MCours.com
Le N°1 du cours et exercices sur Internet

# LE JAVASCRIPT

## - COURS COMPLET -

# NOTE DE L'AUTEUR

Ce cours ne possède pas de copyright, mais l'auteur souhaite réguler son utilisation. Celle-ci est illimitée dans le cadre privé tant que le document conserve son état d'origine. L'auteur autorise la reproduction et la présentation publique à 3 conditions :

- Le document ne doit subir aucune modification.
- Le document doit être présenté sous le nom de son auteur.
- L'auteur doit être averti de toute utilisation ou publication collective.

# L'AUTEUR

L'auteur est un lycéen de 19 ans, en première année d'école d'ingénieur INSA. Ses loisirs sont en partie occupés par la programmation. Les langages qu'il connaît sont le Javascript, le HTML, le PHP/SQL, le C/C++ et le Pascal. A ses heures perdues, il aime aussi écrire, ce qui lui a donné l'idée de réaliser des cours de programmation.

Pour le contacter, utiliser l'adresse e-mail suivante : michael.muraz@insa-lyon.fr. Il est aussi disponible sur MSN Messenger : banzaichico@hotmail.com.

# TABLE DES MATIERES

# 1 AVANT-PROPOS

Ce cours s'adresse à tout programmeur, du débutant ayant soif d'apprendre à l'expert ayant besoin de se remémorer quelque détail, en passant par le programmeur qui souhaite découvrir un langage ou se perfectionner... Son but n'est pas de former quelqu'un au Javascript, car ce serait se vanter. Non, son but est de donner des bases, que le programmeur averti saura compléter par d'autres lectures.

Il – le cours - n'a pas vocation à être exhaustif. Il est plutôt évolutif, c'est-à-dire qu'il n'existe que pour être amélioré. L'auteur n'attend que les remarques, critiques, suggestions… des lecteurs pour améliorer ce document et l'enrichir.

L'auteur ne s'engage pas à fournir un cours parfait. Il peut – et il doit – y avoir des erreurs. Il ne prétend pas posséder le cours sans fautes. Les remarques à ce sujet sont aussi les bienvenues.

Enfin, si un lecteur rencontre un problème, s'il ne comprend pas un point, qu'il n'hésite pas à piocher dans la liste des liens qui sont fournis, et même, si un passage ne semble vraiment pas clair, écrire à l'auteur. Ses coordonnées figurent au début du document, avant la table des matières. Cependant, ne pas abuser de ce service, l'auteur n'est pas une aide en ligne, il a aussi une vie professionnelle ne peut passer son temps à répondre à des questions. Si la réponse se fait attendre, patienter encore. Peut-être n'a-t-il pas le temps ou peut-être n'a-t-il pas encore trouvé la solution…

Merci d'avoir lu cet avant-propos. Bonne lecture et bienvenue dans l'univers de Javascript !

# 2 GENERALITES

## 2.1 Le langage Javascript

Le langage Javascript a été mis au point par Netscape en 1995. Microsoft ayant sorti son propre langage quelques temps après, un standard a été créé, le Javascript[1]. Actuellement, la version du langage est le Javascript 1.5[2]. Ce langage est interprété, c'est-à-dire qu'il n'est pas compilé en langage machine avant exécution, comme le Java ou le C++. Le Javascript est intégré au code HTML, il vous faudra donc des bases assez solides en HTML. Si ce n'est pas le cas, il est conseillé de consulter un cours HTML de toute urgence. Il s'agit, a priori, du premier langage de script créé pour le web. Il permet d'exécuter des commandes du côté utilisateur, donc au niveau du navigateur et non du serveur - comme le PHP. Son principal inconvénient est le fait qu'il ne dispose pas de débogueur, la détection des erreurs en devient fastidieuse. Ensuite, le code est accessible, puisque contenu dans la page envoyée au navigateur. Cela peut être gênant et l'auteur ne veut pas forcément dévoiler ses sources.

## 2.2 Contraintes logicielles

Pour programmer en Javascript, il faut un navigateur web assez récent et un éditeur de texte, le bloc-notes de Windows est suffisant. Une connexion Internet est souhaitable. Au niveau de l'éditeur de texte, il est conseillé d'utiliser un éditeur un peu plus évolué. Le mieux serait bien entendu *Dreamweaver MX*[3], pour ceux qui ont les moyens, bien que ce soit surtout utile lors de la création un site web. Du côté des éditeurs gratuits, *Editplus2*[4] est performant. C'est un éditeur tous langages qui propose une coloration syntaxique. Bien entendu, il est possible d'utiliser un autre éditeur.

## 2.3 Détails techniques

Par convention, l'abréviation JS, utilisée souvent tout au long de ce cours, désigne Javascript. Chaque section concerne un point du langage qui peut être pris comme une leçon. A chaque fin de section, un exemple concret est présenté – quand c'est possible – et un ou plusieurs exercice(s). Ces exercices sont corrigés dans le dossier Solutions. Ce cours est agrémenté de nombreux exemples. Lorsque ces derniers ont un résultat graphique, ils sont dans le dossier Exemples.

[1] Pour plus de précisions, voir le site http://www.commentcamarche.net.
[2] Ne fonctionne qu'avec Nestcape Navigator6.X et Internet Explorer 6.X
[3] Edité par *Macromédia*
[4] Edité par *ES-Computing*, à télécharger sur http://www.telecharger.com.

# 3 LE LANGAGE

## 3.1 Incorporation du code

Comme son nom l'indique, il s'agit d'un langage de script. Le code s'intègre donc dans la page HTML avec les balises `<script>`. Il existe deux façons d'intégrer votre code. La première consiste à le placer entre les balises, tout simplement :

```
<script language = "Javascript">code </script>
```

**Exemple 3.1 – Balise <script>**

```
<script language = "Javascript">
var mavariable = 12 ;
document.write (typeof(mavariable)) ;
</script>
```

Il est aussi possible de placer le code dans un fichier Javascript, comportant l'extension « .js ». Ensuite, il faut l'insérer dans la page HTML avec la balise `<script>`. Ne pas oublier qu'il est préférable de placer le fichier JS dans le même dossier que la page.

```
<script src = "chemin_fichier"></script>
```

**Exemple 3.2 – Inclusion de fichier « .js »**

```
<script src = "date.js"> </script>
```

## 3.2 Spécificités du langage

La première chose qu'il faut noter en Javascript, comme dans le C, est que chaque instruction se termine par un point-virgule `';'`. Il est possible de mettre plusieurs instructions sur la même ligne, puisque l'interpréteur ignore les sauts de lignes. Vous pouvez aussi insérer des blancs où vous voulez – excepté dans des noms de variables ou dans des expressions – cela ne change pas le code.

**Exemple 3.3 - Spécificités**

```
var mavariable = 12 ;
document.write ( typeof(mavariable)) ;
```

Il est utile de commenter son code. Les commentaires se font à l'aide de '//', tout le texte suivant le double slash jusqu'à la fin de la ligne est ignoré par l'interpréteur.

**Exemple 3.4 – Commentaires //**

```
var mavariable = 12 ; //commentaire
document.write ( typeof(mavariable) ) ;
```

Il est aussi possible de mettre des commentaires au milieu d'un ligne, ou sur plusieurs ligne, en les encadrant avec « /* » et « */ ». Ces commentaires sont très utiles et peuvent très bien être utilisés comme les commentaires de fin de ligne.

**Exemple 3.5 – Commentaires /* */**

```
var mavariable = 12 ; /*commentaire sur
plusieurs lignes*/
document.write ( typeof(mavariable) ) ;
```

# 4 LES STRUCTURES DE DONNEES

## 4.1 Les données constantes

Le JS fournit quatre types de constantes déjà définies : les constantes numériques, en notation décimale (75.2) ou flottante, c'est-à-dire scientifique (7.52E-1) ; les constantes booléennes : `true` et `false` ; les chaînes de caractères, encadrées de guillemets ou d'apostrophes (`''`) ; la constante `null` qui signifie « rien », « sans valeur », et qui est attribuée au variables non définies.

## 4.2 Les variables

### 4.2.1 Types de variables

Un nom de variable doit commencer par une lettre (alphabet ASCII) ou le signe « _ » et se composer de lettres, de chiffres et des caractères « _ » et « `$` » (à l'exclusion du blanc). Le nombre de caractères n'est pas précisé. Javascript est sensible à la casse (majuscules et minuscules). Cela signifie que « `MaVariable` » n'est pas équivalent à « `mavariable` ».

Il existe 5 types de variables en Javascript. Elles seront toutes décrites au cours de ce document. En voici la liste :

- Les nombres : `number`
- Les chaînes de caractères : `string`
- Les booléens[5] : `boolean`
- Les objets : `object`
- Les fonctions : `function`

Les fonctions seront abordés un peu plus loin, mais il est utile de connaître la fonction suivante : `typeof()`. Elle retourne[6] le type d'une variable. Voici un exemple d'utilisation :

**Exemple 4.1 – Fonction typeof()**

```
var mavariable = 12 ;
document.write (typeof(mavariable)) ;
```

[5] Variable ne pouvant avoir que deux valeur : true (1) et false (0).

Si la variable n'est pas affectée d'une valeur, elle recevra alors le type `undefined`, comme le montre l'exemple ci-dessous.

**Exemple 4.2 – Type d'une variable sans valeur**

```
document.write (typeof(mavariable)) ;
```

### 4.2.2 Déclaration et affectation

On distinguera ici la déclaration et l'affectation. La première consiste à donner un nom à la variable alors que la seconde consiste à donner une valeur à la variable. La différence est qu'une variable peut-être affectée d'une valeur plusieurs fois alors qu'elle ne peut être définie qu'une seule fois. L'exemple suivant illustre une définition sans affectation.

**Exemple 4.3 – Définition de variable**

```
var Numero;
```

Dans les exemples qui suivent, par facilité, sont regroupées l'affectation et la déclaration. Les variables peuvent être déclarées de deux façons. La première est la façon explicite, avec le mot-clé « `var` ».

**Exemple 4.4 – Affectation explicite de variable**

```
var Numero = 1 ;
```

Il est aussi possible de la faire de façon implicite. On écrit directement le nom de la variable suivi de la valeur que l'on lui attribue.

**Exemple 4.5 – Affectation implicite de variable**

```
Numero = 1 ;
```

### 4.2.3 Manipulations sur les chaînes de caractères

Javascript convertit automatiquement les entiers en chaîne de caractères. Pour les puristes, il s'agit du transtypage automatique. Il est ainsi possible de concaténer des entiers avec des chaînes de caractères.

[6] C'est-à-dire qu'elle « donne » le type de la variable. Ce point sera abordé ultérieurement.

Dans la fonction d'écriture dans le document courant `document.write()`, les données peuvent être séparés par des '`,`' ou des '`+`'.

Certains caractères spéciaux peuvent être insérés dans les chaînes de caractères : le retour arrière (`\b`), le saut de page (`\f`), le saut de ligne (`\n`), l'entrée (`\r`), la tabulation (`&`) et l'apostrophe (`\'`).

On peut insérer du code HTML dans les chaînes de caractères. Ces dernières seront interprétées comme de vraies balises.

Tout ceci sera expliqué plus en détails dans un chapitre réservé à l'objet `String`.

### 4.2.4 Mots réservés

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

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

**TAB. 1 – Mots réservés**

### 4.2.5 Portée des variables

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

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

# 5 OPERATEURS

Comme tout langage informatique, JS possède des opérateurs pour effectuer les calculs. Leur présentation est rapide, surtout pour les plus simples. Dans les exemples, la valeur initiale de `x` sera toujours égale à `11`, et celle de `y` sera égale à `5`.

## 5.1 Opérateurs arithmétiques

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

**TAB. 2 – Opérateurs arithmétiques**

## 5.2 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` |

**TAB. 3 – Opérateurs de comparaison**

Ces opérateurs seront utilisés dans les boucles conditionnelles[7]. Le résultat s'exprime alors en valeur booléenne.

## 5.3 Opérateurs associatifs

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

**TAB. 4 – Opérateurs associatifs**

Ces opérateurs permettent un raccourci d'écriture, sans pour autant changer le résultat. Certains permettent par exemple une incrémentation ou une décrémentation autre que 1.

## 5.4 Opérateurs logiques

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

**TAB. 5 – Opérateurs logiques**

On utilisera ces opérateurs dans les boucles conditionnelles principalement. Chaque condition correspondant à une expression avec un opérateur de comparaison. Ces opérateurs fonctionnent comme un ET logique et un OU logique[8].

[7] Voir pour cela la section concernant les structures de contrôle.
[8] Il s'agit là d'un OU non exclusif.

## 5.5 Opérateurs d’incrémentation

| Signe | Description | Exemple | Signification | Résultat |
|---|---|---|---|---|
| `x++` | incrémentation | `y = x++` | y = x + 1 | `6` |
| `x--` | décrémentation | `y= x--` | y = x - 1 | `4` |

**TAB. 6 – Opérateurs d’incrémentation**

# 6 FONCTIONS

## 6.1 Définition

C'est un groupe d'instruction prédéfini et exécuté lorsqu'il est appelé et que l'on peut appeler plusieurs fois. En Javascript, il existe deux types de fonctions : les fonctions propres à Javascript, appelées méthodes[9]. Elles sont associées à un objet en particulier. En suite, il y a les fonctions définies par le programmeur. Ce sont celles qui seront détaillées ci-après.

## 6.2 Déclaration

Pour déclarer ou définir une fonction, on utilise le mot-clé `function`. La syntaxe d'une déclaration de fonction est la suivante :

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

Le nom d'une fonction suit les mêmes règles que celui d'une variable[10]. Chaque nom de fonction doit être unique dans un même script. Les parenthèses sont obligatoires même si il n'y a pas d'arguments, puisque Javascript reconnaît une fonction grâce à elles, par rapport aux variables dites « normales ».

## 6.3 Fonctions dans l'en-tête

Il est plus prudent de placer les déclarations de fonctions dans l'en-tête `<head>…</head>` pour qu 'elles soient prises en compte par l'interpréteur avant leur exécution dans le corps de la page `<body>…</body>`.

## 6.4 Appel de fonction

Une fonction doit être appelée pour être exécutée, c'est-à-dire qu'il faut écrire son nom pour que l'interpréteur exécute les instructions contenues dans le corps de la fonction. Un appel s'effectue de la manière suivante :

[9] Ces fonctions particulières seront l'objet d'une sous partie de la section abordant la notion objet.

```
nom_de_la_fonction();
```

Il faut que la fonction ait été définie avant l'appel, étant donné que l'interpréteur lit le script de haut en bas.

## 6.5 Manipulation de valeurs à l'aide de fonctions

### 6.5.1 Passage de paramètre

Il est possible passer des valeurs à une fonction. On parle alors de paramètres. Pour passer un paramètre à une fonction, on fournit un nom d'une variable dans la déclaration de la fonction. Il faut avoir auparavant préciser un nom de paramètre dans la déclaration de la fonction. Dans l'exemple suivant, il s'agit d'une fonction affichant le texte dans la page à laquelle on fournit le texte à afficher.

**Exemple 6.1 – Paramètre de fonction**

```
function Exemple(texte) { /*définition avec un
paramètre*/
document.write(texte);
}
Exemple("Salut à tous"); /* appel avec un paramètre*/
```

Il est possible de passer plusieurs paramètres à une fonction, en séparant les paramètres par des virgules. Cela s'indique ainsi :

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

L'exemple suivant montre une fonction prenant un nombre en paramètre et retournant son carré. Elle est appelée via une variable.

[10] Voir la section concernant les structures de données.

**Exemple 6.2 – Paramètres multiples**

```
function cube(nombre) {
       y = nombre*nombre*nombre;
       return y;     //retour de valeur
       }
x = cube(5); //appel avec paramètre
document.write(x); //résultat
```

### 6.5.2 Retour de valeur

Une fonction peut retourner une valeur. Il suffit alors de placer le mot-clé `return` suivi de la valeur ou de la variable à retourner. Par exemple :

**Exemple 6.3 – Retour de valeur**

```
function cube(nombre) { //Définition de la fonction
var c = nombre*nombre*nombre ;
return c;     //Retour du cube du paramètre
}
var x = cube(5) ; // appel avec paramètre
document.write (x) ; //affichage
```

## 6.6 Portée des variables

Une variable déclarée dans une fonction par le mot-clé `var` aura une portée limitée à cette seule fonction. On l'appelle donc variable locale et ne pourra être utilisé dans le reste du script. Voici un exemple :

**Exemple 6.4 – Variable locale déclarée dans une fonction**

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

Si la variable est déclarée sans utiliser le mot var, sa portée alors sera globale.

**Exemple 6.5 – Variable globale déclarée dans une fonction**

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

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.

