en suite ce que pourrait contenir la variable. Les blocs d'instructions se terminent par **break**, ce qui fait sortir du switch case. La dernière instruction peut être un **default**, et indique les instructions à accomplir si le contenu de la variable ne correspond pas aux cases précis. Le default n'est pas obligatoire, mais on ne l'enlève que lorsqu'on est sûr qu'il est impossible que la variable contienne une valeur non comblée par un case.

**Exemple**

```
switch(filiere)
    {
    case "ntic" :
        document.writeln("Nouvelles technologies de l'information et de
la communication.");
        break;
    case "ig" :
        document.writeln("Informatique de gestion. ");
        break;
    case "tlc" :
        document.writeln("Télécommunication. ");
        break;
    default :
        document.writeln("Désolez, aucune filière. ");
    }
```

Si la variable nommée filiere est égale à (donc contient textuellement) l'un des choix indiqués par un case, l'instruction associée (ou le groupe d'instructions) sera exécutée.

*Il ne faut pas oublier de finir les instructions ou blocs d'instructions par un break; , ceci peut créer des erreurs difficile à repérer*

**Exemple**

```
switch(ma_var)
{
var egal_deux = 2
     case 1 :
          alert("la variable vaut 1");
     break;
     case egal_deux :
          alert("la variable vaut 2");
     break;
     default : alert("la variable vaut autre chose que 1 ou 2");
}
```

L'instruction **switch** est aussi une instruction conditionnelle mais qui sert à traiter une variable selon plusieurs résultats qu'elle peut avoir

## 5.2. les boucles

### 5.2.1. la boucle for

L'instruction *for* permet d'exécuter plusieurs fois la même série d'instructions ;
Le nombre de fois d'exécution de notre série d'instruction est fini (connu d'avance)

**syntaxe:**

```
for (compteur; condition; modification du compteur)
{
liste d'instructions ;
}
```

Dans sa syntaxe, il suffit de préciser le nom de la variable qui sert de compteur avec sa valeur de départ, la condition sur la variable pour laquelle la boucle s'arrête (basiquement une condition qui teste si la valeur du compteur dépasse une limite) et enfin une instruction de modification qui incrémente (ou décrémente) le compteur.

**Exemple**:

```
<SCRIPT language = "JavaScript" >
document.write("Table des carrés<BR>");
for (var i = 0; i <15; i++) {
   document.write(i+"² = "+ i*i+"<BR>");
   }
</SCRIPT>
```

**Remarque**

*il faudra toujours vérifier que la boucle a bien une condition de sortie (i.e le compteur s'incrémente correctement)*
*il faut bien compter le nombre de fois que l'on veut faire exécuter la boucle*

### 5.2.2. la boucle while (tant que)

L'instruction répétitive while permet de répéter une séquence d'instructions tant qu'une expression conditionnelle est vraie.

**syntaxe:**

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

**Exemple**

```
<SCRIPT language = "JavaScript" >
a=parseInt(window.prompt("entrer un nombre ?", ""));
```

```
var b;
var prem=true;
b = a-1;
while (b>1 && prem)
{
if (a%b==0) prem = false; b--;
}
if(prem)
{
document.write ("le nombre " +a+" est premier") ;
}
else
{document.write ("le nombre " +a+" n'est pas premier") ;
}
</SCRIPT>
```

*La fonction parseInt() permet de convertir une chanine de caractères en nombre*

### 5.2.3. La boucle do … while

L'instruction répétitive *do … while* permet de répéter une séquence d'instructions jusqu'à la satisfaction d'une condition.

**syntaxe:**

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

**Exemple**

```
<SCRIPT language = "JavaScript" >
var compteur = 0;
do
   {
   document.write(compteur + " ");
   compteur = compteur +3;
   }
while(compteur<=100)

</SCRIPT>
```

## 5.3. Break

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

**Exemple :**

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

</SCRIPT>
```

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
```

## 5.4. 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).

**Exemple ;**

```
<SCRIPT language = "JavaScript" >

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

</SCRIPT>
```

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
```

# VI. LES FONCTIONS

## 6.1. Définition

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 script. C'est à celles-là que nous nous intéressons maintenant.

## 6.2. Déclaration des fonctions

On distingue traditionnellement les procédures et les fonctions.
Javascript ne différencie pas leur syntaxe
Pour déclarer ou définir une fonction, on utilise le mot (réservé) function.
**Syntaxe**:

```
function nom_de_la_fonction (liste d'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, peuvant inclure des chiffres …). Pour rappel, Javascript est sensible à la case. Ainsi **nomfonction**() ne sera pas égal à **Nomfonction**().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.
*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 son code est exécuté.

## 6.3. 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'interprétation lit votre script de haut vers le bas) que votre fonction soit bien définie avant d'être appelée.

## 6.4. 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'est à dire 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 la balise <BODY>.

```
<HTML>
<HEAD>
<SCRIPT language="Javascript">
<!--
 function Chargement()
{
alert('Bienvenue au CUP');
}
 //-->
</SCRIPT>
</HEAD>
<BODY onLoad="Chargement();">
contenu du document
 </BODY>
</HTML>
```

## 6.5. Passer une valeur à 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.

**Exemple.**

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function cube(nombre)
{
calcul= nombre*nombre*nombre ;
return calcul ;
}
</SCRIPT>
</HEAD>
<BODY>
<script language="javascript">
chiffre=parseInt(prompt("donnez un nombre",""));
document.write(chiffre+" au cube est égale à : "+cube(chiffre)+" . Mais
chiffre est toujours égalle à : "+chiffre);
</script>
</BODY>
</HTML>
```

Le nom de la variable est **nombre** et est définie comme un paramètre de la fonction. Dans l'appel de la fonction, on lui fournit la valeur de la variable chiffre :

## 6.6. Passer plusieurs valeurs à 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 …
}
```

**Exemple**

```
<HTML>
```

```
<HEAD>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript" type="text/javascript">
function bienvenue(nom,classe,cours)
{
alert("Bonjour, "+nom+"\nVous êtes etudiant en "+classe+" et au module de
"+cours)
}
bienvenue("Koffi","CUP","Html")
</SCRIPT>
</BODY>
</HTML>
```

## 6.7. 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.

**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.

La valeur de la variable retournée peut être utilisée dans une expression

```
 document.write (cube (5));
```

## 6.8. 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 (au dela de la fonction cube) dans le script, cette variable sera inconnue pour l'interpréteur Javascrript (message d'erreur).