**Exemple 6.6 - Variable globale déclarée hors d'une fonction**

```
var cube=1
function cube(nombre) {
var cube = nombre*nombre*nombre ;
}
```

# 7 STRUCTURES DE CONTROLE

## 7.1 Structures algorithmiques

Quelque soit le langage informatique utilisé[11], un programme informatique utilise 3 catégories principales d'instructions. Il y a les instructions séquentielles, les instructions alternatives - ou conditionnelles – et les instructions itératives - ou répétitives -. Chacune de ces structures possède une forme sous JS et cette dernière sera indiquée ci-après.

## 7.2 Structure séquentielle

C'est une suite d'instructions exécutées dans l'ordre où elles sont écrites, l'une après l'autre. L'exemple le plus fréquent est la fonction.

| Algorithme | Code JS |
|---|---|
| DEBUT<br>Instruction1<br>Instruction2<br>FIN | `{`<br>`Instruction1 ;`<br>`Instruction2 ;`<br>`}` |

**TAB. 7 – Structure séquentielle**

## 7.3 Structures conditionnelles

### 7.3.1 Expression if ... else

Il s'agit de tester une condition et d'effectuer une série d'instructions si la condition est vraie et si elle est fausse, effectuer une autre série d'instructions.

[11] Ce chapitre, excepté pour quelques expressions, est valable pour de nombreux langages, notamment le C++.

| Algorithme | Code JS |
|---|---|
| SI condition<br>ALORS Instructions1<br>SINON Instructions2<br>FINSI | `if (condition) {`<br>`  Instruction1`<br>`  }`<br>`else {`<br>`  Instruction2`<br>`  }` |

**TAB. 8 – Expression if ... else**

L'exemple suivant demande l'âge de l'internaute à l'aide de la méthode `prompt()` et affiche une boite de message à l'aide de la méthode `alert()`.

**Exemple 7.1 – Expression if ... else**

```
x = prompt ("votre age?","age");
if ( x < 40) {
       alert ('vous êtes jeune') ;
       }
else  {
       alert ('vous êtes vieux') ;
       }
```

### 7.3.2 Expression ? :

Il s'agit d'un moyen rapide de tester une condition et de d'exécuter une instruction selon son résultat. Dans cette expression, si la condition est vérifiée, l'instruction 1 est effectuée, sinon, l'instruction 2 est effectuée. Il est utile de préciser que toute l'expression doit se trouver sur la même ligne.

| Algorithme | Code JS |
|---|---|
| SI condition<br>ALORS Instructions1<br>SINON Instructions2<br>FINSI | `(condition) ? instruction 1 :`<br>`instruction 2` |

**TAB. 9 – Expression ? :**

L'exemple qui suit utilise cette expression pour faire le même programme que le précédent. Le code s'en trouve alors largement raccourci. Cependant, il n'est pas possible d'exécuter plusieurs instructions avec cette expression, ce qui la rend peu utile.

**Exemple 7.2 – Expression ? :**

```
x = prompt ("votre age?","age");
age = (x < 40)? "jeune" : "vieux";
alert ('vous êtes ' + age) ;
```

## 7.4 Structures itératives

### 7.4.1 Expression for

Elle permet de répéter des instructions en s'arrêtant à un certain nombre d'itérations (boucles). L'incrémentation indiquée ci-dessous peut être remplacée par une décrémentation.

| Algorithme | Code JS |
|---|---|
| POUR i = valeur initiale A i = valeur finale<br>REPETER instructions<br>FIN POUR | `For (val initiale ; condition ; incrémentation) {`<br>`Instructions`<br>`}` |

**TAB. 10 – Expression for**

Un exemple de l'utilisation des boucles est l'écriture d'une suite de nombres. Il faudra alors réutiliser la variable de la boucle, comme ceci :

**Exemple 7.3 – Expression for**

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

### 7.4.2 Expression while

Elle permet de répéter des instructions tant qu'une condition est vraie. Celle-ci est indiquée au début de la boucle, puis est testée à chaque autre boucle. Dès qu'elle devient fausse, l'exécution des boucles s'arrête.

| Algorithme | Code JS |
|---|---|
| TANT QUE (condition vraie)<br>REPETER instructions<br>FIN TANT QUE | `while (condition) {`<br>`instructions`<br>`}` |

**TAB. 11 – Expression while**

L'affichage d'une suite de nombres peut se coder aussi avec cette expression. Cela donne l'exemple suivant :

**Exemple 7.4 – Expression while**

```
i = 0;
while (i < 10) {
document.write(i + " ");
i++;
}
```

## 7.5 Interruption de boucle

### 7.5.1 Instruction break

Elle permet de mettre fin prématurément à une instruction itérative `while` ou `for`. L'interpréteur passe aux instructions suivant la boucle. Voici un exemple :

**Exemple 7.5 – Instruction break**

```
for (i = 0; i < 10; i++) {
if (i == 5) {
break;
}
document.write(i + " ");
}
```

### 7.5.2 Instruction continue

Elle permet de mettre fin à une itération et de passer à la boucle suivante. Les instructions qui suivent – à l'intérieur de la boucle – ne sont pas exécutées, pour cette itération seulement. Par exemple :

**Exemple 7.6 – Instruction continue**

```
for (i = 0; i < 10; i++) {
if (i == 5) {
continue;
}
document.write(i + " ");
}
```

## 7.6 Exercice

Dans cet exercice, il faut afficher tous les nombres pairs compris entre 0 et 20 inclus, dans l'ordre décroissant. Il ne faudra pas afficher 10 mais 100, et il ne faudra pas non plus afficher 14.

# 8 BOITES DE MESSAGE

## 8.1 Généralités

Voici les boîtes de dialogue, qui sont des méthodes de l'objet `window`, qui sera abordé plus tard, dans la partie objet du langage. Bien que la programmation objet n'aie pas encore été expliquée, il semble important d'évoquer ces boites de dialogue, qui s'avèrent utile en JS. Il en existe 3, qui seront détaillées l'une après l'autre.

## 8.2 Alert

Cette méthode a été utilisée dans certains exemples précédemment. La boîte qui s'affiche ne comporte qu'un texte informatif et un bouton « OK ». La syntaxe est la suivante :

```
alert (paramètres) ;
```

Les paramètres de cette méthode peuvent être plusieurs objets : une chaîne de caractère entre guillemets, une variable ou un enchaînement des deux, séparés par l'opérateur « + » ou « , ». Par exemple :

**Exemple 8.1 – La méthode alert()**

```
x = 5 ;
alert ('Le nombre est ' + x) ;
```

## 8.3 Prompt

Il s'agit d'une boite d'invite, composée d'un texte, d'une zone de texte, d'un bouton « OK » et d'un bouton « Annuler ». Elle permet de récupérer des valeurs entrées par l'internaute. Elle s'utilise ainsi :

```
variable = prompt ("texte", "valeur");
```

La méthode retourne la valeur du champ si le bouton « OK » est choisi dans le cas inverse (bouton « Annuler »), la variable reçoit la valeur « NULL ». Les paramètres sont le texte à afficher dans la boite de message et la valeur par défaut à afficher dans la boite d'invite.

**Exemple 8.2 – La méthode prompt()**

```
x = prompt ('Votre prénom ?','prénom') ;
alert (x) ;
```

## 8.4 Confirm

Il s'agit d'une boite de confirmation, composée d'un texte, d'un bouton « OK » et d'un bouton « Annuler ». Comme son nom l'indique, elle permet de demander confirmation à l'utilisateur. En voici la syntaxe :

```
variable = confirm ("texte");
```

La méthode retourne `true` si on clique sur « OK », et `false` si on clique sur « Annuler ». Il faut entrer en paramètre le texte à afficher dans la boite de message ; qui indique à l'utilisateur ce qu'il doit confirmer.

**Exemple 8.3 – La méthode confirm()**

```
x = prompt ('Votre prénom ?','prénom') ;
y = confirm ('Votre prénom est bien ' + x + ' ?') ;
```

## 8.5 Exercice

Le but est de récupérer et afficher 3 informations de l'utilisateur à l'écran. Il faut demander à l'utilisateur son login. S'il choisit annuler, on affiche la page blanche. Ensuite, on compare le login entré avec celui attendu : s'il est différent, on redemande le login, sinon, on continue le traitement. On demande ensuite le password. On vérifie qu'il est correct et on continue. S'il ne l'est pas, on redemande 2 fois ce password. S'il est toujours incorrect au 3ème essai, on affiche un message d'erreur à l'écran, et le traitement est fini.

Enfin, on demande le prénom de l'utilisateur, puis on demande vérification à l'internaute. Si le prénom est correct, on l'affiche à l'écran. Pour afficher à l'écran, utiliser la méthode `document.write()`, en passant la chaîne à afficher en paramètre.

Cet exercice utilisera les connaissances au niveau des fonctions, des variables, des structures algorithmiques, et bien entendu des boites de dialogue. Dans la solution, le login est « `prof` » et le password est « `abcd` ».

# 9 NOTION OBJET

Bien que ce ne soit pas le but de cette section, il semble important de rappeler le concept objet, notamment pour ceux qui ne le connaissent pas. Il ne s'agit pas d'un cours magistral, mais juste d'un résumé à des fins d'utilisation en JS.

## 9.1 Concept objet

Pour l'instant n'ont été abordées que des variables, avec une valeur, indépendantes les unes des autres. Maintenant, on parlera d'objet. Un objet – en général - possède des caractéristiques et des compétences. On peut voir ces caractéristiques et utiliser les compétences. En informatique, un objet possède des variables et des fonctions, qui permettent de décrire son comportement. Ces variables sont appelées propriétés et ces fonctions sont appelées méthodes. De cette façon, il est possible de voir les propriétés et d'utiliser les méthodes.

Les méthodes et les variables d'un même type d'objet sont regroupés dans une classe, c'est-à-dire une fonction appelée constructeur. Celle-ci portera le nom du type d'objet. Il sera alors possible de créer des objets de ce type, ce qui est le sujet de la section suivante.

## 9.2 Création d'un objet

Un objet est créé en utilisant une fonction spéciale, précédée du mot `new`. Cette fonction est appelée constructeur, et porte le nom de la classe. Un objet est appelé instance de la classe. Voici la syntaxe :

```
var objet = new classe ();
```

L'exemple suivant montre la création d'un objet `Array`, dont la classe est implémentée dans le JS.

**Exemple 9.1 – Création d'une instance de classe**

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

## 9.3 Accès aux propriétés et méthodes

On accède aux propriétés – et aux méthodes – d'un objet en accolant l'objet et sa propriété par l'opérateur « . ». Cela montre que la variable ou la fonction « appartient » à l'objet. Cela s'écrit de la façon suivante :

```
objet.propriété
objet.méthode()
```

L'exemple qui suit n'est qu'une extrapolation du concept objet en l'appliquant à la vie courante. Cependant, il est tout à fait possible de créer un objet et une classe sur ce modèle.

**Exemple 9.2 – Exemple de propriétés**

```
Rémi = new Homme () ;
Rémi.yeux = "bleus" ;
Rémi.cheveux = "bruns" ;
```

L'exemple ci-dessous est l'utilisation d'une méthode de l'objet document. Ce dernier est intégré au langage et il sera décrit dans une section ultérieure. La méthode présentée ici permet d'écrire dans la page en cours.

**Exemple 9.3 – Une méthode de l'objet document**

```
document.write ("Hello world !");
```

## 9.4 Objet this

Il existe un objet très utile en JS objet : `this`. Même s'il sera plus utile en programmation objet, lors de la création de classes, il peut servir de le connaître. Il s'agit d'un objet qui représente l'objet en cours, souvent l'objet passé en paramètre. Il possède alors les propriétés et les méthodes de l'objet. Etant donné son utilité minime avant la programmation objet, cette section ne comportera pas d'exemple à ce sujet.

# 10 FORMULAIRES

## 10.1 Généralités

### 10.1.1 Présentation

Voici la partie importante du JS. Sans les formulaires, les pages HTML ne proposent aucune interactivité avec l'utilisateur. En effet, la page appelée est envoyée par le serveur et le browser ne fait que l'afficher, il ne peut rien faire avec le serveur. Les champs de formulaire (`form`) permettent à l'utilisateur d'entrer des informations auxquelles la page pourra réagir. Ces réactions seront programmées à l'avance par le programmeur. Cela reste assez limité, puisque l'interaction ne quitte pas la machine du client, mais on peut effectuer quelques traitements.

Cette section – et par extension, ce cours – n'a pas vocation a apprendre le HTML. Le fonctionnement de ce langage ne sera donc pas évoqué ici. De plus, les propriétés JS de chaque contrôle seront décrites dans cette section. Il n'est pas utile de les comprendre ou de les utiliser maintenant, mais il semblait plus logique de les répertorier ici.

### 10.1.2 Balise form

Chaque formulaire doit être encadré par les balises `<form>…</form>`. Toutes les balises du formulaire seront comprises entre ces deux-là. Il est à noter qu'il n'est pas possible d'imbriquer plusieurs formulaires.

Dans le cas de l'utilisation de PHP ou d'ASP[12], il sera certainement utile d'envoyer les informations à un serveur. Dans ce cas, il faut indiquer dans la balise `<form>` la méthode (**`post`** ou **`get`**) et l'URL du script qui traitera l'information, comme dans l'exemple suivant :

**Exemple 10.1 – Envoi d'informations à un script**

```
<form method='post' action='traitement.php'>
……
</form>
```

Les informations peuvent aussi être envoyées par mail. Il faudra alors préciser '`mailto:…`' dans l'action.

[12] Langages de programmation semblables au HTML, mais permettant une interface serveur.

**Exemple 10.2 – Envoi d'informations par mail**

```
<form method='post' action='mailto:e-mail'>
……
</form>
```

Dans chaque cas, il ne faut surtout pas oublier d'insérer un bouton `submit` dans le formulaire.

## 10.2 Champ de texte

### 10.2.1 Ligne de texte

Il s'agit de l'élément d'entrée/sortie le plus courant en JS. Une simple ligne où l'utilisateur peut écrire quelques mots. Il est possible d'y afficher un texte. Voici la syntaxe à utiliser :

```
<input type="text" name="nom" value="valeur" size=x
maxlength=y>
```

Cette balise possède 4 attributs HTML. Seul le nom est obligatoire mais les autres sont facultatifs. Voici la liste de ces attributs :

- `name` : le nom du champ (servira lors de la programmation)
- `value` : la valeur que vous voulez afficher dans la ligne de texte (facultatif).
- `size` : la longueur de la ligne de texte. Si le texte est trop long, la ligne défilera (facultatif).
- `maxlength` : le nombre de caractères maximum contenus dans la ligne de texte (facultatif).

Il est possible de rattacher 3 propriétés à ce contrôle. Elles sont assez basiques. Ce sont les suivantes :

- `name` : indique le nom du contrôle.
- `defaultvalue` : indique la valeur par défaut affichée dans la ligne de texte.
- `value` : indique la valeur actuellement dans la zone de texte.

**Exemple 10.3 – Ligne de texte**

```
<form method="post" action="mailto:lapape@le-vatican.com">
<input type="text" name="text1" size=20 maxlength=40><br>
<input type="submit"name="envoi" value="envoyer">
</form>
```

### 10.2.2 Zone de texte

Il s'agit – en résumé - de plusieurs lignes de texte. Des barres de déroulements s'affichent Lorsque le texte est trop long. C'est utile quand l'information demandée est conséquente. On utilise alors la balise `<textarea></textarea>` :

```
<textarea name=”nom” rows=x cols=y>
texte par défaut
</textarea>
```

Les 3 attributs de la balise `<textarea>` sont facultatifs, mais il est conseillé de les déclarer tous les 3 quand même.

- `name` : le nom du champ (servira lors de la programmation)
- `rows` : le nombre de lignes affichés à l'écran. Si le texte est trop long, la zone de texte défilera.
- `cols` : le nombre de colonnes affichées à l'écran.

La balise `<textarea>` possède les mêmes propriétés JS que la ligne de texte, à cela près que la `value` ne correspond pas à l'attribut de la balise, qui n'existe pas. Voici la liste de ces propriétés :

- `name` : indique le nom du contrôle.
- `defaultvalue` : indique la valeur par défaut affichée dans la ligne de texte.
- `value` : indique la valeur actuellement dans la zone de texte.

**Exemple 10.4 – Zone de texte**

```
<form method=”post” action=”mailto:lapape@le-vatican.com”>
<textarea name=”text2” rows=5 cols=20>
zone de texte
</textarea><br>
<input type="submit" name="envoi" value="envoyer">
</form>
```

### 10.2.3 Champ password

Il s'agit d'une ligne de texte dont les caractères sont cachés. Elle sert – comme son nom l'indique – à faire entrer un mot de passe à l'utilisateur. On utilise pour cela la balise `<input>`, déjà utilisée pour la ligne de texte :

```
<input type=”password” name=”nom” value="valeur"
size=x maxlength=z>
```

Les attributs de la balise `<input>` sont ceux vu précédemment. Se reporter à la description de la ligne de texte.

- `name` : le nom du champ (servira lors de la programmation)
- `value` : la valeur que vous voulez afficher dans le champ (facultatif).
- `size` : la longueur de la ligne de texte. Si le texte est trop long, la ligne défilera.
- `maxlength` : le nombre de caractères maximum contenus dans la ligne de texte (facultatif mais conseillé).

Les propriétés du champ `password` sont celles d'une ligne de texte. Il y en a aussi 3 et elle sont identiques :

- `name` : indique le nom du contrôle.
- `defaultvalue` : indique la valeur par défaut affichée dans la ligne de texte.
- `value` : indique la valeur actuellement dans le champ password.

**Exemple 10.5 – Champ password**

```
<form method=”post” action=”mailto:lapape@le-vatican.com”>
<input type=”password” value=”password” name=”text1” size=20
maxlength=40><br>
<input type="submit"name="envoi" value="envoyer">
</form>
```

## 10.3 Cases à sélectionner

### 10.3.1 Boutons radios

Il s'agit de cases à cocher à choix unique, c'est-à-dire que l'on ne peut en sélectionner qu'une – si le code est bien construit -. D'une forme ronde, elles sont liées entre elles au niveau du code JS. On retrouve encore ici la balise `<input>`, précédemment utilisée. Il est nécessaire de préciser le label après le contrôle (***texte***), comme ceci :

```
<input type=”radio” name=”nom” value="valeur"> texte
```

Les attributs de la balise des boutons radios sont différents de ceux de la balise `<input>` habituelle. Il n'y a plus de valeur par défaut, mais la possibilité de pré cocher une case.

- `name` : le nom du champ (servira lors de la programmation). Il doit être identique pour chaque choix.
- `value` : la valeur que vous voulez afficher dans le champ (facultatif).
- `checked` : à mettre sur un seul bouton, qui sera sélectionné par défaut.

Le nombre de propriétés JS est plus important ici. On retrouve les propriétés liées à la sélection d'une case ainsi qu'un index.

- `name` : indique le nom du contrôle.
- `checked` : indique si le bouton radio est coché, actuellement.
- `defaultchecked` : indique si le bouton radio est coché ou non par défaut.
- `value` : indique la valeur du bouton radio.
- `index` : indique le rang du bouton radio, à partir de 0.

**Exemple 10.6 – Boutons radios**

```
<form method=”post” action=”mailto:lapape@le-vatican.com”>
<input type=”radio” name=”choix” value=”1”> choix 1<br>
<input type=”radio” name=”choix” value=”2”> choix 2<br>
<input type=”radio” name=”choix” value=”3”> choix 3<br>
<input type="submit"name="envoi" value="envoyer">
</form>
```

### 10.3.2 Checkbox

Il s'agit de cases à cocher à choix multiple, c'est-à-dire que l'on peut en sélectionner plusieurs. C'est toujours la balise `<input>` qui est utilisée. Il est aussi nécessaire de préciser le label après le contrôle (***texte***) :

```
<input type=”checkbox” name=”nom” value="valeur"> texte
```

On retrouve les paramètres HTML de la balise `<input>` des boutons radio, puisque ici aussi, la sélection des cases entre en jeu.

- `name` : le nom du champ (servira lors de la programmation). Il doit être différent pour chaque choix.
- `value` : la valeur que vous voulez afficher dans le champ (facultatif).
- `checked` : à mettre sur un (ou plusieurs) bouton, qui sera sélectionné par défaut.

Les propriétés JS sont exactement les mêmes, au détail près que plusieurs contrôles peuvent avoir la propriété `checked` ayant la valeur « `true` ».



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

- `name` : indique le nom du contrôle.
- `checked` : indique si la case est cochée, actuellement.
- `defaultchecked` : indique si la case est cochée ou non par défaut.
- `value` : indique la valeur de la case à cocher.

**Exemple 10.7 - Checkbox**

```
<form method=”post” action=”mailto:lapape@le-vatican.com”>
<input type=”checkbox” name=”choix1” value=”1”>choix 1<br>
<input type=”checkbox” name=”choix2” value=”2”>choix 2<br>
<input type=”checkbox” name=”choix3” value=”3”>choix 3<br>
<input type="submit"name="envoi" value="envoyer">
</form>
```

## 10.4 Liste déroulante

Il s’agit d’une liste déroulante dans laquelle on peut sélectionner une option, derrière laquelle on doit préciser le label – au niveau du code - sans quoi la liste sera vide. Cette fois-ci, on utilisera la balise `<select>` couplée à `<option>`. Chaque option de la liste déroulante correspond à une balise `<option>`. Il est nécessaire de préciser le label après chaque option contrôle (***texte***).

```
<select name=”nom” size=x>
<option value="valeur"> texte
</select>
```

Les attributs HTML sont plus nombreux. On retrouve les 3 paramètres classiques auxquels on ajoute la sélection par défaut – sous un autre nom – et la sélection multiple.

- `name` : le nom du champ (servira lors de la programmation).
- `size` : le nombre d’options que vous voulez afficher à l’écran. S’il n’est pas précisé, la liste n’affiche qu’une seule ligne.
- `value` : la valeur que vous voulez afficher dans le champ (facultatif).
- `selected` : à mettre dans une balise `<option>`. Cette option sera sélectionnée par défaut.
- `multiple` : à mettre dans une balise `<select>`. Autorise la sélection de plusieurs options dans la liste déroulante.

Les propriétés JS sont semblables à celles de la balise de `checkbox`. La longueur de la liste apparaît cependant.

- `name` : indique le nom du contrôle.
- `selected` : indique si le bouton radio est sélectionné

- `defaultselected` : indique si le bouton radio est coché ou non par défaut.
- `length` : indique le nombre d'éléments de la liste.

Les exemples suivants montrent la différence entre une liste sans l'attribut `size` et une liste où celui-ci est précis, avec une valeur supérieure à 1.

**Exemple 10.8 – Liste déroulante avec l'attribut size**

```
<form method=”post” action=”mailto:lapape@le-
vatican.com”>
<select name=”choix” size=3>
<option value=”1”>choix 1
<option value=”2”>choix 2
<option value=”3”>choix 3
</select><br>
<input type="submit"name="envoi" value="envoyer">
</form>
```

**Exemple 10.9 – Liste déroulante sans l'attribut size**

```
<form method=”post” action=”mailto:lapape@le-
vatican.com”>
<select name=”choix”>
<option value=”1”>choix 1
<option value=”2”>choix 2
<option value=”3”>choix 3
</select><br>
<input type="submit"name="envoi" value="envoyer">
</form>
```

## 10.5 Boutons

### 10.5.1 Bouton simple

Il s'agit du bouton tout simple, que l'on utilise pour appeler une fonction dans la plupart des cas. Il faut alors utiliser de nouveau la balise `<input>` :

```
<input type=”button” name=”nom” value=”valeur”>
```

Les attributs HTML sont au nombre de 2. Ce sont ceux du contrôle de ligne de texte à une exception près.

- `name` : le nom du champ (servira lors de la programmation).
- `value` : la valeur que vous voulez afficher sur le bouton.

Les propriétés JS restent les mêmes que celles de plusieurs autres contrôles. Ce sont :

- `name` : indique le nom du contrôle.
- `value` : indique la valeur actuelle du bouton.
- `defaultvalue` : indique la valeur par défaut du bouton.

Il est à préciser que le bouton de l'exemple ci-dessous n'a alors aucune utilité. Il faudra lui attribuer une action à l'aide des évènements, ce qui est l'objet de la section suivante.

**Exemple 10.10 – Bouton simple**

```
<form method=”post” action=”mailto:lapape@le-
vatican.com”>
<input type="button"name="click" value="Cliquez">
</form>
```

### 10.5.2 Bouton reset

Il permet de remettre la valeur par défaut de tous les champs du formulaire. Cela offre une interaction supplémentaire à votre formulaire. On utilisera encore la balise <input> :

```
<input type=”reset” name=”nom” value=”valeur”>
```

On retrouve les mêmes attributs que pour le bouton simple.

- `name` : le nom du champ (servira lors de la programmation).
- `value` : la valeur que vous voulez afficher sur le bouton.

Les propriétés JS sont aussi identiques.

- `name` : indique le nom du contrôle.
- `value` : indique la valeur actuelle du bouton.
- `defaultvalue` : indique la valeur par défaut du bouton.

L'exemple ci-dessous possède un bouton reset, qui permet de re-sélectionner par défaut l'option 3 :

**Exemple 10.11 – Bouton reset**

```
<form method=”post” action=”mailto:lapape@le-vatican.com”>
<select name=”choix” size=3>
<option value=”1”>choix 1
<option value=”2”>choix 2
<option value=”3” selected>choix 3
</select><br>
<input type="reset" name="effacer" value="effacer">
</form>
```

### 10.5.3 Bouton submit

Il permet d’exécuter l’action prévue dans la balise <form>, comme indiqué dans la section traitant de cette balise. En règle générale, il permet d’envoyer les données saisies dans le formulaire. On retrouve une nouvelle fois la balise <input> :

```
<input type=”submit” name=”nom” value=”valeur”>
```

On retrouve les mêmes attributs que pour le bouton simple.

- name : le nom du champ (servira lors de la programmation).
- value : la valeur que vous voulez afficher sur le bouton.

Les propriétés JS sont aussi identiques.

- name : indique le nom du contrôle.
- value : indique la valeur actuelle du bouton.
- defaultvalue : indique la valeur par défaut du bouton.

**Exemple 10.12 – Bouton submit**

```
<form method=”post” action=”mailto:lapape@le-vatican.com”>
<select name=”choix” size=3>
<option value=”1”>choix 1
<option value=”2”>choix 2
<option value=”3” selected>choix 3
</select><br>
<input type="submit" name="envoi" value="envoyer">
</form>
```

## 10.6 Contrôle caché

Il permet de mettre dans le script des éléments (souvent des données) qui ne seront pas affichées par le navigateur, mais qui pourront être envoyées par mail ou à un serveur. Il faut toujours utiliser la balise <input> :

```
<input type=”hidden” name=”nom” value=”valeur”>
```

On retrouve les mêmes attributs que pour le bouton simple.

- `name` : le nom du champ (servira lors de la programmation).
- `value` : la valeur qui sera contenue dans le contrôle.

Les propriétés JS sont aussi identiques.

- `name` : indique le nom du contrôle.
- `value` : indique la valeur actuelle du contrôle.
- `defaultvalue` : indique la valeur par défaut du contrôle.

Lors de l’envoi du formulaire ci-dessous, la valeur du contrôle caché sera aussi envoyée. Cela permet par exemple de donner la date ou le nom du formulaire.

**Exemple 10.13 – Contrôle caché**

```
<form method=”post” action=”mailto:lapape@le-vatican.com”
name="form1">
<input type="hidden" name="date" value="30/06/2004">
<input type=”button” name=”click” value=”Vous pouvez
cliquez!” onClick="alert(document.form1.date.value);">
</form>
```

## 10.7 Formulaire complet

Voilà un formulaire complet, utilisant la plupart des contrôles cités ci-dessus. Il pourrait s’agir d’un formulaire réel, mais il est inventé de toutes pièces. De nombreux sites proposent des formulaires, rien n’empêche d’enregistrer la page et d’aller fouiner pour voir leurs astuces…

**Exemple 10.14 – Formulaire complet**

```
<form method=”post” action=”traitement.php”>
Inscription au stage<br>
Identité<br>
<input type="hidden" name="date" value="15/06/04">
<input type=text name=”nom” value=”Nom”> 
<input type=text name=”prénom” value=”Prénom”><br>
<input type=text name=”date” value=”Date de naissance”><br>
Activité<br>
<select name=”Activité”>
<option value=”Collégien”>Collégien
<option value=”Lycéen”>Lycéen
<option value=”Etudiant”>Etudiant
<option value=”Salarié”>Salarié
<option value=”Fonctionnaire”>Fonctionnaire
<option value=”Patron”>Patron
</select><br>
Qualification(s):<br>
<input type=”checkbox” name=”choix1” value=”bafa”> BAFA<br>
<input type=”checkbox” name=”choix2” value=”bafd”> BAFD<br>
<input type=”checkbox” name=”choix3” value=”rien”> rien<br>
Semaine choisie:<br>
<input type=”radio” name=”semaine” value=”1”>
1ère<br>
<input type=”radio” name=”semaine” value=”2”>
2ème<br>
<input type=”radio” name=”semaine” value=”3”>
3ème<br>
Vos motivations :<br>
<textarea name="motivations" rows=5 cols=20></textarea><br>
<input type="submit" name="envoi" value="envoyer">
 
<input type="reset" name="effacer" value="effacer">
</form>
```

## 10.8 Objet des formulaires

La notion d’objet a été évoquée précédemment. Maintenant que tous les composants d’un formulaire sont connus, il va être possible d’utiliser le concept objet appliqué aux formulaires.

Même si les objets du navigateur n'ont pas été abordés, l'objet document sera introduit ici. Il renvoie à la page en cours. Il contient en propriétés tous les éléments de la page. Les attributs des balises deviennent les propriétés de l'élément objet. On y a accès de la façon suivante :

```
document.formulaire.élément.propriété
```

Un exemple éclairera le propos. L'exemple suivant montre comment attribuer une valeur à une ligne de texte.

**Exemple 10.15 – Attribuer une valeur à une ligne de texte**

```
<html>
<head>
<script language = "Javascript">
function f() {
       document.form1.texte.value="voici du texte";
       }
</script>
</head>
<body onLoad="f();">
<form name="form1">
<input type="text" name="texte" value="">
</form>
</body>
</html>
```

Dans l'exemple ci-dessus, la fonction est appelée à la fin du chargement grâce au gestionnaire d'événement placé dans la balise `<body>`.

De cette façon, on peut accéder à chaque élément de chaque formulaire de la page. Il est important de donner un nom, à la fois au formulaire et au contrôle. Ce nom doit de préférence être explicite, même si celui du formulaire peut rester classique, étant donné qu'il y en a rarement plusieurs sur une seule et même page. Lors de l'utilisation des frames, en revanche, ce nom sera important.

Cette manière de pouvoir accéder aux objets est très pratique, même si pour le moment, elle ne semble pas très utile. Cette utilité sera mise en valeur dans la section suivante.

## 10.9 Exercice

Le but de cet exercice est d'arriver à un formulaire complet et assez ergonomique. Il faudra faire attention aux choix des contrôles et à la disposition. La solution proposée n'est pas parfaite – loin de là – mais elle illustre à peu près ce qu'est un formulaire de qualité correcte.

Le formulaire à créer est un formulaire d'adhésion à une association, pour l'activité échecs. Les informations demandées sont les suivantes : nom, prénom, date de naissance, numéro de téléphone, adresse, mail, profession (choix, dont « autres : préciser »), centres d'intérêt (choix), niveau de jeu, motivations. Il faudra présenter ce formulaire de façon assez esthétique, mais la police et la taille de caractères ne sont pas importantes, cela dépend plutôt d'un cours HTML. Bien entendu, il faut envoyer le tout à traitement.php, par la méthode `post` en cliquant sur un bouton.

# 11 EVENEMENTS

## 11.1 Généralités

### 11.1.1 Présentation

Les événements sont l'intérêt du JS en matière de programmation Web. Ils donnent une interactivité à la page que l'internaute consulte, ce qui n'existe pas avec le HTML, si on excepte bien entendu le lien hypertexte. Le JS permet de réagir à certaines actions de l'utilisateur. On nomme évènement le mot *Event*, et gestionnaire d'événement le mot-clé `onEvent`.

### 11.1.2 Fonctionnement

Les évènements sont indiqués dans la balise, d'un formulaire, le plus souvent. Ils apparaissent comme un paramètre supplémentaire et suivent une syntaxe particulière :

```
<balise onEvent="code">
```

Il semble utile d'expliciter le code ci-dessus. La partie balise désigne le nom de la balise HTML qui supporte l'événement. L'attribut `onEvent` désigne le gestionnaire d'événement associé à l'événement *Event*. Enfin, le code inséré entre guillemets fait le plus souvent référence à une fonction définie dans les balises `<head>…</head>`. Il peut cependant s'agir d'instructions directement écrites à cet endroit.

Plusieurs gestionnaires d'évènements peuvent être placés dans la même balise. Certaines balises n'appartenant pas à un formulaire peuvent supporter des gestionnaires d'évènement. Par exemple :

**Exemple 11.1 – Balise avec plusieurs gestionnaires d'évènement**

```
<a href="http://www.google.fr" onClick="alert('vous avez
cliqué!');"
onMouseOver="alert('Héhé') ;"> click?</a>
```

## 11.2 Clic de souris

Lors d'un clic sur un élément de formulaire de la page consultée, l'utilisateur crée un événement – au sens du JS -. Son gestionnaire d'événement est `onClick`.

**Exemple 11.2 - onClick**

```
<input type=”button” onClick=”alert(‘vous avez cliqué
sur le bouton’) ;">
```

Cet évènement est supporté par les balises suivantes : `<input type="button">`, `<input type="checkbox">`, `<input type="radio">`, `<input type="reset">`, `<input type="submit">`, `<a href..>`.