Si la variable est déclarée contextuellement (sans utiliser le mot var) ; sa portée sera globale et elle est accessible par n'importe quelle partie du script.

```
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.

**Remarque**

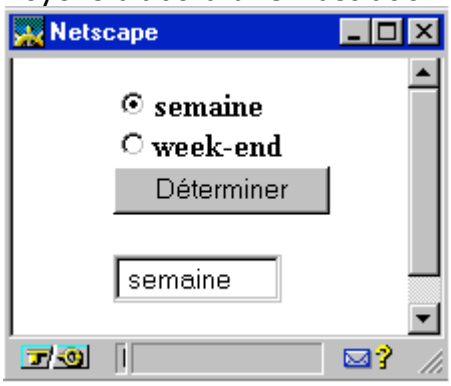
*Pour la facilité de gestion des variables, on ne peut que conseiller de les déclarer en début de script (comme dans la plupart des langages de programmation). Cette habitude vous met à l'abri de certaines complications.*

# VII. UTILISATION DES OBJETS EN JAVASCRIPT

Javascript est un langage « orienté objet ». Nous ne développerons pas ici les concepts de la programmation orientée objet, mais nous nous limiterons à l'utilisation et à la manipulation des objets avec Javascript.

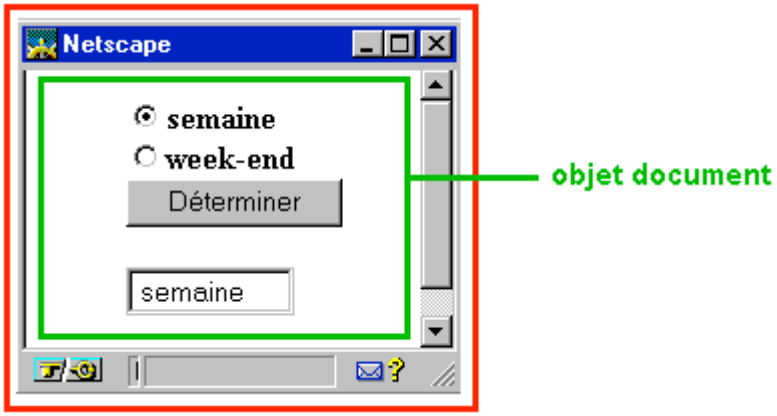
## 7.1. Les objets et leur hiérarchie

Javascript divise toute page web en objets et surtout permet d'accéder à ces objets, d'en retirer des informations et de les manipuler. Ces objets sont classés de manière hiérarchique, l'objet le plus haut dans la hiérarchie (arborescente) étant l'objet fenêtre. Voyons d'abord une illustration des différents objets qu'une page peut contenir.

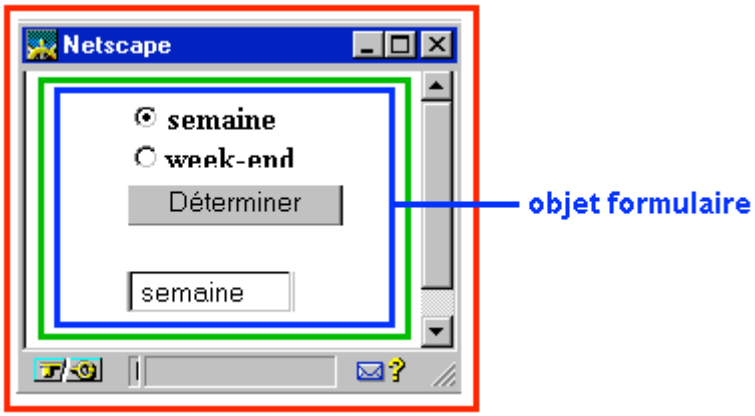



Cette page s'affiche dans une fenêtre. C'est **l'objet fenêtre**.

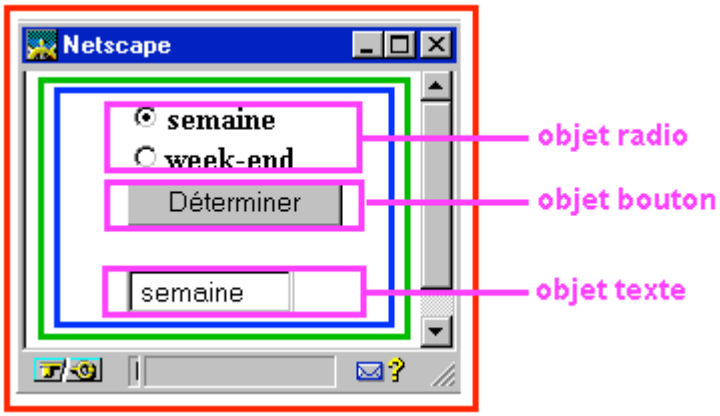
Dans cette fenêtre, il y a un document Html. C'est l'objet document. Autrement dit l'objet fenêtre contient l'objet **document**. D'où la notion de la hiérarchie des objets.




Dans le document précédent, 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 bouton et l'objet texte.




La hiérarchie des objets de cet exemple est donc :

```
                                        Radio
Fenetre        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 référencé.
Soit par exemple, pour le bouton radio ″semaine ″ :

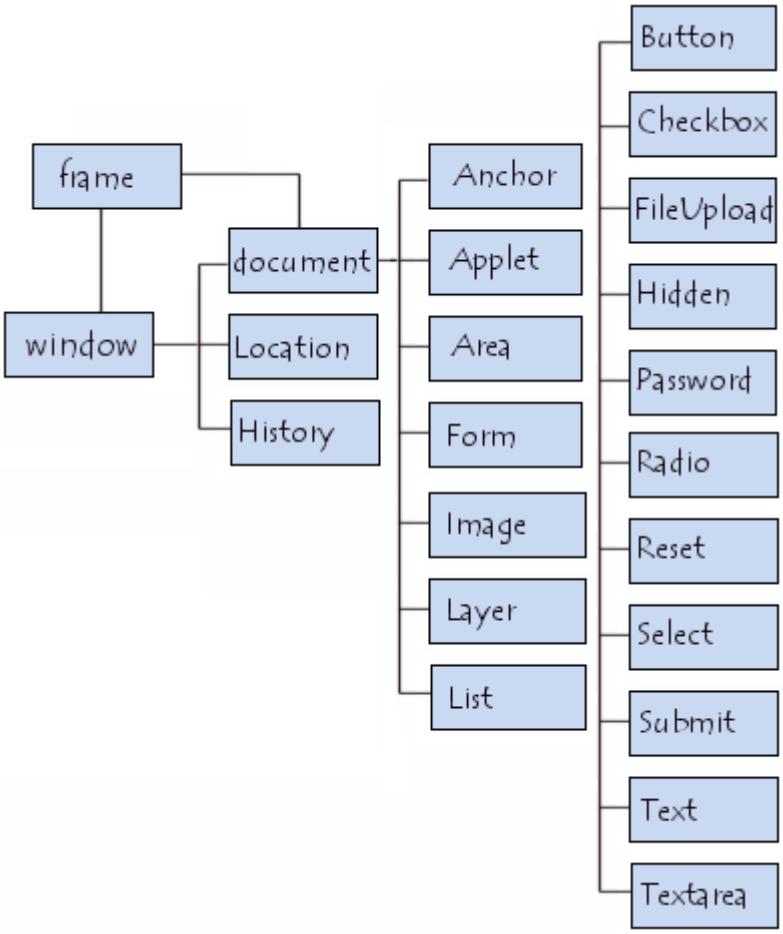
```
(window).document.form.radio [0]
```

l'objet window est mis entre parenthèses car comme il occupe la première place dans la hiérarchie, il est repris par défaut par javascript et devient donc facultatif. Les deux déclarations suivantes sont donc équivalentes, la deuxième étant simplement plus concise : var url = window.location.bref et var url = location.href ;
Pour chaque objet HTML nommé (avec l'attribut NAME), le navigateur crée un objet javascript correspondant de même nom. Si dans la partie HTML de la page on trouve la déclaration d'une image comme suit :

```
<img name = ″image1″ scr = ″img.gif″>
```

Alors on peut désigner cette image par **document.image1** en javascript. On peut également faire référence à cette image par document.images [″image1″] ou encore document.images [0] en supposant que c'est la première image de la page.

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 base sur les objets.
Les objets javascript sont nombreux et variés.
Pour vous donner une idée, voici la hiérarchie des objets standard du navigateur (cas de Netscape extrait du site officiel).
*Les objets du navigateur sont classés dans une hiérarchie qui décrit la page affichée à l'écran, et qui permet d'accéder à n'importe quel objet grâce à une désignation dépendant de la hiérarchie (on part du sommet puis on descend l'arborescence).*




Dans cette hiérarchie, les descendants d'un objet sont des propriétés de ces objets mais peuvent aussi être des objets qui contiennent eux même des propriétés...

## 7.2. Désignation de l'objet courant : this

Il est souvent commode de pouvoir désigner l'objet manipulé sans devoir le nommer de manière explicite, comme nous l'avons décrit précédemment. Il existe le mot clé **this** qui permet de désigner l'objet en cours de manipulation. Cette écriture raccourcie est souvent utilisée (sans risque de confusion) en remplacement du chemin complet de l'objet dans un formulaire.

**Exemple.**

```
<FORM name="form2">
   <INPUT type="button" name="bouton" value='Cliquez-moi !'
onClick="this.value='Touché !'">
</FORM>
```

## 7.3. Les propriétés, les méthodes et les classes des objets

### 7.3.1. Les propriétés des objets

Une propriété est un attribut, une caractéristique, une description de l'objet.
Si une voiture était un objet au sens javascript, ses propriétés pourraient être par exemple sa couleur, son prix, son poids, etc… Tout ce qui la caractérise d'une autre voiture.
L'objet livre a comme propriétés son auteur, son titre, son édition, etc.
De même, les objets javascript ont des propriétés personnalisées. Ces propriétés dépendent bien sûr de l'objet considéré. Par ailleurs, chaque propriété, possédera une valeur (value en anglais). Par exemple, la valeur de la propriété couleur de l'objet voiture pourrait être rouge , ou vert…
Pour accéder aux propriétés, la syntaxe est :

```
nom_de_l'objet.nom_de_la_propriété
```

Pour la valeur, on indiquera :

```
nom_de_l'objet.nom_de_la_propriété.value
```

Dans le cas des boutons radio, une de ses propriétés est par exemple, sa selection ou sa non-selection (checked en anglais) et la valeur est soit true soit false. Pour tester la propriété de sélection, on écrira :