## 11.3 Chargement

### 11.3.1 Load

Il se produit lorsque la page appelée – ou consultée - finit de se charger. Son gestionnaire d’événement est `onLoad`.

**Exemple 11.3 - onLoad**

```
<body onLoad=”alert(‘la page est chargée !’) ;">
```

Ce gestionnaire d’évènement est supporté uniquement par les balises `<body>` et `<frameset>`.

### 11.3.2 UnLoad

Il se produit lorsque vous quittez un document ou une page web. Son gestionnaire d’événement est `onUnLoad`.

**Exemple 11.4 - onUnLoad**

```
<body onUnLoad=”alert(‘Vous quittez la page !’) ;">
```

Ce gestionnaire d’évènement est supporté uniquement par les balises `<body>` et `<frameset>`.

### 11.3.3 Error

Lorsque le chargement d’une page ou d’un image s’interrompt en erreur, cela produit un événement JS. Le gestionnaire d’événement associé est `onError`.

**Exemple 11.5 - onError**

```
<img src="pix.gif" onError=”alert(‘Erreur de chargement !’) ;">
```

Il est supporté par les 3 balises suivantes : `<body>`, `<frameset>` et `<img>`.

### 11.3.4 Abort

Il se produit lors de l’interruption le chargement d’une image. Le gestionnaire d’événement est alors `onAbort`.

**Exemple 11.6 - onAbort**

```
<img src="pix.gif" onAbort=”alert(‘Vous avez
interrompu le chargement de l’image !’) ;">
```

Il est supporté uniquement par les images, c’est-à-dire par la balise suivante : `<img>`.

## 11.4 Passage de la souris

### 11.4.1 MouseOver

Cela se produit lors du survol d’un lien ou d’une zone d’image activable. Une zone d’image activable est une partie d’image qui a été transformée en lien. Le gestionnaire associé à cet évènement est `onMouseOver`.

**Exemple 11.7 - onMouseOver**

```
<a href="http://www.google.fr"
onMouseOver=”alert(‘Pour aller sur google.fr, cliquer
ici’) ;">http://www.google.fr</a>
```

Ce gestionnaire est supporté par les balises de lien, donc les 2 balises suivantes : `<a href…>` et `<area href…>`.

### 11.4.2 MouseOut

Lorsque le curseur de la souris sort de la zone de survol d’un lien ou d’une zone d’image activable, cela produit aussi un évènement. Son gestionnaire d’événement est `onMouseOut`.

**Exemple 11.8 - onMouseOut**

```
<a href="http://www.google.fr" onMouseOut=”alert(‘Vous
ne voulez pas y aller ?’) ;">http://www.google.fr</a>
```

Il est supporté par mêmes balises que `onMouseOver` : `<a href…>` et `<area href…>`.

## 11.5 Focus des contrôles

### 11.5.1 Focus

Il se produit lors de l’activation d’un contrôle texte ou d’une sélection. Son gestionnaire d’événement est `onFocus`.

**Exemple 11.9 - onFocus**

```
<input type="text" value="votre nom" name=”nom”
onFocus=”alert(‘Ecrivez votre nom ici’) ;">
```

Il est supporté par les balises suivantes : `<input type="text">`, `<select>`, `<textarea>` et `<input type="password">`.

### 11.5.2 Blur

Lorsque l’utilisateur quitte un contrôle texte ou sélection. Le gestionnaire associé est `onBlur`.

**Exemple 11.10 - onBlur**

```
<input type="text" value="votre nom" name=”nom”
onBlur=”alert(‘Vous n\’avez rien oublié ?’) ;">
```

Ce gestionnaire est supporté par les balises `<input type="text">`, `<select>`, `<textarea>` et `<input type="password">`.

## 11.6 Changement

Lorsque la valeur d'un texte ou d'une option change dans un formulaire, cela produit aussi un événement en JS. Si l'utilisateur clique dans la zone de texte mais que vous ne touchez pas au texte, rien ne se produit. Son gestionnaire d'événement est le suivant `onChange`.

**Exemple 11.11 - onChange**

```
<input type="text" value="votre nom" name=”nom”
onChange=”alert('Vous avez changé votre nom ??') ;">
```

Il est supporté par les balises suivantes : `<input type="text">`, `<select>`, `<textarea>`, `<input type="password">`.

## 11.7 Sélection

Lorsque l'internaute sélectionne du texte dans un champ de texte. Le gestionnaire d'événement associé est `onSelect`.

**Exemple 11.12 - onSelect**

```
<input type="text" value="votre nom" name=”nom”
onSelect=”alert('Vous avez sélectionné un champ') ;">
```

Il est supporté par les 2 balises de texte, c'est-à-dire `<input type="text">` et `y`.

## 11.8 Envoi

Lorsque l'internaute clique sur un bouton `submit` d'un formulaire de type « `post` » ou « `get` ». Le gestionnaire d'événement est alors `onSubmit`.

**Exemple 11.13 - onSubmit**

```
<input type="submit" value="Envoyer" name=”envoi”
onSubmit=”alert('C'est parti !') ;">
```

Il est supporté uniquement par la balise `<input type="submit">`.

## 11.9 Reset

Lorsque l'internaute clique sur un bouton `reset` d'un formulaire. Son gestionnaire d'événement est `onReset`.

**Exemple 11.14 - onReset**

```
<input type="reset" value="Effacer" name=”effacer”
onSubmit=”alert(‘On efface tout !’) ;">
```

Il est supporté uniquement par la balise <`input type="reset"`>.

## 11.10 Objet document

Avant de voir les évènements, l'intérêt de l'accès aux éléments à travers l'objet document était discutable. Maintenant, on voit qu'il est possible de changer des valeurs selon des évènements, ce qui est très intéressant. Il est aussi possible de demander confirmation, de demander si on veut vraiment changer la valeur… Tout est permis. Cette partie ne comporte pas d'exemple, car la section suivante en présente un assez complet.

## 11.11 Exemple concret

Voici un exemple – parmi tant d'autres – de ce que pourrait être un formulaire interactif, avec tous les évènements auxquels on peut penser.

**Exemple 11.15 – Exemple de formulaire évènementiel**

```
<HTML>
<HEAD>
<TITLE>New Document</TITLE>
<script language="Javascript">
var TestLog = 0; //test pour le login
function ChangeLog() {
	TestLog++; //incrémente le test
	if (TestLog > 1) { //si le login a été changé plus d'une fois
		alert("Vous changez de login?\n Décidez-vous!");
		}
	}
function VerifPass () { // si les deux password sont différents
	if (document.form1.pass1.value != document.form1.pass2.value)
		{
		alert("Vous avez entré deux passwords différents !\nVérifiez les.");
		}
	}
function VerifMail () { //confirmation du mail
	 var test = confirm ("Votre mail est bien : " + document.form1.mail.value +
	 " ?");
	if (test == false) { //si l'internaute ne confirme pas
		alert("N'oubliez pas de changer votre mail!");
		}
	}
</script>
</HEAD>
<BODY>
<form name="form1">
	<center>Formulaire d'inscription sur notre site</center><br/>
	Login : <input type="text" name="login" onChange="ChangeLog();"
value="login"><br/>
	Password : <input type="password" name="pass1"><br/>
	Password (vérification) : <input type="password" name="pass2"
onBlur="VerifPass();"><br/>
	Adresse e-mail : <input type="text" name="mail" onBlur="VerifMail();"
value="@"><br/>
	Vous voulez recevoir la newsletter? <input type="radio" name="news"
value="yes">Oui <input type="radio" name="news" value="no" onClick="alert('Tant
pis pour vous!')">Non<br/>
	<input type="submit" name="sub" value="Envoi"> <input type="reset"
name="reset" value="Effacer">
</form>
</BODY></HTML>
```

## 11.12 Exercice

### 11.12.1 Bouton

L'exercice consiste à faire un bouton dont la valeur s'incrémente à chaque clic.

### 11.12.2 Phrase

L'exercice consiste à demander 3 informations à un utilisateur, puis de les afficher dans une zone de texte en faisant une phrase. Il faudra demander le nom à l'aide d'une boite de message, ainsi que l'âge et la ville avec des lignes de texte. Ensuite, en cliquant sur un bouton, on affiche une phrase contenant les 3 informations dans une zone de texte.

# 12 OBJET ARRAY

## 12.1 Généralités

L'objet `Array` n'est rien d'autre qu'un tableau. Un tableau est en quelque sorte une liste d'éléments indexés que l'on pourra lire – chercher les données - ou écrire – entrer les données – à volonté.

## 12.2 Création et affectation

### 12.2.1 Création

Un tableau se crée comme n'importe quel objet de classe. On invoque le constructeur de classe. Il faut indiquer en paramètre du constructeur le nombre maximum d'éléments (x) que l'on souhaite entrer dans le tableau. Ce nombre est cependant facultatif, car JS prend en compte le numéro du dernier élément entré comme taille du tableau si le ce nombre n'est pas indiqué. Voici la syntaxe à suivre :

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

**Exemple 12.1 – Création de tableau**

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

### 12.2.2 Affectation

On affecte un tableau très simplement. Il suffit d'affecter une valeur (`y`) à la variable avec le numéro de l'élément (`i`) entre crochets. Ces derniers ne sont pas présents lors de la création mais sont indispensables lors de l'affectation. Le numéro commence à 0 et finit au nombre maximum moins 1. Voici donc la syntaxe à suivre :

```
variable = new Array(x) ;
variable [i] = y;
```

**Exemple 12.2 – Affectation simple de tableau**

```
var MonTableau = new Array (2) ;
MonTableau [0] = 7 ;
MonTableau [1] = 4 ;
```

Evidemment, à ce rythme-là, l'affectation est longue, surtout si le tableau compte plus que quelques éléments. C'est pourquoi il est nécessaire d'utiliser la boucle itérative `for`. L'exemple suivant entre une série de nombre en ajoutant 1 à chaque fois. Il s'agit d'un exemple rapide et simple, mais on peut imaginer faire un calcul ou demander une valeur à l'utilisateur à chaque boucle.

**Exemple 12.3 – Affectation itérative de tableau**

```
var MonTableau = new Array (5) ;
for ( var i = 0 ; i < 5 ; i++) {
MonTableau [i] = i;
document.write (MonTableau[i]);
}
```

## 12.3 Accès aux éléments

On accède aux données d'un tableau en indiquant le numéro de l'élément entre crochets. On affecte cette valeur à une variable, par exemple. On peut aussi l'entrer comme paramètre d'une fonction.

```
variable1 = new Array(x) ;
variable1 [i] = y ;
variable2 = variable1 [i] ;
```

**Exemple 12.4 – Accès aux éléments d'un tableau**

```
var MonTableau = new Array (2) ;
MonTableau [0] = 7 ;
var element = MonTableau [0] ;
document.write (MonTableau [0]) ;
```

## 12.4 Tableau à deux dimensions

Pour créer un tableau à 2 dimensions, il faut utilise l'astuce suivante : déclarer chaque élément comme un nouveau tableau.

**Exemple 12.5 – Tableau à deux dimensions**

```
var MonTableau = new Array (2) ;
MonTableau [0] = new Array (7) ;
```

Il est bien entendu plus rapide d'utiliser une instruction itérative pour créer ce tableau à 2 dimensions.

**Exemple 12.6 – Tableau à deux dimensions**

```
var MonTableau = new Array (5) ;
for ( var i = 0 ; i < 5 ; i++) {
MonTableau [i] = new Array (3);
}
```

Avec ce système, il est possible de créer un tableau à 3, 4, 5 dimensions ou plus, bien que l'utilité en soit quelque peu douteuse…

## 12.5 Propriétés et méthodes

Comme tout objet, l'objet `Array` possède une propriété et des méthodes qui s'avèrent assez utiles.

### 12.5.1 Propriété

L'objet `Array` ne possède qu'une propriété – `length` – qui désigne le nombre d'éléments du tableau. Elle s'utilise selon la syntaxe suivante :

```
variable = new Array (x) ;
y = variable.length ;
```

**Exemple 12.7 – Propriété length**

```
var MonTableau = new Array (2) ;
document.write (MonTableau.length) ;
```

### 12.5.2 Méthodes

L'objet `Array` possède plusieurs méthodes qui permettent de manier les éléments du tableau. La syntaxe est la suivante :

```
tableau.méthode() ;
```

La liste de ces méthodes est longue, elles permettent de manier les éléments d'un tableau avec facilité. Certaines ne sont pas utiles, mais il faut connaître les autres. En voici la liste avec leur syntaxe :

- `join (séparateur)` : regroupe tous les éléments du tableau en une seule chaîne. Chaque élément est séparé par un séparateur. Ci celui-ci n'est pas précisé, ce sera une virgule.
- `reverse ()` : inverse l'ordre des éléments sans les trier.
- `sort ()` : Renvoie les éléments par ordre alphabétique, s'ils sont de même nature.
- `concat(tableau)` : concatène le tableau passé en paramètre avec celui de la méthode.
- `pop()` : supprime le dernier élément du tableau.
- `push(élément1,…)` : ajoute l(es) élément(s) passé(s) en paramètre à la fin du tableau.
- `shift()` : supprime le premier élément du tableau.
- `slice(début,fin)` : renvoie les éléments contenus entre la position supérieure ou égale à début et inférieure à fin.
- `splice(début, nombre, éléments)` : supprime les éléments à partir de la position début et sur nombre de position. Les éléments sont remplacés par ceux fournis en paramètre (facultatif).
- `unshift(élément1,…)` : ajoute l(es) élément(s) passé(s) en paramètre au début du tableau.

**Exemple 12.8 – Méthodes de l'objet Array**

```
var MonTableau = new Array (4) ;
MonTableau [0] = "Moi" ;
MonTableau [1] = "Toi" ;
MonTableau [2] = "Lui" ;
MonTableau [3] = "Elle" ;
MonTableau.reverse() ;
document.write (MonTableau.join(' ')) ;
MonTableau.sort() ;
document.write ("<br>" + MonTableau.join(' ')) ;
```

## 12.6 Exemple concret

Le but de cet exemple est de remplir et afficher les valeurs à 0.1 près comprises entre 0 et 10 exclu. Pour cela, on utilise un tableau à deux dimensions, dont la première indique la partie entière du nombre, et la deuxième dimension indique la partie décimale. On va d'abord créer le tableau, puis le remplir et enfin l'afficher.

**Exemple 12.9 – Exemple de manipulation de tableau**

```
<script language="Javascript">
var Tableau = new Array (10) ; // objet tableau
for (i = 0; i < 10; i++) { // pour chaque élément...
	Tableau[i] = new Array(10); // on redéfinit un
tableau
	}
// remplissage du tableau
for (i = 0; i < 10; i++) { /* pour chaque élément de
la première dimension*/
	for (j = 0; j < 10; j++) { /* pour chaque
élément de la seconde dimension*/
		Tableau[i][j] = i + "." + j; //on remplit
l'élément
		}
	}
// affichage du tableau
for (i = 0; i < 10; i++) { /* pour chaque élément de
la première dimension */
	for (j = 0; j < 10; j++) { /* pour chaque
élément de la seconde dimension */
		document.write( Tableau[i][j] + " ; " );
/*on affiche l'élément, avec un point virgule */
		}
	document.write ("<br>"); /* a chaque unité, on
revient à la ligne. */
	}
</script>
```

## 12.7 Exercice

Le but de l'exercice est d'afficher les entiers compris entre 1 et 10 inclus dans l'ordre décroissant. Il faudra utiliser pour cela un tableau de 10 éléments.

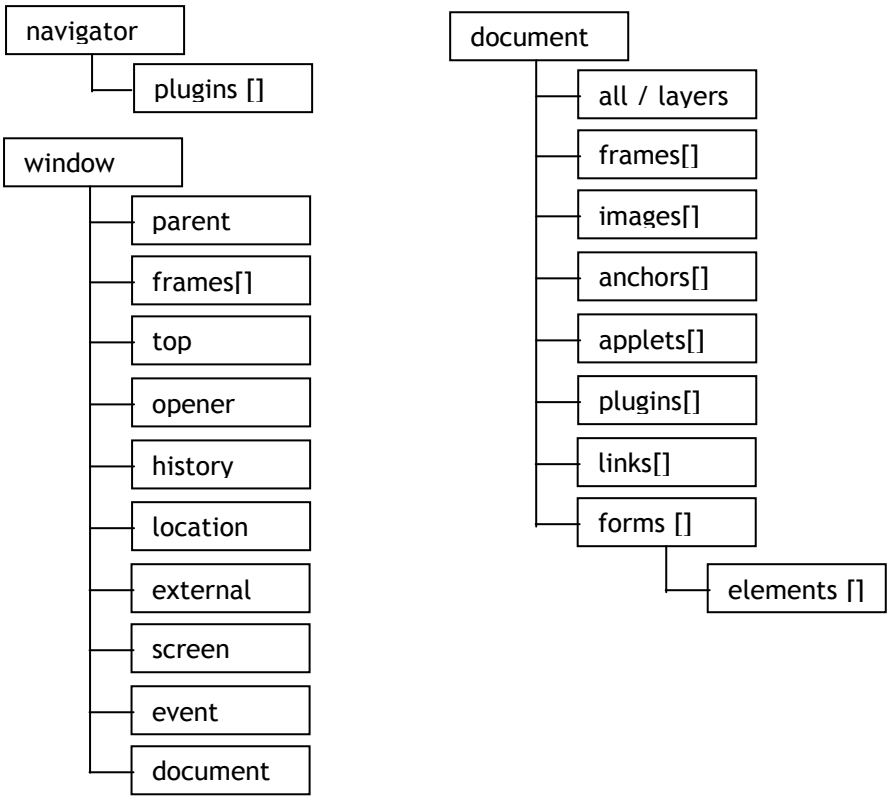
# 13 OBJETS DU NAVIGATEUR

## 13.1 Généralités

Les classes prédéfinies en JS ont été abordées précédemment. Il existe aussi plusieurs objets en JS rattachés à la fenêtre, à la page et au navigateur. Ils sont appelés `window`, `document` et `navigator`. Ce sont 3 classes distinctes - sans lien entre elle – puisque l'héritage n'existe pas en JS. Chaque classe sera décrite l'une après l'autre.

## 13.2 Arborescence

Voici l'arborescence des objets de JS. Elle comprend les 3 objets principaux ainsi que tous les « sous-objets » contenus dans chaque objet principal.



# 14 OBJET NAVIGATOR

Il s'agit – comme son nom l'indique – du navigateur - ou browser -. Les deux principaux étant sûrement Microsoft Internet Explorer et Netscape Navigator. L'objet `navigator` possède toutes les caractéristiques du navigateur ainsi que certaines de l'ordinateur de l'utilisateur. Cela peut s'avérer utile pour tester la compatibilité de certains codes avec un browser.

L'objet `navigator` étant intégré dans le langage, il n'est pas nécessaire de créer une instance de cette classe.

## 14.1 Propriétés

Toutes ces propriétés ne font que donner des informations sur le type de navigateur de l'utilisateur. Pour y accéder, il faut suivre la syntaxe habituelle d'un objet :

```
variable = navigator.propriété ;
```

Les propriétés qui suivent fonctionnent sous tous les navigateurs. Elles concernent principalement les caractéristiques du navigateur qui affiche la page en cours. Certaines propriétés ne fonctionnent que sous Netscape ou Internet Explorer, elles seront évoquées ensuite.

- `appCodeName` : nom de code du navigateur.
- `appName` : nom complet du navigateur.
- `appVersion` : numéro de version du navigateur ainsi que d'autres informations de plate-forme.
- `userAgent` : informations de `appCodeName` et de `appVersion` réunies.

Certaines propriétés ne fonctionnent qu'avec Microsoft Internet Explorer. Il est d'important de veiller à tester le type de browser avant d'exécuter le script !

- `appMinorVersion` : numéro de version mineure.
- `browserLanguage` : code langue du browser.
- `userLanguage` : code langue de l'utilisateur.
- `systemLanguage` : code langue du système d'exploitation.
- `cpuClass` : classe du processeur.
- `platform` : code type de plate-forme (pc, mac, linux …).
- `cookieEnabled` : booléen qui indique si l'utilisateur accepte les cookies.
- `onLine` : booléen qui indique si le poste est connecté à Internet.

Une propriété existe sous Netscape pour remplacer la propriété `browserLanguage` d'Internet Explorer. Il s'agit de `language` qui indique le code langue du browser.

## 14.2 Méthodes

Dans cette section sont présentées les méthodes de `navigator`, mais malheureusement, leur utilité est encore inconnue à l'auteur de ce document, qui saurait gré à un lecteur au courant de l'en avertir. Voici la syntaxe à utiliser :

```
variable = navigator.méthode() ;
```

Ces deux méthodes semblent tester l'activation du Java et du « `taint` » (infection ?). Voici leur nom : `javaEnabled()` et `taintEnabled()`.

## 14.3 Objets de navigator

L'objet `navigator` contient un autre objet, qui n'hérite pas de lui. Il s'agit du tableau `plugins`, qui n'est reconnu que par Netscape. Il s'agit de la liste de tous les plugins installés. Voici sa syntaxe :

```
variable = navigator.plugin.propriété ;
```

Cet objet possède 7 propriétés dont 6 qui sont appliquées à ses éléments. Elles permettent de connaître diverses caractéristiques des plugins installés. En voici la liste :

- `plugins.length` : nombre de plugins
- `plugins[i].name` : nom du plugin n° i.
- `plugins[i].filename` : nom de l'exécutable du plugin n° i.
- `plugins[i].description` : description du plugin n° i.
- `plugins[i].length` : nombre de types de fichiers supportés par le plugin n° i.
- `plugins[i][j].type` : type n° j du plugin n° i.
- `plugins[i][j].suffixes` : Extensions des fichiers du type n° j du plugin n° i.

## 14.4 Exemple

Le but de l'exemple 13.1 est d'afficher les informations du navigateur, en fonction de celui-ci. On commencera donc par tester le type de navigateur. L'exemple 13.1 n'est pas commenté car il n'a rien de nouveau excepté les propriétés vues ci-dessus.

**Exemple 14.1 – Objet navigator**

```
<script language="Javascript">
document.write (navigator.appCodeName + "<br>");
document.write (navigator.appName + "<br>");
document.write (navigator.appVersion + "<br>");
document.write (navigator.userAgent + "<br>");
if (navigator.appName == "Microsoft Internet Explorer") {
       document.write (navigator.appMinorVersion + "<br>");
       document.write (navigator.browserLanguage + "<br>");
       document.write (navigator.userLanguage + "<br>");
       document.write (navigator.systemLanguage + "<br>");
       document.write (navigator.cpuClass + "<br>");
       document.write (navigator.platform + "<br>");
       document.write (navigator.cookieEnabled + "<br>");
       document.write (navigator.onLine + "<br>");
       }
else {
       document.write (navigator.language + "<br>");
       }
</script>
```

# 15 OBJET WINDOW

Il s'agit – comme son nom l'indique – de la fenêtre du browser en cours d'exécution. Tous les éléments de la fenêtre sont des propriétés ou des méthodes de `window`.

L'objet `window` étant intégré dans le langage, il n'est pas nécessaire de créer une instance de cette classe.

## 15.1 Propriétés

Toutes ces propriétés correspondent à des éléments de la fenêtre ouverte. Elles permettent de changer quelques détails sympathiques dans le visuel d'une page web. Pour y accéder, on suit la syntaxe habituelle d'un objet :

```
window.propriété;
```

Il existe 6 propriétés qui correspondent chacune à des éléments ou des caractéristiques de la fenêtre. Voici leur liste :

- `defaultStatus` : le texte par défaut affiché dans la barre d'état.
- `status` : le texte à afficher dans la barre d'état, prioritaire par rapport à `defaultStatus`.
- `name` : le nom de la fenêtre
- `screenTop` : ordonnée du point supérieur gauche de la fenêtre.
- `screenLeft` : abscisse du point supérieur gauche de la fenêtre.
- `closed` : booléen indiquant si la fenêtre est fermée.

## 15.2 Méthodes

### 15.2.1 Généralités

L'objet `window` possède de nombreuses méthodes, dont certaines qui ont déjà été vues précédemment – les boîtes de dialogue -, et qui peuvent offrir quelques atouts à une page web. La plupart renvoient une valeur à la variable qui appelle la fonction. Voici leur syntaxe :

```
variable = window.méthode();
```

Certaines propriétés ne nécessitent pas de préciser le suffixe « `window.` » pour fonctionner. C'est notamment le cas des boites de dialogue.

### 15.2.2 Liste des méthodes

Le nombre des méthodes étant élevé, elles sont présentées sous forme d'une liste que voici :

- `alert('texte')` : boite de message avec un bouton unique.
- `prompt('texte', 'valeur_défaut')` : boite d'invite avec un texte informatif et une zone de texte avec une valeur par défaut facultative.
- `confirm('texte')` : boite de confirmation avec un texte informatif et deux boutons OK et Annuler.
- `print('texte')` : afficher le texte dans la page.
- `open('URL', 'nom', options)` : ouvre une page pop-up avec les caractéristiques données en paramètres.
- `close()` : ferme la fenêtre.
- `focus()` : donne le focus à la page.
- `blur()` : enlève le focus à la page.
- `moveBy(x,y)` : déplacement relatif du coin supérieur gauche de la fenêtre.
- `moveTo(x,y)` : déplacement absolu du coin supérieur gauche de la fenêtre.
- `resizeBy(x,y)` : redimensionnement relatif de la fenêtre.
- `resizeTo(x,y)` : redimensionnement absolu de la fenêtre.
- `scrollBy(x,y)` : scroll relatif.
- `scrollTo(x,y)` : scroll absolu.
- `setTimeOut('fonction', temps)` : déclenche une minuterie en millisecondes.
- `clearTimeout('timer')` : suspend la minuterie.
- `stopTimeOut('timer')` : arrête une minuterie.
- `setInterval(code, délai)` : exécute le code – sous forme de String - passé en paramètre à chaque fois que le délai est écoulé.
- `clearInterval(timer)` : arrête la minuterie définie avec `setInterval()`.
- `stop()` : arrête le chargement de la page.
- `home()` : ouvre la page de démarrage de l'internaute.

### 15.2.3 Exemples

Voici, pour illustrer la liste proposée ci-dessus, deux façons de faire un chronomètre. La première utilise la méthode `setInterval()` – moins connue - , ce qui économise un ligne de code.

La seconde utilise la méthode `setTimeout()` – plus connue - qui doit être rappelée à chaque boucle. Le reste du code est identique. Il sert à incrémenter à chaque fois les secondes, et permet de rendre l'affichage un peu plus agréable.

**Exemple 15.1 – La méthode setInterval()**

```
<HTML>
<HEAD>
<TITLE>New Document</TITLE>
<script language="Javascript">
var seconds = 0;
var minutes = 0;
var hours = 0;
function chrono () {
       if (seconds == 60) {
              seconds = 0;
              minutes++;
              }
       if (minutes == 60) {
              minutes = 0;
              hours++;
              }
       h = hours;
       m = minutes;
       s = seconds;
       if (s < 10) s = "0" + s;
       if (m < 10) m = "0" + m;
       if (h < 10) h = "0" + h;
       document.form1.chrono.value = h + ":" + m + ":" + s;
       seconds++;
       }
</script>
</HEAD>
<BODY onLoad="window.setInterval('chrono();',1000)">
<form name="form1">
<center><input type="text" value="" name="chrono"
size=6></center>
</form>
</BODY>
</HTML>
```

**Exemple 15.2 – La méthode setTimeOut()**

```
<HTML>
<HEAD>
<TITLE>New Document</TITLE>
<script language="Javascript">
var seconds = 0;
var minutes = 0;
var hours = 0;
function chrono () {
       if (seconds == 60) {
              seconds = 0;
              minutes++;
              }
       if (minutes == 60) {
              minutes = 0;
              hours++;
              }
       h = hours;
       m = minutes;
       s = seconds;
       if (s < 10) s = "0" + s;
       if (m < 10) m = "0" + m;
       if (h < 10) h = "0" + h;
       document.form1.chrono.value = h + ":" + m + ":" + s;
       seconds++;
       window.setTimeout('chrono()',1000);
       }
</script>
</HEAD>
<BODY onLoad="chrono();">
<form name="form1">
<center><input type="text" value="" name="chrono"
size=6></center>
</form>
</BODY>
</HTML>
```

## 15.3 Evènements

On peut rattacher certains évènements à l'objet `window`. Ils seront placés dans la balise `<body>`, grâce au gestionnaire d'événement `onEvent`. On utilise pour cela la syntaxe des gestionnaires d'évènements :

```
<body onEvent="code">
```

5 évènements sont rattachés à l'objet `window`. Ils concernent – logiquement – les actions effectuées sur la fenêtre. Voici leur liste :

- `load` : fin de chargement de la page.
- `unLoad` : déchargement de la page.
- `focus` : prise de focus de la fenêtre ou d'un de ses éléments.
- `blur` : perte de focus de la fenêtre ou d'un de ses éléments.
- `resize` : redimensionnement de la fenêtre.

## 15.4 Objets

L'objet `window` contient plusieurs autres objets assez réduits qui sont donc présentés dans cette section. La syntaxe est la suivante :

```
variable = window.objet.propriété ;
variable = window.objet.méthode() ;
```

### 15.4.1 Frames

Il s'agit d'un tableau contenant toutes les frames déclarées dans la page. Il ne possède ni propriétés ni méthodes. Chaque élément de ce tableau possède les propriétés et méthodes d'une frame. Voici la syntaxe pour accéder à une frame de la page :

```
window.frames[i].propriété ;
```

### 15.4.2 Parent

Il s'agit de la page principale, qui contient la déclaration de toutes les frames. Il possède les mêmes attributs que l'objet `window`. On accède aux propriétés et méthodes ainsi :

```
window.parent.méthode() ;
window.parent.propriété ;
```

### 15.4.3 Top

Il s'agit de la page de plus haut niveau du document affiché à l'écran. Cet objet possède un intérêt uniquement lorsqu'il y a des frames dans le document affiché. Il possède les mêmes attributs que l'objet `window`. Voici la syntaxe à suivre :

```
window.top.méthode() ;
window.top.propriété ;
```

### 15.4.4 Opener

Il s'agit de la page responsable de l'ouverture de la page courante, c'est-à-dire qui possède un lien hypertexte vers la page ouverte. Il possède les mêmes attributs que l'objet `window`. Voici la syntaxe :

```
window.opener.méthode() ;
window.opener.propriété ;
```

### 15.4.5 History

Il s'agit de l'historique des pages consultées. On utilise la syntaxe objet habituelle :

```
window.history.méthode() ;
window.history.propriété ;
```

Il possède une propriété et 3 méthodes. Qui permettent de naviguer dans l'historique comme les boutons de la barre d'outils. En voici la liste :

- `length` : le nombre d'entrées de l'historique.
- `back ()` : permet de retourner à la page précédente.
- `forward ()` : permet d'aller à la page suivante.
- `go (numéro)` : permet d'aller à la page du numéro passé en paramètre.

### 15.4.6 Location

Il s'agit de toutes les informations contenues dans l'URL de la page en cours. Voici la propriété et les méthodes de cet objet, qui se déclarent selon la syntaxe objet. L'objet `window.location` renvoie l'URL complète de la page en cours. On utilise la syntaxe suivante :

```
variable = window.location.méthode() ;
variable = window.location.propriété ;
```

Les propriétés suivantes renvoient des informations plus précises concernant l'URL. Voici leur liste :

- `hash`
- `host`
- `hostName`
- `pathName`
- `href`
- `port`
- `protocol`
- `search` : renvoie la partie de l'URL située après le « `?` ».

Il existe deux méthodes de l'objet `window.location`, qui concernent toutes deux le rechargement de la page. Ce sont les suivantes :

- `reload ()` : recharge la page. Ne fonctionne pas sous tous les navigateurs.
- `replace (page)` : recharge la page en cours sans modifier l'historique.

### 15.4.7 Screen

Il s'agit de toutes les informations du système d'affichage de l'utilisateur. Il y a 4 propriétés rattachées à cet objet. Voici la syntaxe à utiliser :

```
variable = window.screen.propriété ;
```

Il y a 5 propriétés qui concernent l'écran et l'affichage. Elles peuvent être utiles quant à l'affichage de votre page. Les voici :

- `availHeight` : hauteur en pixels de la zone utilisable pour l'affichage.
- `availWidth` : largeur en pixels de la zone utilisable pour l'affichage.
- `height` : hauteur de la fenêtre en pixels - contient barres de menus, d'état, de titre et de scrolling.
- `width` : largeur de la fenêtre en pixels - contient barres de menus, d'état, de titre et de scrolling.
- `colorDepth` : Contient la profondeur de couleur en nombre de bits.

### 15.4.8 Event

Il s'agit d'un objet propre à Microsoft Internet Explorer. Il renvoie le type d'événement qui a eu lieu. Sa syntaxe est la suivante :

```
variable = window.event.propriété ;
```

Les deux propriétés de cet objet concernent – bien entendu – les évènements qui ont lieu sur la page. Ce sont les suivantes :

- `altKey` : renvoie le code du caractère frappé au clavier.
- `button` : renvoie le type de clic de souris effectué.

### 15.4.9 External

Il s'agit aussi d'un objet propre à Microsoft Internet Explorer. Il permet d'accéder aux propriétés du navigateur. On utilise la syntaxe habituelle :

```
variable = window.external.propriété ;
```

Il possède une seule propriété qui s'avère utile pour inciter le visiteur à revenir. Il s'agit de :

- `AddFavorite(URL,titre)` : Ajoute une ligne à la liste des favoris. Demande confirmation à l'internaute.