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

Pour être complet, this peut aussi être utilisé pour créer une ou plusieurs propriétés d'une boite. 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 ;
}
```

### 7.3.2. Les méthodes des objets

Toujours au sens javascript, les objets possèdent des ″Méthodes″. C'est en quelque sorte, ce que l'objet en question est capable de faire (les actions qu'il peut accomplir dans un programme javascript). Dans l'exemple de notre voiture, elle est capable d'avancer, de reculer, de tourner etc…Nous dirons alors que avancer, reculer, tourner, sont les ″Méthodes″ de l'objet voiture.
Ici encore, on voit bien que les méthodes d'un objet, dépendent de l'objet considéré.
Si une voiture est un objet et qu'une machine à laver est un autre, ils ne peuvent pas avoir tous les deux les mêmes méthodes.
La même notation que pour les propriétés s'applique pour exécuter les méthodes.
Par exemple, une des méthodes de l'objet document est writeln ( ) tandis que alert ( ) est une méthode de l'objet window. Ainsi, on notera : document.writeln (″texte″) ou window.alert (″message de bienvenue″).

### 7.3.3. Les classes d'objets

On appelle Classe d'objets la description d'un ensemble de propriétés et de méthodes liées à un objet. Les propriétés sont les données, et les méthodes sont les fonctions qui manipulent ces données. Un objet est donc une instance de classe.
Nous retrouvons naturellement, parmi les classes d'objets prédéfinies dans le langage, les classes :

- array : les tableaux
- boolean : les booléens
- date : les dates
- math : les constantes et fonctions mathématiques
- number : les valeurs numériques (entiers ou réels)
- string : les chaînes de caractères

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

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

Notons que texte 2 a été initialisé simultanément à sa déclaration. On peut aussi désigner la propriété longueur de cette chaîne de caractère par texte2.length ;
Toujours pour la classe String, la méthode prédéfinie concat () permet de concaténer (mettre bout-à-bout) deux chaînes. Si, dans la continuité de l'exemple précédent on écrit :

```
Texte1 = texte2.concat ("monsieur nabeledi") ;
```

Alors le contenu de l'objet texte1 devient " bonjour monsieur nabeledi", c'est-à-dire la concaténation de la chaîne contenue dans le texte2 et de la chaîne "monsieur nabeledi".
Les noms choisis par le navigateur pour instancier les objets des différentes classes sont les mêmes que ceux de la classe mais typographiés en minuscule. Quand plusieurs objets appartiennent à une même classe, ils sont regroupés dans un tableau et peuvent être distingués par leur place dans le tableau. Ainsi, l'objet (unique) de la classe window s'appelle window et les différents objets cadres (éventuellement plusieurs dans une fenêtre) s'appellent window.frames [0], document.frames[1], etc.

## 7.4. L'objet window

### 7.4.1. Les particularités de l'objet window

L'objet window est l'objet par excellence dans Javascript, car il est le parent de chaque objet qui compose la page web, il contient donc :

- l'objet document: la page en elle-même
- l'objet location: le lieu de stockage de la page
- l'objet history: les pages visitées précédemment
- l'objet frames: les cadres (division de la fenêtre en sous-fenêtres)

### 7.4.2. Les propriétés de l'objet window

| propriété | description | lecture seule |
|---|---|---|
| defaultStatus | Il s'agit du message qui s'affiche par défaut dans la barre d'état du navigateur | non, modifiable à tout moment |
| frames | Il s'agit d'un tableau qui contient les cadres présents dans la fenêtre | Tous les éléments de *frames* sont en lecture seule |
| length | nombre de cadres (nombre d'éléments du tableau *frames* | Lecture seule |
| name | nom de la fenêtre dans laquelle on se trouve | Lecture seule |
| parent | Il s'agit de la fenêtre qui englobe celle dans laquelle on se trouve (si il y en a une..) | Lecture seule, contient des propriétés |
| self | Synonyme de la fenêtre actuelle | Lecture seule, contient des propriétés |

| | | |
|---|---|---|
| | (redondance ou précision?) | |
| status | Il s'agit d'un message temporaire qui s'affiche dans la barre d'état du navigateur suite à un événement | non, modifiable à tout moment, vous devez retournez la valeur *true* pour l'utiliser avec *onMouseOver* |
| Top | Il s'agit de la fenêtre de plus haut niveau, celle qui contient tous les cadres (frames) | Lecture seule, contient des propriétés |
| window | Il s'agit de la fenêtre actuelle... | Lecture seule, contient des propriétés |

- Les propriétés window, self, frames, top et parent permettent de naviguer dans le système de sous-fenêtres, appelées *frames*.
- les propriétés *self* et *window* sont les mêmes, elles permettent de désigner la page en cours, leur seul but est d'accentuer la précision de la hiérarchie pour une meilleure lecture du code. en effet, *self.name* est plus explicite que *name*
- les propriété *top* et *parent* permettent de remonter dans la hiérarchie pour atteindre des fenêtre de niveau supérieur, notamment pour "sortir" des frames...
- la propriété *frames* est un tableau qui permet de cibler un cadre spécifique faisant partie de la fenêtre où on se trouve. Pour désigner un cadre on utilise soit la notation *window.frames[i]* où *i* représente le numéro du cadre, soit *window.nom_du_cadre* en désignant le cadre directement par le nom qu'on lui a assigné dans la balise *frameset*.

### 7.4.3. Les méthodes de l'objet window

L'objet window possède des méthodes relatives à l'ouverture et à la fermeture des fenêtres.

- **Les méthodes alert(), confirm() et prompt()**

Les méthodes alert(), confirm() et prompt() sont des méthodes qui font apparaître une boîte de dialogue, elles sont expliquées en détail dans le chapitre Boîte de dialogue.

- **Les méthodes open(), et close()**

Les méthodes open() et close() sont bien évidemment destinées à permettre l'ouverture et la fermeture de fenêtres. Toutefois la syntaxe de la méthode open() est longue car elle admet un nombre important de paramètres :

**La méthode open() permet d'ouvrir une fenêtre:**

```
window.open("URL","nom_de_la_fenetre","options_de_la_fenetre");
```

Si vous utilisez cette instruction dans un lien hypertexte, veillez à remplacer les guillemets doubles par des guillemets simples :

```
<A href="javascript:window.open('URL','nom_de_la_fenetre','options_de_la_
fenetre')">Lien vers URL</A>
```

Pour les mêmes raisons, le nom de la fenêtre ne doit pas contenir de guillemets. Si cela devait arriver, vous avez la possibilité de remplacer les guillemets doubles par leur équivalent HTML ("), et les apostrophes par "\'" ou bien "’".
URL désigne l'url de la page qui sera affichée dans la nouvelle fenêtre, c'est-à-dire l'emplacement physique de celle-ci.

**Les options de la fenêtre sont les suivantes:**

| option | Description |
|---|---|
| directories = yes/no | Affiche ou non les boutons de navigation |
| location = yes/no | Affiche ou non la barre d'adresse |
| menubar = yes/no | Affiche ou non la barre de menu (fichier, edition, ...) |
| resizable = yes/no | Définit si la taille de la fenêtre est modifiable ou non |
| scrollbars = yes/no | Affiche ou non les ascenceurs (barres de défilement) |

| | |
|---|---|
| status = yes/no | Affiche ou non la barre d'état |
| toolbar = yes/no | Affiche ou non la barre d'outils |
| width = largeur (en pixels) | Définit la largeur |
| height = hauteur (en pixels) | Définit la hauteur |

Ainsi, il est possible d'utiliser cette méthode avec n'importe quel gestionnaire d'événement, directement dans le code à exécuter ou bien dans une fonction.

- les options doivent être saisies les unes après les autres, séparées par des virgules, sans espace
- l'ensemble des options doit être encadré par les guillemets

Chacune des fenêtres doit cependant être fermée, il faut donc se servir de la méthode close() qui permet de fermer une fenêtre.
La méthode close() requiert le nom de la fenêtre comme argument, il suffit donc de créer un bouton (image, hypertexte, ou bouton de formulaire) qui permettra de fermer cette fenêtre.

**Pour un lien hypertexte, le code sera :**

```
<A href="javascript:self.close('nom_de_la_fenetre_');">
Cliquez ici pour fermer la fenêtre
</A>
```

**Pour un bouton (image), le code sera :**

```
<A href="javascript:self.close('nom_de_la_fenetre_');">
<img src="nom_de_l_image">
</A>
```

Il est bien évidemment possible d'utiliser cette procédure avec tous les gestionnaires d'événement, en utilisant par exemple une syntaxe proche de celle-ci :

```
<A href="javascript:;" onMouseOver="self.close('nom_de_la_fenetre_');" >
<img src="nom_de_l_image">
</A>
```

## 7.5. L'objet String

### 7.5.1. Généralités

En Javascript, les chaînes de caractères (par exemple, 'CUP se trouve à Cocody', ou " ici, nous apprenons le javascript avec Monsieur Nabélédi") sont en fait des objets String.
String est ce qu'on appelle un objet intégré, car il existe déjà.
On peut créer un objet String de deux manières :

```
var Chaine = new String("bonjour !");
document.write(typeof Chaine);// Sortie -> object
```

ou

```
var Chaine = "bonjour !";
document.write(typeof Chaine);// Sortie -> string
```

Vous remarquez que le type renvoyé par l'opérateur typeof n'est pas le même. C'est cependant bel et bien la même chose.
String n'échappe pas à la règle : lui aussi possède des propriétés et des méthodes.

### 7.5.2. Propriétés

- **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.

**syntaxe:**

```
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 ou d'éléments dans le formulaire ?
- de boutons radio. Combien y a-t-il de boutons radio dans un groupe ?
- de cases à cocher. Combien y a-t-il de cases à cocher dans un groupe ?
- d'options. Combien y a-t-il d'options dans un Select ?
- de frames. Combien y a-t-il de frames "enfants" ?
- d'ancres, de liens, etc.