## 15.5 Exemple concret

Dans l'exemple suivant, le but est d'afficher les attributs de la fenêtre (hauteur/largeur, place affichable) ainsi que l'URL en cours. Il n'y a pas de commentaires car les instructions utilisées sont détaillées ci-dessus.

**Exemple 15.3 – Les objets de window**

```
<script language="Javascript">
document.write (window.location + "<br>");
document.write (window.screen.availHeight + "<br>");
document.write (window.screen.availWidth + "<br>");
document.write (window.screen.height + "<br>");
document.write (window.screen.width + "<br>");
</script>
```

## 15.6 Exercice

Le but de l'exercice est de donner à l'utilisateur la possibilité de modifier la fenêtre à l'aide de boutons. Il pourra redimensionner et revenir à la taille initiale, la déplacer et la remettre en position initiale, lui enlever le focus, et la fermer. Lors de la fermeture, l'utilisateur sera averti par une boite de message.

# 16 OBJET DOCUMENT

Il s'agit – comme son nom l'indique – de la page en cours d'exécution. Tous les éléments de la page sont des propriétés ou des méthodes de document. L'objet document étant intégré dans le langage, il n'est pas nécessaire de créer une instance de cette classe.

L'objet document fait partie de l'objet `window`, mais il n'est pas nécessaire de préciser le suffixe "`window`."

## 16.1 Propriétés

Toutes ces propriétés correspondent à des éléments de la page ouverte. Cela permet d'uniformiser et de changer quelques détails de la page. On les utilise grâce à la syntaxe habituelle :

```
document.propriété;
```

La liste des propriétés est longue ; elle comprend tout ce qui concerne la page en cours. Voici leur liste :

- `bgColor` : couleur du fond.
- `fgColor` : couleur du texte.
- `linkColor` : couleur des liens ni actifs ni visités.
- `alinkColor` : couleurs des liens actifs.
- `vlinkColor` : couleurs des liens visités.
- `cookie` : chaîne de caractères contenant les cookie.
- `lastModified` : date de dernière modification du fichier source
- `referrer` : adresse de la page par laquelle la page en cours a été appelée.
- `title` : titre du document, indiqué par les balises `<title>…</title>`. N'est modifiable qu'avec Microsoft Internet Explorer.
- `fileSize` : taille de la page en octets.
- `defaultCharset` : jeu de caractère du document chargé.
- `mimeType` : type du document chargé.
- `URLUnencoded` : URL complète de la page, avec les caractères spéciaux encodés.
- `URL` : URL de la page.
- `protocol` : protocole de chargement de la page.
- `domain` : domaine de l'URL complète de la page.

L'exemple qui suit permet d'afficher à l'écran toutes les informations sur la page courante. Cet affichage a une utilité faible, mais ces propriétés peuvent être utiles dans un script de test.

**Exemple 16.1 – Propriétés de l'objet Document**

```
<script language="Javascript">
document.write ("Taille du fichier : " + document.fileSize + "<br>");
document.write ("Type mime : " + document.mimeType + "<br>");
document.write ("Jeu de caractères : " + document.defaultCharset + "<br>");
document.write ("URL décodée : " + document.URLUnencoded + "<br>");
document.write ("URL : " + document.URL + "<br>");
document.write ("Protocole : " + document.protocol + "<br>");
document.write ("Dernière modification : " + document.lastModified +
"<br>");
</script>
```

## 16.2 Méthodes

L'objet document possède de plusieurs méthodes, dont certaines qui ont été évoquées déjà vu précédemment. Voici la syntaxe à suivre :

variable = document.*méthode()*;

Elles permettent d'agir sur la page affichée et de savoir ce qui est affiché. Voici la liste de ces méthodes :

- `write ('texte')` : affiche le texte et le code HTML dans la page courante.
- `getSelection ()` : renvoie le texte qui a été sélectionné dans la page.
- `handleEvents()` : créé un gestionnaire d'évènement.
- `captureEvents()` : détecte un évènement.
- `open ()` : ouvre une nouvelle fenêtre de votre browser.
- `close ()` : ferme le flux d'affichage externe.
- `getElementById(ID)` : renvoie un objet HTML en fonction de son ID. A ne pas confondre avec le Name.
- `getElementsByName(nom)` : renvoie un objet HTML en fonction de son name.
- `getElementsByTagName(type)` : renvoie un tableau de toutes les balises HTML du type passé en paramètre.

L'exemple suivant réécrit dans une zone de texte les valeurs des lignes de textes au-dessus, séparées par des espaces.

**Exemple 16.2 – Méthodes de l'objet Document**

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
<HTML>
<HEAD>
<TITLE>New Document</TITLE>
<script language="Javascript">
function f() {
	var tab = document.getElementsByTagName ("input");
	var result = '';
	for (i = 0; i < tab.length-1; i++) {
		result = result + (tab[i].value) + " ";
		}
	document.form1.result.value = result;
	}
</script>
</HEAD>
<BODY>
<form name="form1">
<input type="text" name="1" value="1"><br/>
<input type="text" name="2" value="2"><br/>
<input type="text" name="3" value="3"><br/>
<input type="text" name="4" value="4"><br/>
<input type="text" name="5" value="5"><br/>
<input type="button" value="click!" name="click"
onClick="f();"><br/>
<textarea rows=4 cols=40 name="result"></textarea>
</form>
</BODY>
</HTML>
```

## 16.3 Evènements

On peut rattacher certains évènements à l'objet document. Ils concernent les actions de la souris et du clavier :

- `onClick` : clic de souris sur un élément de la page.
- `onDblClick` : le double clic de souris.
- `onKeyPress` : la frappe d'une touche de clavier.

## 16.4 Objets

L'objet document contient plusieurs autres objets assez réduits qui sont présentés dans cette partie. On les utilise à l'aide de cette syntaxe :

```
document.objet.propriété ;
document.objet.méthode() ;
```

Il est utile de préciser que tout élément de la page est en soit un objet de document. On y accède selon la syntaxe ci-dessus. Il serait trop lourd de donner la liste des éléments, qui sont répertoriés dans tout cours HTML digne de ce nom.

D'autres objets existent et permettent de « naviguer » dans la page et ses éléments.

### 16.4.1 All

Il s'agit d'un tableau contenant tous les calques déclarés dans la page dans les balises `<div>…</div>`. Il s'agit d'une particularité de Microsoft Internet Explorer. Il possède les propriétés de la balise `<div>`. Voici la syntaxe à suivre :

```
document.all["calque"].style.propriété = x ;
```

### 16.4.2 Layers

C'est l 'équivalent de l'objet all pour Netscape, pour les calques des balises  ou <layer>. Il possède les propriétés de la balise `<div>`. La syntaxe est la suivante :

```
document.layers["calque"].style.propriété = x ;
```

### 16.4.3 Forms

Il s'agit d'un tableau contenant tous les formulaires du document. Il possède une propriété – `elements[]` – qui est un tableau de tous les éléments du formulaire.

```
document.forms[i].elements[j] ;
```

### 16.4.4 Anchors

Il s'agit d'un tableau contenant toutes les ancres – les balises `<a>` - de la page courante. Il ne possède ni propriétés ni méthodes. Chaque élément du tableau possède les propriétés d'une balise `<a>`.

```
document.anchors[i] ;
```

### 16.4.5 Images

Il s'agit d'un tableau contenant toutes les images – les balises `<img>` - de la page courante. Il ne possède ni propriétés ni méthodes. Cela permet de faire des effets, par exemple des rollovers, sur les images. Chaque élément possède les propriétés et méthodes de la balise `<img>`.

```
document.images[i] ;
```

### 16.4.6 Applets

Il s'agit d'un tableau contenant toutes les applets java déclarées dans la page courante. Il ne possède ni propriétés ni méthodes.

```
document.applets[i] ;
```

### 16.4.7 Plugins

Il s'agit du tableau plugins, qui n'est reconnu que par Netscape. Il s'agit de la liste de tous les plugins installés. Il est aussi rattaché à l'objet `navigator`. Pour ses propriétés, voir la section concernant les objets de `navigator`.

### 16.4.8 Frames

Il s'agit d'un tableau contenant toutes les frames déclarées dans la page courante. Il ne possède ni propriétés ni méthodes. Chaque élément est un objet `window` et a les mêmes méthodes et propriétés.

```
document.frames[i] ;
```

## 16.5 Exercice

Le but est de créer un formulaire avec un texte, dans lequel on pourra entrer une couleur dans une zone de texte et avec un click, on change soit la couleur du texte, soit la couleur du fond.

# 17 OBJETS DU NOYAU JAVASCRIPT

## 17.1 Généralités

Le langage JS n'est pas un langage orienté objet, mais il possède une partie des concepts d'un langage objet comme le C++. Ce concept[13] a été abordé précédemment. Certains de ces objets ne sont pas inconnus, notamment l'objet `boolean`, et l'objet `Array`. Ci-après les 4 autres objets intégrés au JS seront décrits. Il s'agit de l'objet `Math`, de l'objet `String`, de l'objet `Date` et de l'objet `Image`.

## 17.2 Quelques précisions

Certains de ces objets seront déjà définis, comme l'objet `Math`. Il ne sera alors pas nécessaire de les déclarer. L'objet `String` est défini par une variable – il s'agit des chaînes de caractères – et n'a pas besoin de constructeur. Les objets `Date` et `Image` nécessitent un constructeur, intégré au langage, qu'il faut appeler selon la syntaxe habituelle. Certains objets `Image` existent déjà dans une page, il s'agit des balises `<img>`. Ces dernières n'ont donc pas besoin de faire appel au constructeur.

[13] Pour les étourdis, il s'agit du chapitre 8. NOTION OBJET

# 18 OBJET MATH

## 18.1 Propriétés

Ces propriétés ne sont rien d'autre que des constantes mathématiques, qui s'utilisent selon la syntaxe objet habituelle. Voici leur liste :

- `Math.PI`
- `Math.LN2`
- `Math.LN10`
- `Math.E`
- `Math.LOG2E`
- `Math.LOG10E`

## 18.2 Méthodes

Les fonctions mathématiques usuelles ont été transcrites en langage JS à travers les méthodes de l'objet `Math`. La plupart des fonctions s'utilisent selon la même syntaxe :

```
var1 = math.fonction(paramètres) ;
```

### 18.2.1 Fonctions diverses

Ces fonctions permettent d'effectuer des calculs classiques. Ces fonctions étant simples, aucun exemple ne sera fourni.

- `abs(x)` : renvoie la valeur absolue de x.
- `ceil(x)` : renvoie l'entier supérieur à x.
- `floor(x)` : renvoie l'entier inférieur à x.
- `round(x)` : renvoie l'entier le plus proche de x.
- `max(x,y)` : renvoie le plus grand nombre entre x et y.
- `min(x,y)` : renvoie le plus petit nombre entre x et y.
- `pow(x,y)` : renvoie le nombre x à la puissance y.
- `random(x,y)` : renvoie un nombre aléatoire entre 0 et 1.
- `sqrt(y)` : renvoie la racine carrée de x.

### 18.2.2 Fonctions trigonométriques

- `sin(x)`
- `asin(x)`
- `cos(x)`
- `acos(x)`
- `tan(x)`
- `atan(x)`

### 18.2.3 Fonctions logarithmiques

- `exp(x)`
- `log(x)`

## 18.3 Simplification

Lorsque l'objet Math est beaucoup utilisé dans une portion de code, l'écriture deviendra vite pénible. Il est possible de ne pas avoir à écrire le mot `math` à chaque fois. Il suffit d'encadrer la zone de code par des accolades et l'expression `with(math)` comme ceci :

```
with (math) {
code...
}
```

**Exemple 18.1 – Simplification objet**

```
with (Math) {
x = sqrt (238) ;
y = sin (x) ;
document.write(y) ;
}
```

## 18.4 Exercice

L'exercice est relativement simple. Il consiste à afficher la racine carrée de tous les x entiers, compris dans [0 ; 20]. Le résultat doit être arrondi à l'entier.

# 19 OBJET STRING

## 19.1 Généralités

Un objet `string` est une chaîne de caractères. Il possède une propriété et 7 méthodes. Cette classe permet la manipulation des caractères qui s'avère très utile en JS. Il est à préciser que l'objet `string` ne se construit pas comme les autres objets. Une chaîne de caractère est en soi un objet `string`. Il suffit donc de déclarer une variable et de lui affecter une valeur chaîne de caractères.

## 19.2 Propriété

Il s'agit de la longueur de la chaîne, `length`. Elle s'utilise de deux façons différentes :

```
variable1 = variable2.length ;
variable = ("chaine").length ;
```

**Exemple 19.1 – Propriété length**

```
x = "Mon château" ;
y = x.length ;
document.write(y) ;
```

## 19.3 Méthodes

### 19.3.1 CharAt()

Cette méthode renvoie le caractère situé à la position x fournie en paramètre. Le numéro de la position est compris entre 0 et la longueur de la chaîne –1. Voici la syntaxe de cette méthode :

```
chaine1 = chaine2.charAt(x) ;
chaine1 = ("chaine").charAt(x) ;
chaine1 = charAt(chaine2,x) ;
```

**Exemple 19.2 – String.charAt()**

```
x = "Mon Château" ;
y = x.charAt(4) ;
document.write(y) ;
```

### 19.3.2 FromCharCode()

Cette méthode renvoie les nombres ASCII passés en paramètres sous forme de chaîne de caractère. En voici la syntaxe :

```
chaine.fromCharCode(i1,i2,i3)
```

**Exemple 19.3 – String.FromCharCode()**

```
document.write(y.formCharCode(12,105,123,104)) ;
```

### 19.3.3 CharCodeAt()

Cette méthode renvoie le code ASCII du caractère présent à la position indiquée en paramètre.

```
variable = chaine1.charCodeAt(position) ;
```

**Exemple 19.4 – String.charCodeAt()**

```
y = "lepape@le-vatican.com";
document.write(y.charCodeAt(6)) ;
```

### 19.3.4 IndexOf()

Cette méthode renvoie la première position d'une chaîne partielle dans une autre chaîne en partant de gauche, à partir de la position x indiquée en paramètre. Si elle n'est pas présente, la méthode renvoie –1. Le numéro de la position est compris entre 0 et la longueur de la chaîne –1. La syntaxe à suivre est :

```
variable=chaine.indexOf (chaine_partielle, x)
```

**Exemple 19.5 – String.indexOf()**

```
x = "maman" ;
y = x.indexOf("ma",0) ;
document.write(y) ;
```

### 19.3.5 LastIndexOf()

Cette méthode renvoie la première position d'une chaîne partielle dans une autre chaîne en partant de gauche, à partir de la position x indiquée en paramètre. Si elle n'est pas présente, la méthode renvoie –1. Le numéro de la position est compris entre 0 et la longueur de la chaîne –1. La syntaxe est :

```
variable = chaine.lastIndexOf (chaine_partielle, x)
```

**Exemple 19.6 – String.lastIndexOf()**

```
x = "maman" ;
y = x.lastIndexOf("ma",4) ;
document.write(y) ;
```

### 19.3.6 Substring()

Cette méthode renvoie la sous-chaîne comprise entre les positions x et y indiquées en paramètres, dans la chaîne principale. Il faut utiliser :

```
variable = chaine.substring (x,y)
```

**Exemple 19.7 – String.substring()**

```
x = "maman" ;
y = x.substring(1,3) ;
document.write(y) ;
```

### 19.3.7 Substr ()

Cette méthode renvoie le texte une sous-chaîne de la chaîne de la méthode, à partir du début et sur n caractères. Voici la syntaxe :

```
variable = chaine1.substr(début, n)
```

**Exemple 19.8 – String.substr()**

```
y = "lepape@le-vatican.com";
document.write(y.substr(5,2)) ;
```

### 19.3.8 Slice()

Elle est équivalente à `substring()`. Le paramètre qui indique la fin de la sous-chaîne est facultatif. S'il n'est pas précisé, la sous-chaîne se termine à la fin de la chaîne. En voici la syntaxe :

```
variable = chaine.slice(début, fin)
```

**Exemple 19.9 – String.slice()**

```
y = "lepape@le-vatican.com";
document.write(y.slice(7)) ;
```

### 19.3.9 Split()

Cette méthode renvoie un tableau de sous-chaînes découpées selon le séparateur passé en paramètres.

```
variable = chaine.split(séparateur)
```

**Exemple 19.10 – String.split()**

```
x = "lepape@le-vatican.com" ;
y = x.split("@") ;
document.write(y[0] + "<br>" + y[1]) ;
```

### 19.3.10 Concat()

Cette méthode renvoie la concaténation de la chaîne passée en paramètre avec celle de la méthode.