### 7.5.3. Méthodes

Nous allons voir quelques méthodes ; la liste étant longue.

- **charAt(Index)**

Retourne le caractère positionné à l'index spécifié par l'argument.
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.

**syntaxe:**

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

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

**exemples**

```
var str="Javascript";
var chr=str.charAt(0);
var chr="Javascript".charAt(0);
ou var chr=charAt(str,0);
ou var chr=charAt("Javascript",0);
```

La réponse est "J".

```
var str="Javascript";
var chr=str.charAt(9);
var chr=charAt(str,9);
```

La réponse est "t".

```
var str="Javascript";
var chr=charAt(str,13);
```

La réponse est ""
soit vide.

- **indexOf(chaine, Index)**

Recherche la chaine *chaine* dans l'objet à partir du caractère *Index* si celui ci est spécifié. Sinon, la recherche s'effectue à partir du début de la chaîne.
En cas de succés, la méthode renvoi la position de *chaine* dans l'objet, sinon la valeur -1 est retournée :

```
var Chaine = "Joyeux Noël";
document.write(Chaine.indexOf("Noël"));// Sortie -> 7
document.write(Chaine.indexOf("Anniversaire"));// Sortie -> -1
```

- **lastIndexOf(chaine, Index)**

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

**Exemple**

```
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.

- **blink()**

Retourne la chaîne de caractères encadrée des balises `<BLINK>` et `</BLINK>`.

```
var Chaine = "Clignotant";

document.write(Chaine.blink());// ecrit <BLINK>Clignotant</BLINK>
```

- **fontcolor(couleur)**

Retourne la chaîne de caractères encadrée des balises <FONT color="couleur" >et </FONT>, où couleur est l'argument transmis à la méthode.

**Syntaxe :**

```
String chaine.fontcolor(String couleur)
```

**Exemple :**

```
var Chaine = "La couleur";

document.write(Chaine.fontcolor("RED"));//<FONT color=RED>La
couleur</FONT>
ou
document.write(Chaine.fontcolor("#00FF00"));
```

*Remarque*
*il en sera de même pour plusieurs méthodes*

| *METHODE* | *EFFET* |
|---|---|
| *big()* | *<big> et </big>* |
| *bold()* | *<B> et </B>* |
| *fixed()* | *<PRE>et </PRE>* |
| *fontsize(taille)* | *<FONT size="taille"> et </FONT>* |
| *italics()* | *<I> et </I>* |
| *link(URL)* | *<A href="URL"> et </A>* |
| *strike()* | *<STRIKE> et </STRIKE>.* |
| *sub()* | *<SUB> et </SUB>* |
| *sup()* | *<SUP> et </SUP>* |

*Etc..*

- **Substring(debut,fin)**

Extrait une sous-chaîne d'une chaîne en partant de l'indice debut et jusqu'à l'indice fin.
Si fin n'est pas précisé, la chaîne est extraite depuis le début jusque sa fin.
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.

**Syntaxe**

```
String chaine.substring(Integer debut [, Integer fin])
```

**Exemple**

```
<SCRIPT language=javascript>
var date = "06/12/2007";
var jour = date.substring(0,2);
var mois = date.substring(3,5);
var annee = date.substring(6,10);
document.write(jour+"<BR>");
document.write(mois+"<BR>");
document.write(annee+"<BR>");
// 06
// 12
// 2007
</SCRIPT>
```

- **slice(deb, fin)**

Retourne la chaîne de caractère située entre le caractère `deb` et le caractère `fin`. Si `fin` est omis, slice renvoi les caractères jusqu'à la fin de la chaîne.

```
var Chaine = "Sad Hill";
document.write(Chaine.slice(4, 8));// Sortie -> Hill
```

- **split(RegExp)**

**Syntaxe :**

```
Array chaine.split(RegExp motif)
```

Retourne un tableau de sous-chaînes de caractères en utilisant comme séparateur le motif de l'expression régulière.

**Exemple Découpage de chaîne**

```
<SCRIPT language=javascript>
var chaine="Jean-Paul, Arthur ; Léon, Marcel ; Paul";
var reg=new RegExp("[ ,;]+");
document.write("Chaîne d'origine : " + chaine + "<BR>");
var tableau=chaine.split(reg);
for (var i=0; i<tableau.length; i++) {
 document.write("tableau[" + i + "] = " + tableau[i] + "<BR>");
}
</SCRIPT>
```

```
Chaîne d'origine : Jean-Paul, Arthur ; Léon, Marcel ; Paul
tableau[0] = Jean-Paul
tableau[1] = Arthur
tableau[2] = Léon
tableau[3] = Marcel
tableau[4] = Paul
```

*Explication*

*Ce script extrait de la variable chaine tous les prénoms séparés par des espaces, des , ou des ;, grâce au motif [ ,;]+ qui indique la présence d'au moins un espace, une virgule ou un point-virgule.*

- **toLowerCase()**

Retourne la chaîne de caractères avec tous les caractères transformés en minuscules.

```
var Chaine = "Hello";
```

```
document.write(Chaine.toLowerCase());// Sortie -> hello
```

- **toUpperCase()**

Retourne la chaîne de caractères avec tous les caractères transformés en majuscules.

```
var Chaine = "Hello";
document.write(Chaine.toUpperCase());// Sortie -> HELLO
```

**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.*

## 7.6. L'objet Math

Pour manipuler les nombres, voici l'objet Math.

### 7.6.1. Les méthodes

**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
28 soit 256
```

**random()**

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

La méthode random() renvoie la valeur d'un nombre aléatoire choisi entre 0 et 1.

```
<SCRIPT language="javascript">

for(var i=0;i<10;i++) {
 document.write(Math.random());
 document.write("<BR>");
}// renvoi 10 nombres décimaux aléatoire
</SCRIPT>
```

**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.

*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.*

```
<script language="javascript">
str='-43.12345';
str2='$5.50';
x=parseFloat(str);
y=parseFloat(str2);

document.write(x); // on a -43.12345
document.write(y); // on a 0 ou NaN en fonction des navigateurs et des os

</script>
```

**parseInt()**

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

Retourne la partie entière d'un nombre avec une virgule.
Convertit une chaîne en un nombre entier

```
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().

### 7.6.2. 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);
```

### 7.6.3. 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;
```

### 7.6.4. recapitulatif

| Propriétés | Description |
|---|---|
| E | Constance d'Euler (environ 2,718) |
| LN2 | Logarithme de 2 (environ 0,693) |
| LN10 | Logarithme de 10 (environ 2,302) |
| LOG2E | Logarithme base 2 de e (environ 1,442) |
| LOG10E | Logarithme base 10 de e (environ 0,434) |
| PI | Valeur de Pi (environ 0,707) |
| SQRT1_2 | Racine carrée de ½ (environ 0,707) |
| SQRT2 | Racine carrée de2 (environ 1,414) |

| Méthodes | Description |
|---|---|
| abs (x) | Renvoie la valeur absolue de x |
| acos (x) | Renvoie arc cosinus de x |
| asin (x) | Renvoie arc sinus de x |
| atan (x) | Renvoie l'arc tangente de x |
| ceil (x) | Renvoie l'entier immédiatement inférieur ou égal à x |
| cos (x) | Renvoie le cosinus de x |
| exp (x) | Renvoie l'exponentiel de x |
| floor (x) | Renvoiel'entier immédiatement inférieur ou égale à x |
| log (x) | Renvoie le logarithme de x |
| max (x, y) | Renvoie le maximum de x et y |
| min (x, y) | Renvoie le minimum de x et y |
| parslnt(x) | Renvoie la partie entière d'un nombre àvirgule x (convertit aussi une chaaine de caractère en nombre) |
| pow (x,y) | Renvoie x exposant y |
| random ( ) | Renvoie un nombre aléatoire entre 0 et 1 |
| round (x) | Renvoie l'entier le plus proche de x |
| sin (x) | Renvoie le sinus de x |
| sqrt (x) | Renvoie la raine carrée de x |
| tan (x) | Renvoie la tangente de x |

## 7.7. L'objet Date

### 7.7.1. introduction

L'objet *Date* permet de travailler avec toutes les variables qui concernent les dates et la gestion du temps. Il s'agit d'un objet inclus de façon native dans Javascript, et que l'on peut toujours utiliser.

**La syntaxe de base pour créer un objet date est :**

```
Nom_de_l_objet = new Date();
```

Cette syntaxe permet de stocker la date et l'heure actuelle.
Elle a d'autres variantes avec des paramètres particuliers :

1. **Nom_de_l_objet = new Date("jour, mois date année heures:minutes:secondes")**
   *les paramètres sont une chaîne de caractère suivant scrupuleusement la notation ci-dessus*
2. **Nom_de_l_objet = new Date(année, mois, jour)**
   *les paramètres sont trois entiers séparés par des virgules.*
   *Les paramètres omis sont mis à zéro par défaut*
3. **Nom_de_l_objet = new Date(année, mois, jour, heures, minutes, secondes[, millisecondes])**
   *les paramètres sont six entiers séparés par des virgules.*
   *Les paramètres omis sont mis à zéro par défaut*

Les dates en Javascript sont stockées de la même manière que dans le langage Java, c'est-à-dire qu'il s'agit du nombre de millisecondes depuis le 1er janvier 1970. Ainsi, toute date antérieure au 1er janvier 1970 fournira une valeur erronée.
*Avec les versions de Javascript inférieures à la version 1.3, pour manipuler des dates antérieures à "l'année zéro" il vous sera nécessaire de créer un objet date spécifique.*
*A partir de la version 1.3, il est possible de manipuler des dates de plus ou moins 100 000 000 de jours par rapport au premier janvier 1970.*

### 7.7.2. Les méthodes de l'objet *Date*

La date est stockée dans une variable sous la forme d'une chaîne qui contient le jour, le mois, l'année, l'heure, les minutes, et les secondes. Il est donc difficile d'accéder à un seul élément d'un objet date avec les fonctions de manipulation de chaînes de caractères, étant donné que chacun des éléments peut avoir une taille variable. Heureusement, les méthodes de l'objet Date fournissent un moyen simple d'accéder à un seul élément, ou bien de le modifier.

**Leur syntaxe est la suivante :**

```
Objet_Date.Methode();
```

- **Connaître la date**

Les méthodes dont le nom commence par le radical *get* (mot anglais qui signifie *récupérer*) permettent de renvoyer une partie de l'objet *Date* :

| Méthode | Description | Type de valeur retournée |
|---|---|---|
| getDate() | Permet de récupérer la valeur du jour du mois | L'objet retourné est un entier (entre 1 et 31) qui correspond au jour du mois : |
| getDay() | Permet de récupérer la valeur du jour de la semaine pour la date spécifiée | L'objet retourné est un entier qui correspond au jour de la semaine :<br>0: dimanche<br>1: lundi ... |
| getFullYear() | Permet de récupérer la valeur de l'année sur 4 chiffres pour la date passée en paramètre | L'objet retourné est un entier qui correspond à l'année (XXXX) : 2007 |

| | | |
|---|---|---|
| getHours() | Permet de récupérer la valeur de l'heure | L'objet retourné est un entier (entre 0 et 23) qui correspond à l'heure de l'objet Date. |
| getMilliseconds() | Permet de récupérer le nombre de millisecondes | L'objet retourné est un entier (entre 0 et 999) qui correspond aux millisecondes de l'objet passé en paramètre. |
| getMinutes() | Permet de récupérer la valeur des minutes | L'objet retourné est un entier (entre 0 et 59) qui correspond aux minutes de l'objet Date. |
| getMonth() | Permet de récupérer le numéro du mois | L'objet retourné est un entier (entre 0 et 11) qui correspond au mois :<br>0: janvier<br>1: février ... |
| getSeconds() | Permet de récupérer le nombre de secondes | L'objet retourné est un entier (entre 0 et 59) qui correspond aux secondes de l'objet passé en paramètre. |
| getTime() | Permet de récupérer le nombre de millisecondes depuis le 1er janvier 1970 | L'objet retourné est un entier. Cette méthode est très utile pour convertir des dates, soustraire ou ajouter deux dates, etc. |
| getTimezoneOffset() | Retourne la différence entre l'heure locale et l'heure GMT (Greenwich Mean Time) | L'objet retourné est un entier, il représente le nombre de minutes de décalage |
| getYear() | Permet de récupérer la valeur de l'année sur 2 chiffres (les deux derniers) pour l'objet Date. | L'objet retourné est un entier qui correspond à l'année (XX) :<br>07 |

- **Modifier le format de la date**

Les deux méthodes suivantes ne permettent de travailler que sur l'heure actuelle (objet *Date()*) leur syntaxe est donc figée :

| Méthode | Description | Type de valeur retournée |
|---|---|---|
| toGMTString() | Permet de convertir une date en une chaîne de caractères au format GMT | L'objet retourné est une chaîne de caractère du type :<br>Wed, 27 Nov 2007 15:15:20 GMT |
| toLocaleString() | Permet de convertir une date en une chaîne de caractères au format local | L'objet retourné est une chaîne de caractère dont la syntaxe dépend du système, par exemple :<br>27/11/07 15:15:20 |

- **Modifier la date**

Les méthodes dont le nom commence par le radical *set* (mot anglais qui signifie *règler*) permettent de modifier une valeur :

| Méthode | Description | Type de valeur en paramètre |
|---|---|---|
| setDate(X) | Permet de fixer la valeur du jour du mois | Le paramètre est un entier (entre 1 et 31) qui correspond au jour du mois |
| setDay(X) | Permet de fixer la valeur du jour de la semaine | Le paramètre est un entier qui correspond au jour de la semaine :<br>0: dimanche<br>1: lundi ... |
| setHours(X) | Permet de fixer la valeur de l'heure | Le paramètre est un entier (entre 0 et 23) qui correspond à l'heure |
| setMinutes(X) | Permet de fixer la valeur des minutes | Le paramètre est un entier (entre 0 et 59) qui correspond aux minutes |
| setMonth(X) | Permet de fixer le numéro du mois | Le paramètre est un entier (entre 0 et 11) qui correspond au mois :<br>0: janvier<br>1: février ... |
| setTime(X) | Permet d'assigner la date | Le paramètre est un entier représentant le nombre de millisecondes depuis le 1er janvier 1970 |

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

```
<SCRIPT language=javascript>
var ladate=new Date()
document.write("Heure brute : ");
document.write(ladate.getHours()+":"+ladate.getMinutes()+":"+ladate.getSe
conds())
document.write("<BR>");
var h=ladate.getHours();
if (h<10) {h = "0" + h}
var m=ladate.getMinutes();
if (m<10) {m = "0" + m}
var s=ladate.getSeconds();
if (s<10) {s = "0" + s}
document.write("Heure formatée : ");
document.write(h+":"+m+":"+s)
</SCRIPT>
```

## 7.8. L'objet Navigator

### 7.8.1. Les particularités de l'objet *navigator*

L'objet *navigator* est un objet qui permet de récupérer des informations sur le navigateur qu'utilise le visiteur. Cela paraît totalement inutile à première vue, toutefois, comme vous le savez sûrement, il existe de grandes différences entre différentes versions d'un même navigateur (intègration de nouvelles technologies), ainsi qu'entre des navigateurs de types différents (les deux antagonistes sont généralement Netscape Navigator © et Microsoft Internet Explorer qui d'une part n'interprètent pas toutes les balises HTML et les instructions Javascript de la même manière, et qui, d'autre part, possède parfois des balises HTML propriétaires, c'est-à-dire qui leur sont propres...).
Toutes les propriétés de l'objet *navigator* sont en lecture seule, elles servent uniquement à récupérer des informations et non à les modifier (il serait idiot de vouloir modifier les informations d'un navigateur...).

### 7.8.2. Les propriétés de l'objet *navigator*

Les propriétés de l'objet *navigator* sont peu nombreuses, elles permettent en fait de retourner des portions de l'information sur votre navigateur qui est en fait une chaîne de caractères. Etant donné que ces propriétés sont statiques, il est nécessaire de les faire précéder par *navigator* :

```
navigator.propriete
```

Dans le tableau suivant, la colonne de droite donne le résultat fourni par la propriété pour votre navigateur :

```
<script language="Javascript">

<!--
document.write(navigator.propriete);