```
variable = chaine1.concat(chaine2)
```

**Exemple 19.11 – String.concat()**

```
x = "Ecrivez-moi à " ;
y = "lepape@le-vatican.com";
z = x.concat(y) ;
document.write(z) ;
```

### 19.3.11 ToLowerCase()

Cette méthode renvoie la chaîne avec tous les caractères en minuscules. Voici comment elle s'utilise :

```
variable = chaine.toLowerCase()
```

**Exemple 19.12 – String.toLowerCase()**

```
x = "MAMAN" ;
y = x.toLowerCase() ;
document.write(y) ;
```

### 19.3.12 ToUpperCase()

Cette méthode renvoie la chaîne avec tous les caractères en majuscules. La syntaxe est la suivante :

```
variable = chaine.toUpperCase()
```

**Exemple 19.13 – String.toUpperCase()**

```
x = "Maman" ;
y = x.toUpperCase() ;
document.write(y) ;
```

### 19.3.13 FontColor()

Cette méthode renvoie le texte de l'objet en y ajoutant les balises `<font color=couleur>…</font>`. Elle s'utilise ainsi :

```
variable = chaine1.fontColor(couleur) ;
```

**Exemple 19.14 – String.fontColor()**

```
y = "lepape@le-vatican.com";
document.write(y.fontColor("blue")) ;
document.write(y.fontColor("red")) ;
```

### 19.3.14 FontSize()

Cette méthode renvoie le texte de l'objet en y ajoutant les balises `<font size=taille>…</font>`.

```
variable = chaine1.fontSize(taille)
```

**Exemple 19.15 – String.fontSize()**

```
y = "lepape@le-vatican.com";
document.write(y.fontSize("16")) ;
document.write(y.fontSize("8")) ;
```

### 19.3.15 Fixed()

Cette méthode renvoie le texte de l'objet en y ajoutant les balises `<pre>…</pre>`. La syntaxe est :

```
variable = chaine1.fixed()
```

**Exemple 19.16 – String.fixed()**

```
y = "lepape@le-vatican.com";
document.write(y.fixed()) ;
```

### 19.3.16 Bold()

Cette méthode renvoie le texte de l'objet en y ajoutant les balises `<b>…</b>`.Voici la syntaxe :

```
variable = chaine1.bold()
```

**Exemple 19.17 – String.bold()**

```
y = "lepape@le-vatican.com";
document.write(y.bold()) ;
```

### 19.3.17 Strike()

Cette méthode renvoie le texte de l'objet barré. A utiliser avec précautions. Sa syntaxe est la suivante :

```
variable = chaine1.strike()
```

**Exemple 19.18 – String.strike()**

```
y = "lepape@le-vatican.com";
document.write(y.strike()) ;
```

### 19.3.18 Sub()

Cette méthode renvoie le texte de l'objet en y ajoutant les balises <sub>…</sub>.

```
variable = chaine1.sub()
```

**Exemple 19.19 – String.sub()**

```
y = "lepape@le-vatican.com";
document.write(y.sub()) ;
```

### 19.3.19 Big()

Cette méthode renvoie le texte de l'objet en y ajoutant les balises <big>…</big>.

```
variable = chaine1.big()
```

**Exemple 19.20 – String.big()**

```
y = "lepape@le-vatican.com";
document.write(y.big()) ;
```

### 19.3.20 Sup()

Cette méthode renvoie le texte de l'objet en y ajoutant les balises <sup>…</sup>.

```
variable = chaine1.sup()
```

**Exemple 19.21 – String.sup()**

```
y = "lepape@le-vatican.com";
document.write(y.sup()) ;
```

### 19.3.21 Blink()

Cette méthode renvoie le texte de l'objet en y ajoutant les balises <blink>…</blink>. Ne fonctionne que sous Netscape Navigator.

```
variable = chaine.blink()
```

**Exemple 19.22 – String.blink()**

```
y = "lepape@le-vatican.com";
document.write(y.blink()) ;
```

### 19.3.22 Small()

Cette méthode renvoie le texte de l'objet en y ajoutant les balises <small>…</small>.

```
variable = chaine.small()
```

**Exemple 19.23 – String.small()**

```
y = "lepape@le-vatican.com";
document.write(y.small()) ;
```

### 19.3.23 Italics()

Cette méthode renvoie le texte de l'objet en y ajoutant les balises …. Sa syntaxe est :

```
variable = chaine.italics()
```

**Exemple 19.24 – String.italics()**

```
y = "lepape@le-vatican.com";
document.write(y.italics()) ;
```

### 19.3.24 Link()

Cette méthode renvoie le texte de l'objet en y ajoutant les balises <a href=URL >…</a>.

```
variable = chaine1.link(URL)
```

**Exemple 19.25 – String.link()**

```
y = "lepape@le-vatican.com";
document.write(y.link("http://www.google.fr"));
```

### 19.3.25 Anchor()

Cette méthode crée un ancre, et renvoie le texte de l'objet en y ajoutant les balises <a name=ancre >…</a>.

```
variable = chaine1.anchor(ancre)
```

**Exemple 19.26 – String.anchor()**

```
y = "lepape@le-vatican.com"
document.write(y.anchor("ancre")) ;
```

## 19.4 Manipulations sur les chaînes de caractères

La manipulation des chaînes demande certaines connaissances syntaxiques. Ces dernières ont déjà été précisées dans la section concernant les structures de données, mais elles doivent être détaillées. Elles concernent l'affectation, la concaténation, et les caractères spéciaux.

### 19.4.1 Affectation

Les chaînes de caractères se présentent sous deux formes. Elles sont soient encadrées de quotes ' ', soient encadrées de guillemets " ". On affecte les chaînes à leur variable comme toute variable.

**Exemple 19.27 – Affectation**

```
x = "Maman" ;
y = 'Maman' ;
```

Dans l'exemple ci-dessus, les chaînes x et y sont équivalentes. Elles contiennent la même donnée.

### 19.4.2 Concaténation

Il est possible d'ajouter des chaînes – de les mettre à la suite l'une de l'autre – grâce à l'opérateur « + ».

**Exemple 19.28 – Concaténation**

```
x = "Maman" ;
y = 'Papa' ;
z = x + " " + y ;
document.write (z);
```

Dans les fonctions, comme la méthode `document.write()`, il est possible d'ajouter ces chaînes à l'aide de l'opérateur « + » ou « , ».

**Exemple 19.29 – Concaténation**

```
x = "Maman" ;
y = 'Papa' ;
document.write (x + " " + y);
document.write (x , " " , y);
```

### 19.4.3 Caractères spéciaux

Certains caractères permettent de faire un effet dans l'affichage, d'autres doivent être précédés du caractère « \ ». Ils sont répertoriés dans le tableau suivant.

| Caractère | Affichage |
|---|---|
| \b | touche de suppression |
| \f | formulaire plein |
| \n | retour à la ligne |
| \r | appui sur la touche *ENTREE* |
| \t | tabulation |
| \" | guillemets |
| \' | apostrophes |
| \\ | caractère antislash |

**TAB. 12 – Caractères spéciaux**

Les autres caractères spéciaux – si on peut les appeler ainsi – sont les balises HTML. En effet, celles-ci peuvent être insérées dans les chaînes de caractère, et seront interprétées comme des balises par le navigateur lors de l'écriture avec la méthode `document.write()`.

**Exemple 19.30 – Caractères spéciaux**

```
x = "Maman" ;
y = 'Papa' ;
document.write (x + "<br>" + y);
```

## 19.5 Exercice

Dans cet exercice, il faut créer un formulaire avec un bouton et une zone de texte. L'internaute entre une série de mots séparés par des espaces. Ensuite, on récupère ces mots, et on les sépare par des virgules. Enfin, on les affiche à l'écran dans différents formats avec `document.write()`.

# 20 OBJET DATE

## 20.1 Généralités

La date et l'heure sont regroupées en JS dans la classe `Date`. On créé alors une variable `Date` grâce au constructeur, comme ceci :

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

La date et l'heure en JS ne commencent qu'à partir du 1er janvier 1970, 0h 0min 0s. Toute référence à une date antérieure donnera un résultat aléatoire.

## 20.2 Méthodes

### 20.2.1 Type Get

Une fois une variable `Date` créée, elle possède les informations de la date courante. Il faut cependant récupérer ces informations pour pouvoir les exploiter. Les fonctions suivantes remplissent ce rôle en récupérant certaines informations de la variable `Date`. La syntaxe générale à suivre est la suivante :

```
variable1 =new Date()
variable2 = variable1.getInfo();
```

Les méthodes permettent chacune d'accéder à une information particulière de la date. Voici leur liste :

- `getYear()` : Retourne les 2 derniers chiffres de l'année. Il faudra donc rajouter le préfixe "20".
- `getFullYear()` : Retourne la date sur 4 chiffres.
- `getMonth()` : Retourne le mois sous la forme d'un entier compris entre 0 et 11.
- `getDate()` : Retourne le jour du mois sous forme d'un entier compris entre 1 et 31.
- `getDay()` : Retourne le jour de la semaine sous forme d'un entier compris entre 0 et 6.
- `getHours()` : Retourne l'heure sous forme d'un entier compris entre 0 et 23.
- `getMinutes()` : Retourne les minutes sous forme d'un entier compris entre 0 et 59.
- `getSeconds()` : Retourne les secondes sous forme d'un entier compris entre 0 et 59.
- `getMilliseconds()` : retourne les millisecondes de la date. A ne pas confondre avec `getTime()`.

### 20.2.2 Type Set

Il est aussi possible de remplir la variable Date avec des données personnelles. Les fonctions suivantes remplissent la variable – qui est une chaîne – `Date` avec les données désirées. Elles s'utilisent toutes de la même façon :

```
variable1 = new Date()
variable1.setInfo(paramètre);
```

Chacune des méthodes prend en paramètre l'information qui sera introduite dans la variable Date. La liste de ces méthodes est ci-dessous :

- `setYear()` : Assigne les 2 derniers chiffres de l'année, sous forme d'un entier supérieur à 1900.
- `setFullYear()` : Assigne l'année sur 4 chiffres.
- `setMonth()` : Assigne le mois sous la forme d'un entier compris entre 0 et 11.
- `setDate()` : Assigne le jour du mois sous forme d'un entier compris entre 1 et 31.
- `setDay()` : Assigne le jour de la semaine sous forme d'un entier compris entre 0 et 6.
- `setHours()` : Assigne l'heure sous forme d'un entier compris entre 0 et 23.
- `setMinutes()` : Assigne les minutes sous forme d'un entier compris entre 0 et 59.
- `setSeconds()` : Assigne les secondes sous forme d'un entier compris entre 0 et 59.
- `setMilliseconds()` : assigne les millisecondes de la date. A ne pas confondre avec `setTime()`.

### 20.2.3 L'heure

L'heure est très utile en JS, elle possède donc plusieurs méthodes. La syntaxe est la suivante :

```
variable1 =new Date();
variable1.méthode();
```

Ces méthodes permettent de manipuler toutes sortes de données concernant l'heure. En voici la liste :

- `getTime()` : Renvoie l'heure courante sous forme d'un entier représentant le nombre de millisecondes depuis le 1er janvier 1970 00h 00min 00s.
- `getTimezoneOffset()` : Renvoie la différence entre l'heure locale et l'heure GMT sous forme d'un entier en minutes.
- `setTime()` : Assigne l'heure courante sous forme d'un entier représentant le nombre de millisecondes depuis le 1er janvier 1970 00h 00min 00s.

- `toGMTString()` : Renvoie la valeur actuelle de la variable `Date` en heure GMT.
- `toLocaleString()` : Renvoie la valeur actuelle de l'heure de la variable `Date`. C'est plus rapide que de combiner `getHours()`, `getMinutes()`, et `getSeconds()`.

## 20.3 Exemple

Le but de cet exemple est d'afficher une horloge dans une ligne de texte. Pour cela, on utilise la méthode `window.setTimeout()`, qui rappelle la fonction d'affichage toute les secondes.

**Exemple 20.1 – Date**

```
<HTML><HEAD>
<script language="Javascript">
//la fonction que l'on doit appeler
function GetTime () {
       var time = new Date (); // objet Date()
       var hours = time.getHours();//on récupère les
heures
       var min = time.getMinutes(); //on récupère les
minutes
       var sec = time.getSeconds();//on récupère les
secondes
/* on rajoute un 0 si le chiffre est inférieur à 10 */
       if (hours < 10) hours = "0" + hours;
       if (min < 10) min = "0" + min;
       if (sec < 10) sec = "0" + sec;
/* affichage de l'heure dans une zone de texte */
       document.time.heure.value = hours + ":" + min +
":" + sec;
       window.setTimeout('GetTime()',1000); /* le timer
rappelle la fonction toutes les secondes */
       }
</script>
</HEAD>
<BODY onLoad="GetTime();">
<form name="time">
Voici l'heure :
<!-- la zone de texte qui sert à l'affichage -->
<center><input type="text" name="heure" value=""
size=6></center>
</form>
</BODY></HTML>
```

## 20.4 Exercice

### 20.4.1 Nombre de jours

Le but de l'exercice est de calculer le nombre de jours écoulés depuis l'an 2000.

### 20.4.2 Heure GMT

Cet exercice consiste à donner l'heure GMT de deux façons différentes.

# 21 OBJET IMAGE

## 21.1 Rappel HTML

Il semble utile de rappeler que la balise `<img>` possède de nombreux attributs dont certains qu'il est conseillé de déclarer en Javascript. Ci-dessous, la liste de ces attributs :

- `src` : URL, souvent relative, de l'image.
- `name` : nom de l'image dans la page, très important en JS.
- `width` : largeur de l'image affichée.
- `height` : hauteur de l'image affichée.
- `align` : alignement de l'image par rapport au texte.

Il est important de rajouter qu'une image peut servir d'ancre pour un lien hypertexte, c'est-à-dire être placée entre les balises `<a href>…</a>`. Cela permet certaines astuces d'affichage.

## 21.2 L'objet

La classe Image correspond à la balise HTML `<img>`. Son emploi est assez difficile et ce cours ne décrira pas en détail toutes ses facettes. Il permet de manipuler les images de façon dynamique, ce qui est impossible avec le HTML. On rappellera que les images sont stockées dans le tableau `images[]` de l'objet document.

Grâce à un objet Image, on peut pré charger une image et la stocker en cache, contrairement au HTML. L'image ne sera cependant pas affichée. Elle le sera à l'aide de la balise `<img>`.

Un objet Image est appelé selon la syntaxe objet habituelle, avec le constructeur `Image()`. L'objet possède alors les propriétés de la balise HTML `<img>`, dont la liste figure dans la section suivante.

## 21.3 Propriétés

### 21.3.1 Syntaxe

Un objet Image possède plusieurs propriétés, que l'on peut assimiler aux attributs de la balise `<img>`. La syntaxe est la suivante :

```
variable = new Image();
variable.propriété = x ;
var2 = variable.propriété ;
```

### 21.3.2 Src

Il s'agit de l'URL absolue ou relative le l'image. Elle peut être modifiée. Cela permet de charger en cache l'image lors du chargement de la page lorsqu'il s'agit d'un objet créé en JS. Lorsque cette propriété est appliquée à une balise `<img>`, cela permet de changer l'image affichée par la balise, comme dans l'exemple ci-dessous :

**Exemple 21.1 – Image.src**

```
<img src="0.gif" name="image1"
onMouseOver="document.image1.src='2.gif';"
onMouseOut="document.image1.src='0.gif';">
```

### 21.3.3 Name

C'est le nom défini par l'attribut `name="…"` de la balise `<img>`. A ne pas confondre avec l'ID. Permet de trouver l'image dans le tableau `document.images[]`.

**Exemple 21.2 – Image.name**

```
<img src="0.gif" name="image1"
onMouseOver="alert('Adresse de l\'image : ' +
document.images['image1'].src);">
```

### 21.3.4 Id

C'est l'ID défini par l'attribut `id="…"` de la balise `<img>`. A ne pas confondre avec le nom. Permet de retrouver l'image grâce à la méthode `document.getElementById()`.

**Exemple 21.3 – Image.id**

```
<img src="0.gif" id="image1"
onMouseOver="alert('Adresse de l\'image : ' +
document.getElementById('image1').src);">
```

### 21.3.5 Width

Il s'agit de la largeur de l'image. Elle contient la valeur de l'attribut `width` de la balise `<img>`. Si cet attribut n'est pas précisé, elle contient la largeur réelle de l'image. Ne peut être modifiée.

**Exemple 21.4 – Image.width**

```
<img src="0.gif" id="image1"
onMouseOver="alert('Largeur de l\'image : ' +
document.getElementById('image1').width);">
```

### 21.3.6 Height

Il s'agit de la hauteur de l'image. Elle contient la valeur de l'attribut `height` de la balise `<img>`. Si cet attribut n'est pas précisé, elle contient la hauteur réelle de l'image. Ne peut être modifiée.

**Exemple 21.5 – Image.height**

```
<img src="0.gif" id="image1"
onMouseOver="alert('Hauteur de l\'image : ' +
document.getElementById('image1').height);">
```

### 21.3.7 Complete

C'est un booléen qui indique si le chargement de l'image est terminé. Renvoie `true` si le chargement est achevé et `false` dans le cas contraire.

**Exemple 21.6 – Image.complete**