// -->

</script>
```

| Propriété | Description | Pour votre navigateur |
|---|---|---|
| appCodeName | retourne le code du navigateur. Un navigateur a généralement pour nom de code Mozilla, le moteur utilisé par la plupart des navigateurs (internet explorer, netscape, mais aussi beaucoup de navigateurs sous Unix...). Cette valeur sera différente si le navigateur du client est pas basé sur un autre moteur (e.g. Opera, ...). | Mozilla |
| appName | retourne le nom du navigateur (la plupart du temps la marque). Cette propriété est utile pour différencier les navigateurs de Netscape et de Microsoft). | Microsoft Internet Explorer |
| appVersion | retourne la version du navigateur. Cette propriété prend la forme suivante :<br>Numéro de version( plateforme (système d'exploitation), nationalité)<br>Elle est utile pour connaître le système d'exploitation de l'utilisateur, mais surtout, associée avec la propriété navigator.appName elle permet de connaître les fonctionnlités que supporte le navigateur de votre visiteur. | 4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322) |
| language | renvoie une chaîne de caractère donnant la langue utilisée par la navigateur du client. Cette propriété n'est comprise que par les navigateurs supportant les versions 1.2 et supérieures de Javascript. | undefined |
| mimeTypes | Cette propriété renvoie un tableau répertoriant les types MIME supportés par le navigateur, c'est-à-dire les types de fichiers enregistrés. | |
| platform | Cette propriété renvoie une chaîne de caractère indiquant la plateforme sur laquelle le navigateur fonctionne, c'est-à-dire le système d'exploitation du client. Cette propriété n'est comprise que par les navigateurs supportant les versions 1.2 et supérieures de Javascript. | Win32 |
| plugins | Cette propriété renvoie un tableau contenant la liste des plugins installés sur la machine cliente. | |
| userAgent | retourne la chaîne de caractère qui comprend toutes les informations sur le navigateur de client. Les propriétés ci-dessus offrent un moyen pratique de récupérer une partie de cette information. | Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322) |

### 7.8.3. Les méthodes de l'objet *navigator*

Les méthodes de l'objet *navigator* permettent d'effectuer certaines opérations concernant le navigateur du client. Dans la mesure où il s'agit de méthodes statiques, il est indispensable de les faire précéder par *navigator*.

| Méthode | Description | Pour votre navigateur |
|---|---|---|
| javaEnabled() | Cette méthode permet de vérifier si le navigateur du client est configuré pour exécuter des applets Java. | true |
| plugins.refresh() | La méthode refresh() de la propriété plugin permet de rafraîchir la liste des plugins installés sur le poste client. | |
| preference("preference",valeur) | Cette méthode supportée à partir de la version 1.2 de Javascript permet à un script signé de redéfinir les préférences du navigateur. Le script doit ainsi obtenir les privilèges suffisants pour pouvoir effectuer ces actions. | |
| SavePreferences() | Cette méthode supportée à partir de la version 1.2 de Javascript permet de sauvegarder les modifications apportées aux préférences du navigateur du client. | |
| taintEnabled() | Cette méthode permet de vérifier que la protection des données a été activée grâce à la méthode taint() de Javascript. Cette méthode est obsolète depuis la version 1.2 de Javascript, et ne fonctionne ainsi que sur Netscape Navigator 3. | |

*Remarque*
*Les navigateurs ne supportent pas tous le Javascript de la même façon, ainsi, suivant les instructions Javascript que l'on utilisera (suivant qu'il s'agit de Javascript 1.0, 1.1, ou 1.2) il faudra exécuter des instructions différentes. Par respect pour vos visiteurs il est généralement bon de mettre en début de script une fonction qui permet de déterminer la version du navigateur (et/ou sa marque) et exécuter les instructions appropriées en fonction du résultat.*
*Voici une grille avec les navigateurs, les valeurs de propriétés et quelques fonctionnalités associées à ceux-ci :*

| *Navigateur* | *Version* | *navigator.appName* | *navigator.appVersion* | *Javascript* |
|---|---|---|---|---|
| *Ns Navigator* | *2* | *Netscape* | *2.0 (Win95; I)* | *1.0* |
| *Ns Navigator* | *3* | *Netscape* | *3.xx (Win95; I)* | *1.1* |
| *Ns Communicator* | *4* | *Netscape* | *4.xx (Win95; I)* | *1.1* |
| *Ms Internet Explorer* | *3* | *Microsoft Internet Explorer* | *2.0 (compatible; MSIE 3.0x; Win*)* | *1.0* |
| *Ms Internet Explorer* | *4* | *Microsoft Internet Explorer* | *4.0x (compatible; MSIE 4.0x; Win*)* | *1.2* |
| *Ms Internet Explorer* | *5* | *Microsoft Internet Explorer* | *5.xx (compatible; MSIE 5.0x; Win*)* | *1.2* |

### Exemples de script

Il est parfois bon de pouvoir exécuter un ensemble d'instructions différent selon que le navigateur utilisé est Netscape Navigator ou Microsoft Internet Explorer, voici un exemple de petit script permettant la distinction :

```
Nom = navigator.appName;
```

```
if (Nom == 'Netscape') {
placer ici les instructions à exécuter s'il s'agit
de Netscape Navigator 4 ou supérieur
}

if (Nom == 'Microsoft Internet Explorer') {
placer ici les instructions à exécuter s'il s'agit
de Microsoft Internet Explorer 4 ou supérieur
}
```

Une méthode améliorée si jamais on a souvent besoin de faire le test de navigateur, car l'accès à l'objet *navigator* n'est effectué qu'une seule fois. Ensuite deux variables booléenne sont utilisées, et le test consiste uniquement à regarder si la variable contient 1 au 0 et le navigateur n'a pas besoin de comparer la chaine *nom* à chaque test de navigateur...

```
Nom = navigator.appName;
```

```
ns = (Nom == 'Netscape') ? 1:0
ie = (Nom == 'Microsoft Internet Explorer') ? 1:0

if (ns) {
placer ici les instructions à exécuter s'il s'agit
de Netscape Navigator 4 ou supérieur
}

if (ie) {
placer ici les instructions à exécuter s'il s'agit
de Microsoft Internet Explorer 4 ou supérieur
}
```

Imaginons que l'on veuille maintenant afficher du DHTML (HTML dynamique, c'est-à-dire dont les objets peuvent bouger et être modifiés à loisir...), il faut alors vérifier que les

versions des navigateurs sont supérieures à 4 (seuls Netscape Navigator 4 (ou supérieur) et Microsoft Internet Explorer 4 (ou supérieur) le supportent.
Voici le script permettant de vérifier que les versions sont supérieures à 4 :

```
Nom = navigator.appName;

Version = navigator.appVersion;

ns4 = (Nom == 'Netscape' && Version >= 4 ) ? 1:0
ie4 = (Nom == 'Microsoft Internet Explorer' && Version >= 4 ) ? 1:0

if (ns4) {
placer ici les instructions à exécuter s'il s'agit de
Netscape Navigator 4 ou supérieur
}


if (ie4) {
placer ici les instructions à exécuter s'il s'agit de
Microsoft Internet Explorer 4 ou supérieur
}
```

## 7.9. L'objet Array

Un tableau permet de stocker plusieurs valeurs de même type, comme par exemple plusieurs entiers dans une seule variable.
Les tableaux sont donc un type à part entière.

### 7.9.1. Tableau à une dimension

En JavaScript, la déclaration d'un tel objet se fait comme suit :

```
var nom_du_tableau = new Array();
var nom_du_tableau = new Array(n);//n précise le nombre d'élément du
tableau
```

Ce sont les éléments que nous mettrons dans le tableau qui détermineront le type du tableau.
Le lecteur connaissant d'autres langages de programmation de type impératif peut être surpris qu'il ne soit pas obligatoire de préciser la taille du tableau que l'on déclare. Les tableaux sont en effet dynamiques, c'est-à-dire qu'on peut leur ajouter à tout moment, autant d'éléments que l'on veut.
On peut initialiser le tableau à sa création :

```
var fil= new Array("ntic","ig","tlc","2im");
tableau_entiers = [56,12,43,4,9];
```

Il faut noter que la deuxième ligne emprunte une notation uniquement acceptée par des navigateurs récents.
Pour accéder à une valeur particulière du tableau, il faut indiquer sa place dans le tableau, à l'aide de crochets. Attention, le premier élément est numéroté 0.
**Exemple :**

```
var fil= new Array("ntic","ig","tlc","2im");
document.write(fil[2]); // affiche tlc
```

Tout tableau possède une longueur. La longueur du tableau est le numéro de case occupée le plus grand, plus un ou si vous voulez c'est le nombre d'éléments du tableau. Les

tableaux étant dynamiques, l'interpréteur met à jour la longueur du tableau à chaque affectation.
Le programmeur peut obtenir cette longueur en consultant la propriété **length** du tableau considéré : par exemple, on peut faire afficher la longueur du tableau par :

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

Les tableaux sont très utiles car ils permettent de manipuler de manière concise un ensemble de valeurs.
En particulier, il sera pratique de charger et d'afficher les valeurs d'un tableau avec une boucle.

**Exemple1**

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";
   }
}
```

### 7.9.2. Tableaux multidimensionnels

Les tableaux utilisés jusqu'à présent étaient des vecteurs, avec une dimension. Cependant, on a très vite besoin de tables, à deux dimensions. Une limitation importante et décevante de JavaScript est qu'on ne peut décrire simplement des tableaux multidimensionnels. Cependant, il existe des moyens détournés de créer de tels tableaux.
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 une dimension :

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

**Exemple**

| Tarif | T. Small | T. Médium | T. Large |
|---|---|---|---|
| Chemises | 1200 | 1250 | 1300 |
| Polos | 800 | 850 | 900 |
| T-shirts | 500 | 520 | 540 |

```
<html>
<head>
<script language="javascript">
mont = new Array(3);
mont[0] = new Array(3);
mont[1] = new Array(3);
mont[2] = new Array(3);
mont[0][0]="1200"; mont[0][1]="1250"; mont[0][2]="1300";
mont[1][0]="800"; mont[1][1]="850"; mont[1][2]="900";
mont[2][0]="500"; mont[2][1]="520"; mont[2][2]="540";

function affi() {
i = document.form.liste.selectedIndex;
```

```
j= document.form.taille.selectedIndex
document.form.txt.value=mont[i][j];
}
</script>
</head>
<body>

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

</body>
</html>
```

### 7.9.3. Les propriétés de l'objet *Array*

L'objet Array possède deux propriétés caractéristiques: les propriétés *input* et *length*.
Le tableau suivant décrit les propriétés de l'objet *Array*.

| Propriété | Description |
|---|---|
| constructor | Cette propriété contient le constructeur de l'objet Array. |
| Input | Cette propriété permet de faire une recherche dans le tableau à l'aide d'une expression régulière |
| Length | Cette propriété contient le nombre d'éléments du tableau. |
| prototype | Cette propriété permet d'ajouter des propriétés personnalisées à l'objet. |

### 7.9.4. Les méthodes standards de l'objet *Array*

Le tableau suivant décrit les méthodes de l'objet *Array*.

| Méthode | description |
|---|---|
| **concat**(tab1, tab2[, tab3, ...]) | Cette méthode permet de concaténer plusieurs tableaux, c'est-à-dire de créer un tableau à partir des différents tableaux passés en paramètre. |
| **join**(tableau) ou tableau.**join**() | Cette méthode renvoie une chaîne de caractères contenant tous les éléments du tableau. |
| **pop**(tableau) ou tableau.**pop**() | Cette méthode supprime le dernier élément du tableau et retourne sa valeur. |
| tableau.**push**(valeur1[, valeur2, ...]) | Cette méthode ajoute un ou plusieurs éléments au tableau. |
| tableau.**reverse**() | Cette méthode inverse l'ordre des éléments du tableau. |
| tableau.**shift**() | Cette méthode supprime le premier élément du tableau. |
| tableau.**slice**() | Cette méthode renvoie un tableau contenant une partie (extraction) des éléments d'un tableau. |

| | |
|---|---|
| tableau.**splice**() | Cette méthode ajoute/retire des éléments d'un tableau. |
| tableau.**sort**() | Cette méthode permet de trier les éléments d'un tableau. |
| tableau.**unshift**(valeur1[, valeur2, ...]) | Cette méthode renvoie le code source qui a permis de créer l'objet Array. |
| tableau.**tostring**() | Cette méthode renvoie la chaîne de caractères correspond à l'instruction qui a permis de créer l'objet Array. |
| tableau.**unshift**() | Cette méthode permet dajouter un ou plusieurs élément au début du tableau. |
| tableau.**valueof** | Cette méthode retourne tout simplement la valeur de l'objet Array auquel elle fait référence. |

## 7.10. L'objet RegExp

### 7.10.1. Les particularités de l'objet *RegExp*

L'objet *RegExp* est un objet permettant de manipuler des expressions régulières, c'est-à-dire des modèles créés à l'aide de caractères ASCII permettant de manipuler des chaînes de caractères, afin de trouver des portions de la chaîne correspondant au modèle.
La création d'un objet *RegExp* se crée à l'aide d'une simple expression comme suit :

```
Expression = /motif/drapeau
```

Il est également possible de créer un tel objet de manière plus classique à l'aide de son constructeur :

```
Expression = new RegExp("motif","drapeau")
```

Le motif représente l'expression régulière en elle-même tandis que le drapeau (optionnel) permet de préciser le comportement de l'expression régulière :

**g** indique une recherche globale sur la chaîne de caractère et indique une recherche de toutes les occurences.
**i** indique une recherche non sensible à la casse, c'est-à-dire que la recherche se fait indépendamment de l'écriture en majuscule ou minuscule de la chaîne.
**gi** combine les deux comportements précédents.

#### a. Construire une expression régulière

Les expressions régulières permettent de rechercher des occurrences (c'est-à-dire une suite de caractères correspondant à ce que l'on recherche) grâce à une série de caractères spéciaux. L'expression régulière en elle-même est donc une chaîne de caractère contenant des caractères spéciaux et des caractères standards.

Pour rechercher un caractère faisant partie des caractères spéciaux, il suffit de **le faire précéder d'un antislash** (**sauf entre crochets**) :

| Caractère spécial | Echappement |
|---|---|
| \ | \\ |
| . | \. |
| $ | \$ |
| [ | \[ |
| ] | \] |

| | |
|---|---|
| ( | \( |
| ) | \) |
| { | \{ |
| } | \} |
| ^ | \^ |
| ? | \? |
| * | * |
| + | \+ |
| - | \- |

### b. Début et fin de chaîne

Les symboles ^ et $ indiquent respectivement le début et la fin d'une chaîne, et permettent donc de la délimiter.

```
"^debut": chaîne qui commence par "debut"
```

```
"fin$": chaîne qui se termine par "fin"
```

```
"^chaîne$": chaîne qui commence et se termine par "chaîne"
```

```
"abc": chaîne contenant la chaîne "abc"
```

### c. Nombre d'occurences

Les symboles *, + et ?, signifient respectivement "zéro ou plusieurs", "au moins un", "un ou aucun", et permettent de donner une notion de quantité.

```
"abc+": chaîne qui contient "ab" suivie de un ou plusieurs "c" ("abc",
"abcc", etc.)
"abc*": chaîne qui contient "ab" suivie de zéro ou plusieurs "c" ("ab",
"abc", etc.)
"abc?": chaîne qui contient "ab" suivie de zéro ou un "c" ("ab" ou "abc"
uniquement)
"^abc+": chaîne commençant par "ab" suivie de un ou plusieurs "c" ("abc",
"abcc", etc.)
```

### d. Accolades

Les accolades {X,Y} permettent de donner des limites précises de nombre d'occurences.

```
"abc{2}": chaîne qui contient "ab" suivie de deux "c" ("abcc")
"abc{2,}": chaîne qui contient "ab" suivie de deux "c" ou plus ("abcc"
etc..)
"abc{2,4}": chaîne qui contient "ab" suivie 2, 3 ou 4 "c" ("abcc" ..
"abcccc")
```

Il est à noter que le premier nombre de la limite est obligatoire ("{0,2}", mais pas "{,2}"). Les symboles vu précedemment ('*', '+', et '?') sont équivalents à "{0,}", "{1,}", et "{0,1}".

### e. Parenthèses capturantes

Les parenthèses ( ) permettent de représenter une séquence de caractères et de capturer le résultat. Les occurences correspondant au motif entre parenthèses sont accessibles via la méthode *exec()* de l'objet RegExp ou bien les méthodes *search()*, *match()* et *replace()* de l'objet String.

```
"a(bc)+": chaîne qui contient "a" suivie de au moins
une occurence de la chaîne "bc"
```

### f. barre verticale

La barre verticale | se comporte en tant qu'opérateur OU

```
"(un|le)": chaîne qui contient "un" ou "le"

"(un|le) chien": chaîne qui correspond à
 "un chien" ou "le chien"
```

```
"nabeledi\.((net)|(com)|(org))":
chaîne qui correspond à  :
"nabeledi.net"
"nabeledi.com"
"nabeledi.org"
```

### g. Caractère quelconque

Le point (.) indique n'importe une occurence de n'importe quel caractère.

```
"^.{3}$": chaîne qui contient 3 caractères
".*": Tous les caractères
```

### h. Liste de caractères

Les crochets *[ ]* définissent une liste de caractères autorisés (ou interdits). Le signe - permet quand à lui de définir un intervalle. Le caractère ^ après le premier crochet indique quand à lui une interdiction.

```
"[abc]": chaîne qui contient un "a", un "b", ou un "c".
"[a-z]": chaîne qui contient un caractère compris entre "a" et "z".
"[^a-zA-Z]": chaîne qui ne commence pas par une lettre.
```

En effet entre crochets, chaque caractère représente ce qu'il est. Pour représenter un ] il faut le mettre en premier (ou après un ^ si c'est une interdiction). Etant donné que le signe - sert à définir un intervalle, il est nécessaire de commencer ou de terminer par ce caractère lorsque l'on veut indiquer qu'il fait partie des caractères autorisés :

```
"[-ag]": chaîne qui contient un moins (-), un "a", ou un "g"
"[a-g]": chaîne qui contient un caractère compris entre "a" et "g"
"[\+?{}.]": chaîne qui contient un de ces six caractères
"[]-]": chaîne qui contient le caractère "]" ou le caractère "-"
```

### i. Caractères spéciaux

Il existe des caractères spéciaux (précédés d'une barre oblique inverse) représentant des types de caractères

| Caractère spécial | Utilité |
|---|---|
| \b | Permet de capturer une coupure de mot, c'est-à-dire des caractères situés au |

| | |
|---|---|
| | tout début ou à la fin d'un mot. Par exemple "he\b" permet de capturer "CommentCaMarche" mais pas "chenil". De la même façon "\bCo" permet de capturer "CommentCaMarche" mais pas "DéCor". |
| \B | Permet de capturer les caractères non précédés ou suivis d'une coupure de mot, c'est-à-dire des caractères situés au milieu d'un mot. Par exemple "ment\B" permet de capturer "CommentCaMarche" mais pas "Comment Ca Marche". |
| \c*Caractère* | Permet de capturer un caractère de contrôle (correspondant à la combinaison *Ctrl+Caractère*. Par exemple "\cC" permet de capturer la séquence *Ctrl+c*. |
| \d | Permet de capturer un caractère numérique. *\d* est ainsi équivalent à *[0-9]*. |
| \f | Permet de capturer un saut de page. |
| \n | Permet de capturer un saut de ligne. |
| \r | Permet de capturer un retour chariot. |
| \s | Permet de capturer un "caractère blanc" (espace, retour chariot, tabulation, saut de ligne, saut de page). |
| \S | Permet de capturer un "caractère non blanc" (tous les caractères sauf espace, retour chariot, tabulation, saut de ligne, saut de page). |
| \t | Permet de capturer une tabulation horizontale. |
| \v | Permet de capturer une tabulation verticale. |
| \w | Permet de capturer un caractère alphanumérique (y compris le caractère _). *\w* est ainsi équivalent à *[a-zA-Z0-9_]*. |
| \w | Permet de capturer un caractère non alphanumérique. *\W* est ainsi équivalent à *[^a-zA-Z0-9_]*. |
| \o*Nombre* | Permet de capturer un nombre en base octale (base 8). |
| \x*Nombre* | Permet de capturer un nombre en base hexadécimale (base 16). |

## j. Tableau récapitulatif des caractères spéciaux

| Caractère | Utilité |
|---|---|
| \ | Le caractère antislash représente lui-même ou le caractère spécial qui le suit. |
| [] | Les crochets définissent une liste de caractères. |
| () | Les parenthèses définissent un élément composé de l'expression régulière qu'elle contient. |
| {} | Les accolades lorsqu'elles contiennent un ou plusieurs chiffres séparés par des virgules représentent le nombre d'occurences de l'élément les précédant (par exemple *p{2,5}* correspond à *ppp*, *pppp* ou *ppppp* |
| - | Un moins entre deux caractères dans une liste représente un intervalle (par exemple *[a-d]* représente *[abcd]*). |
| . | Le caractère point représente un caractère quelconque. |
| * | Le caractère astérisque indique un nombre d'occurences indéterminé (y compris aucune) de l'élément le précédant. |

| | |
|---|---|
| + | Le caractère plus indique une ou plusieurs occurences de l'élément le précédant. |
| ? | Le caractère "point d'interrogation" indique une occurence éventuelle (0 ou 1) de l'élément le précédant. |
| \| | La barre verticale signifie l'occurrence de l'élément situé à sa gauche ou de celui situé à sa droite (*lard*\|*cochon*) |
| ^ | Placé en début d'expression il signifie "chaîne commençant par .. "<br>Utilisé entre crochet, immédiatement après le crochet ouvrant, il signifie "ne contenant pas les caractères suivants... |
| [abc] | Permet de rechercher les caractères compris entre les crochets. |
| [^abc] | Permet de rechercher tous les caractères sauf ceux compris entre les crochets. |
| $ | Placé en fin d'expression il signifie "chaîne finissant par ... " |

### 7.10.2. Les propriétés de l'objet RegExp

Le résultat d'une expression régulière est stockée dans l'objet RegExp. Les propriétés de l'objet RegExp contiennent des chaînes correspondant à la dernière occurence trouvée.
La syntaxe pour manipuler ces données est la suivante :

```
RegExp.propriété
```

| Propriété | Description |
|---|---|
| $_ | Propriété correspondant à la propriété *input*. |
| $* | Propriété correspondant à la propriété *multiline*. |
| $& | Propriété correspondant à la propriété *LastMatch*. |
| $+ | Propriété correspondant à la propriété *LastParen*. |
| $` | Propriété correspondant à la propriété *LeftContext*. |
| $' | Propriété correspondant à la propriété *RightContext*. |
| global | Propriété booléenne indiquant si la recherche est globale (*true*) ou non (*false*). |
| ignoreCase | Propriété booléenne indiquant si la recherche est sensible à la casse (*true*) ou non (*false*). |
| input | Indique la chaîne d'entrée sur laquelle la recherche est réalisée. |
| lastIndex | Indique la position à laquelle la recherche suivante va se faire. |
| lastMatch | Contient la dernière occurence trouvée. |
| lastParen | Contient la dernière occurence correspondant à un motif entre parenthèses. |
| leftContext | Contient la chaîne situé à gauche de l'occurence trouvée. |
| multiline | Propriété booléenne indiquant si la recherche porte sur plusieurs lignes (*true*) ou non (*false*). |
| rightContext | Contient la chaîne situé à droite de l'occurence trouvée. |
| source | Contient le motif de l'expression régulière. |

### 7.10.3. Les méthodes de l'objet RegExp

Les méthodes de l'objet RegExp permettent d'appliquer l'expression régulière à une chaîne de caractères.
Le tableau suivant décrit les méthodes de l'objet RegExp :

| **Méthode** | **Description** |
|---|---|
| **Expression.compile**("chaine"); | Permet de redéfinir une nouvelle expression régulière. |
| **Expression.exec**("chaine"); | Effectue une recherche sur la chaîne de caractère avec l'expression régulière définie. Cette méthode retourne un tableau contenant les occurences trouvées. |
| **Expression.test**("chaine"); | Teste une chaîne de caractère avec l'expression régulière. Cette méthode retourne *True* si la recherche est fructueuse, *false* dans le cas contraire. |

# VIII. LES EVENEMENTS

## 8.1. Généralités

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.

## 8.2. Les événements

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. 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 champ 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 |

## 8.3. Les gestionnaires d'événements

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

**syntaxe**

```
onévénement="fonction()";
```

**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.

### 8.3.1. onclick

Evénement classique en informatique, le clic de la souris.

**Exemple :**

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

```
</FORM>
```

### 8.3.2. 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 au CUP");
}
function au_revoir() {
alert("Au revoir et à bientot");
}
</SCRIPT>
</HEAD>
<BODY onLoad='bienvenue()' onUnload='au_revoir()'>
Html normal
</BODY>
</HTML>
```

### 8.3.3. 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.

**Exemple**

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

**Exemple**

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

```
</BODY>
</HTML>
```

### 8.3.4. 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".

### 8.3.5. 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>
```

### 8.3.6. 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.

### 8.3.7. onselect

Cet événement se produit lorsque l'utilisateur a sélectionné tout ou partie d'une zone de texte dans une zone de type text ou textarea.

### 8.3.8. Gestionnaires d'événement 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 sélection | Blur, onChange, onFocus |
| Bouton Submit | onClick |
| Bouton Reset | onClick |

# Sources

**http://www.editeurjavascript.com/cours/**
**http://www.ccim.be/ccim328/js/index.htm**
**http://www.commentcamarche.net/javascript/**
http://fr.selfhtml.org/javascript/
http://www.toutjavascript.com/