```
<img src="0.gif" id="image1"
onMouseOver="alert('chargement complété ? ' +
document.getElementById('image1').complete);">
```

### 21.3.8 Alt

Elle contient le texte qui affiché avant le chargement de l'image et en info bulle lorsqu'elle est survolée. Contient la valeur de l'attribut `alt="…"` de la balise `<img>`. Ne peut être modifiée.

**Exemple 21.7 – Image.alt**

```
<img src="0.gif" name="image1" alt="Image"
onMouseOut="alert('alt : ' +
document.getElementById('image1').alt);">
```

### 21.3.9 FileSize

Elle contient la taille en octets de l'image. N'est accessible que lorsque le chargement est terminé, d'où l'utilité de la propriété `complete`.

**Exemple 21.8 – Image.fileSize**

```
<img src="0.gif" id="image1" alt="Image"
onMouseOver="alert('Octets : ' +
document.getElementById('image1').fileSize);">
```

## 21.4 Afficher une image

Il suffit de prendre l'élément `<img>` voulu dans la page et de changer sa propriété. Cet élément est assimilé à un objet `Image`, faisant partie de l'objet document.

**Exemple 21.9 – Afficher une image**

```
document.image1.src = 'img2.jpg' ;
```

Dans ce cas précis, l'image change et img2.jpg est affiché dans le champ `image1`. L'intérêt de créer un objet `Image` reste donc discutable, puisqu'on ne l'utilise pas…

L'objet `image` permet de stocker l'image en cache au moment du chargement de la page. Il ne reste plus qu'à trouver le moyen de l'afficher… De plus, on peut créer un tableau d'objets `Image`, ce qui nous facilitera les manipulations ensuite.

**Exemple 21.10 – Afficher une image à l'aide d'un tableau**

```
<html>
<head>
<script language="Javascript">
//tableau
var tab = new Array (4) ;

//remplissage d'images
for (i = 0; i < 4; i++) {
       tab[i] = new Image();
       tab[i].src = i + ".gif";
       }

//fonction d'affichage
function f() {
       for (i = 0; i < 4; i++) {
              document.images[i].src = tab[i].src ;
              }
       }
</script>
</head>
<body>
<img  name="img0"><br/>
<img  name="img1"><br/>
<img  name="img2"><br/>
<img  name="img3"><br/>
<input type="button" name="click" value="Afficher les
images" onClick="f();"> <!—appel des l'affichage-->
</body>
</html>
```

L'exemple ci-dessus permet de charger 5 images et des les afficher dans 5 balises `<img>` différentes grâce au tableau `images[]`.

## 21.5 Exemple concret

Ici, le but est de créer une page avec une image et un bouton. Lorsque l'on clique sur le bouton, l'image change. On a au total 4 images différentes. On utilise un tableau d'objets `Image`, une balise `<img>`, une balise `<input>`, et une fonction. Lorsque les 4 images ont défilé, on recommence à la première. Pour cela, un compteur est utilisé. Voici cet exemple :

**Exemple 21.11 – Clic déclenchant un changement d'image**

```
<HTML>
<HEAD>
<TITLE>New Document</TITLE>
<script language="Javascript">
//création d'un tableau de 4 éléments
var tab_images = new Array(4);
// pour chaque élément
for (i = 0; i < tab_images.length; i++)  {
       tab_images[i] = new Image(); /*on crée un objet Image
        et on lui donne un fichier à charger */
       tab_images[i].src = i + ".gif";
             }
//on initialise un compteur
var nb = 0;
// la fonction qui sera appelée.
function changer()  {
      //incrémenter le compteur
       nb++;
      //si on en est 4, on remet à 0 le compteur
       if (nb == tab_images.length)  {
              nb = 0;
              }
      //affichage de l'image
       window.document.image.src = tab_images[nb].src;
       }
</script>
</HEAD>
<!-- la fonction est appelée au chargement de la page -->
<BODY onload="changer();">
<img src="" name="image"><br><!-- la balise <img> -->
<input type="button" name="bouton" value="cliquez"
onClick="changer();">
</BODY>
</HTML>
```

## 21.6 Exercice

Ici, le but est aussi de créer une page avec une image. Lorsque passe la souris sur l'image, l'image change. Quand on « ressort » de l'image, l'image de base revient. 4 images différentes s'afficheront en un cycle.

# 22 PROGRAMMATION MULTI-CADRES

## 22.1 Rappel HTML

Les frames sont un élément du langage HTML. Il s'agit du partage de la fenêtre en plusieurs cadres où l'on pourra afficher différentes pages HTML. Les frames se déclarent dans le fichier principal avec la balise `<frameset>` suivie de plusieurs balises `<frame>`. Il n'y a alors pas de balise <body> dans la page principale. Pour plus de précisions, consulter un cours HTML.

## 22.2 Frame

Les frames ne sont pas très utilisées en JS, mais il est important de savoir quelques détails en cas d'utilisation de frames au niveau HTML. Il est important, lorsque le JS est utilisé, de préciser le nom de la frame dans la balise `<frame>`.

**Exemple 22.1 – Attribut name des frames**

```
<frame src="fichier.htm" name="cadre1">
```

Ce nom ne paraît pas très utile en HTML, mais il prend toute son importance en JS. Il sera utilisé dans deux cas : la cible des liens ainsi que pour accéder à un élément d'une autre frame de la page.

## 22.3 Lien hypertexte

Lorsque de l'utilisation des frames, il peut s'avérer utile d'afficher une page dans une frame différente de celle où se trouve le lien. C'est là où intervient l'attribut `target` de la balise `<a href…>`. Il permet d'indiquer la frame où l'on veut que la page appelée s'affiche. Voici la syntaxe :

```
<a href="lien" target=”frame”>texte</a>
```

La référence indiquée dans l'attribut `target` peut être de deux types. Dans un premier cas, il s'agit du nom de la frame déclarée dans la page principale, cas où il faut savoir ce nom. Sinon, si on fait référence au cadre parent, on indiquera « `_parent` », et si on fait référence à la fenêtre complète, on indiquera « `_top` ».

**Exemple 22.2 – Attribut target des liens**

```
<a href="lien1.htm" target=”frame1”>cliquez ici</a>
<a href="lien2.htm" target=”_parent”>ou ici</a>
```

## 22.4 Accès aux éléments des frames

Lorsque de l'utilisation des frames, il est souvent nécessaire en JS d'accéder aux éléments des autres frames. L'objet `window` – qui a été décrit précédemment – fournit de quoi le faire. Il contient l'objet `parent`, qui possède les mêmes propriétés que lui.

### 22.4.1 Objet frames[]

La première façon d'accéder aux éléments d'une autre frame se fait bien sûr par le tableau `frames[]` – propriété de parent – dont le numéro des frames est attribué dans l'ordre de déclaration de celles-ci. On accède ensuite à chaque élément de la frame ainsi qu'à ses propriétés et méthodes. La syntaxe est la suivante :

```
parent.frames[i].objet.propriété
parent.frames[i].objet.méthode()
```

**Exemple 22.3 – Objet frames[]**

```
parent.frames[1].form1.action = "get";
```

**Exemple 22.4 – Objet frames[]**

```
parent.frames[1].form1.button1.value = "Click";
```

Dans l'exemple ci-dessus, on accède à la valeur du bouton appelé « `Button1` » du formulaire nommé « `form1` ».

### 22.4.2 Nom de frame

L'autre façon d'accéder aux éléments d'une frame est d'utiliser son nom. Ce sera alors un objet de parent. Il faut donc prendre soin à donner un nom aux frames lors de leur déclaration. On accède ensuite à chaque élément de la frame ainsi qu'à ses propriétés et méthodes.

```
parent.nom.objet.propriété
parent.nom.objet.méthode()
```

Les exemples de la section précédente peuvent être écrits de la façon suivante, sans aucune différence au niveau du résultat.

**Exemple 22.5 – Nom de frame**

```
parent.frame1.form1.action = "get";
```

**Exemple 22.6 – Nom de frame**

```
parent.frame1.form1.button1.value = "Click";
```

## 22.5 Exemple concret

L'exemple 22.7 permet de faire un compteur dans lequel les secondes impaires sont affichées une frame et les secondes paires sur l'autre frame.

**Exemple 22.7 – Compteur sur deux frames**

```
<HTML>
<HEAD>
<TITLE>New Document</TITLE>
<script language="Javascript">
var sec = 0;
var min = 0;
var hrs = 0;
var test = 1;
function f() {
	sec++;
	if (sec == 60) {
		sec = 0;
		min++;
		}
	if (min == 60) {
		min = 0;
		hrs++;
		}
	h = hrs;
	m = min;
	s = sec;
	if (h < 10) h = "0" + h;
	if (m < 10) m = "0" + m;
	if (s < 10) s = "0" + s;
	if (test == 1) {
		parent.frame1.form1.out1.value = h + ":" +
m + ":" + s;
		test = 2;
		}
	else if (test == 2) {
		parent.frame2.form2.out2.value = h + ":" +
m + ":" + s;
		test = 1;
		}
	window.setTimeout('f();',1000);
	}
</script>
</HEAD>
<frameset cols="50%,50%" onLoad="f();">
	<frame src="1.htm" name="frame1">
	<frame src="2.htm" name="frame2">
</frameset>
</HTML>
```

Ci-dessous, les deux fichiers servant de frames dans l'exemple ci-dessus. Elles sont nommées respectivement « 1.htm » et « 2.htm ».

**Exemple 22.8 – Frame 1**

```
/* 1.htm */

<HTML>
<HEAD>
<TITLE>New Document</TITLE>
<script language="Javascript">
</script>
</HEAD>
<body>
<form name="form1">
<center><input type="text" name="out1"
value="00:00:00"></center>
</form>
</body>
</HTML>
```

**Exemple 22.9 – Frame 2**

```
/* 2.htm */

<HTML>
<HEAD>
<TITLE>New Document</TITLE>
<script language="Javascript">
</script>
</HEAD>
<body>
<form name="form2">
<center><input type="text" name="out2"
value="00:00:00"></center>
</form>
</body>
</HTML>
```

## 22.6 Exercice

L'exercice comporte 2 frames, avec chacune un bouton. Chaque clic sur un bouton incrémente la valeur de l'autre bouton.

# 23 COOKIES

## 23.1 Présentation

En JS, il est possible de travailler avec les cookies. Etant donné l'absence de gestion d'écriture/lecture de fichier, les cookies sont le seul moyen de stocker des informations permanentes sur la machine de l'utilisateur. Ces dernières pourront être récupérées plus tard et réutilisées.

Cela permet de compter le nombre de visites de l'internaute, de créer une liste de préférence de navigation sur le site, de conserver le login et le password afin de se connecter directement à un compte… Les applications des cookies sont nombreuses.

Le seul risque de cette méthode est la suppression ou le refus des cookies par l'utilisateur[14].

Le cookie en lui-même se présente sous la forme d'une chaîne de caractère qui contient des informations concaténées : l'information que l'on veut conserver, la date d'expiration, l'auteur – `path` – le nom de domaine, la sécurisation.

## 23.2 Création de cookie

On crée un cookie avec l'objet cookie de l'objet document. Il s'agit d'une chaîne de caractère dans laquelle on indique les informations voulues. Voici la syntaxe :

```
document.cookie = "informations"
```

Dans les informations, il y a tout d'abord la chaîne que l'on souhaite conserver. Ensuite, il faut mettre la date d'expiration, le path, le nome de domaine, et – si besoin – la fait que le cookie soit protégé. Seuls les deux premiers champs sont obligatoires. Voici la syntaxe complète :

```
document.cookie = "variable = contenu ; expires =
date ; path = nom ; domain = domaine ; secure =
true/false" ;
```

Il semble utile de décrire chaque champ indiqué ci-dessus. Si la chaîne est mal écrite, le cookie ne sera pas utilisable, et deviendra par conséquent inutile.

[14] Cette option est disponible dans les menus de votre navigateur. Pour Internet Explorer, ce la se situe dans le menu Outils | Options Internet, onglet Sécurité, dans la rubrique Personnaliser le niveau.

- `information` : il s'agit de ce qui doit être stocké dans le cookie. Il faut définir un nom à la variable et lui affecter une valeur, un contenu. Comme on peut le voir ci-dessus, les champs sont séparés par des points-virgules, il ne faut donc pas insérer des « `;` » dans le contenu.

- `expires` : contient la date d'expiration, à laquelle le cookie sera détruit. La valeur est en secondes. Le plus simple consiste à utiliser un objet Date.

- `path` : le chemin de la page qui a créé le cookie.

- `domain` : le domaine de la page qui a créé le cookie.

- `secure` : booléen qui indique si le cookie doit utiliser le protocole HTTP (false) ou le protocole sécurisé HTTPS (true).

**Exemple 23.1 – Création de cookie**

```
document.cookie = "visite=1; expires=31/12/2004;
path=le-vatican.com/index.php; secure=true";
```

Dans l'exemple ci-dessus, il s'agit d'un cookie créé par « `lepape` », qui expire le 31/12/2004, auquel on peut accéder uniquement par un échange sécurisé, et qui compte le nombre de visites.

## 23.3 Récupération de cookie

La récupération d'un cookie est plus complexe, mais elle est essentielle, car sinon le cookie est inutile. Le but est de trouver le type d'information que l'on cherche, et ensuite de lire la valeur. La maîtrise de l'objet `String` est obligatoire pour ce genre d'exercice. Cette partie explique pas à pas la manière pour récupérer un cookie. On commence par mettre tous les cookies dans une variable, selon le modèle suivant :

```
variable = document.cookie ;
```

Ensuite, on cherche le nom de la « variable » dans la chaîne du cookie. On récupère ce qui est situé entre le signe « = » et le « ; ». Cela semble simple, mais il faut utiliser les méthodes des objets `String` et utiliser des instructions logiques. Ci-dessous, l'exemple montre en détail comment récupérer un cookie.

Cet exemple n'est qu'une manière parmi d'autres. On peut imaginer quantités de façons, mais celle-ci est sûrement une des plus simples.

**Exemple 23.2 – Récupération de cookie**

```
//création du cookie
document.cookie = "visite=1;expires=31/12/2004";
//variable à trouver
var variable_a_trouver = "visite" ;
//on rajoute le signe =
variable_a_trouver = variable_a_trouver + "=" ;
//on récupère le cookie
var chaine = document.cookie ; var longueur =
variable_a_trouver.length ;
//on récupère la longueur à trouver
var resultat ;
/* si le cookie n'est pas vide, on commence à chercher. */
if (chaine.lenght > 0)  {
 /* on cherche la position du début de la variable */
      var debut = chaine.indexOf(chaine_a_trouver,0);
//si on a trouvé cette position, on continue
      if (debut >=0)  {
            var fin ;
/* on rajoute la longueur de la variable, pour arriver au début
du contenu */
            debut = debut + longueur ;
/* on cherche la fin du contenu, c'est-à-dire le premier « ; » */
            fin = chaine.indexOf(";", debut) ;
//si on trouve la position du point-virgule
            if (fin>=0)  {
/* on récupère la chaine située entre le "=" et le ";" */
                  resultat = unescape(chaine.substring(debut,fin) ;
                  }
/* si il n'y a pas de ";", c'est que c'est la fin de la chaîne */
            else  {
/* on récupère la chaine située après le "=" la fin de la chaîne
*/
                  resultat =
                  unescape(chaine.substring(debut,chaine.length) ;
                  }
            }
}
```

## 23.4 Modification de cookie

Modifier un cookie est aussi simple que de le créer. En réalité, il suffit de le recréer avec un contenu différent et une date actualisée.

## 23.5 Suppression de cookie

Pour supprimer un cookie, il faut tout simplement le recréer, avec la même valeur pour éviter de se compliquer les choses, et lui donner une date d'expiration passée. Sinon, il suffit d'attendre sa date d'expiration, ce qui n'est pas toujours satisfaisant.

# 24 PROGRAMMATION OBJET

## 24.1 Présentation

Bien que le Javascript ne soit pas un langage orienté objet, il donne la possibilité de créer ses propres objets. Cette section s'adresse à des programmeurs possédant un niveau avancé[15] car la programmation objet est un point assez difficile.

Les programmeurs C++ seront étonnés de voir que les classes JS sont très simplifiées comparées à celles du C++. Elles ne demandent pas une définition complète de la classe, mais il suffit de déclarer la fonction constructrice.

## 24.2 Déclaration de classe

Contrairement au C++ - qui demande une déclaration détaillée – déclarer une classe en JS se fait simplement en déclarant la fonction constructeur de classe. On déclare à l'intérieur les propriétés à l'aide de l'objet this. On retrouve l'objet `this` découvert précédemment. Comme cela a été expliqué, il désigne l'objet en cours. Voici la syntaxe :

```
function nom_classe ( paramètres ) {
this.propriété = paramètre1 ;
}
```

On déclare l'objet avec la syntaxe habituelle et le mot-clé `new`. Le nome de la classe est en général le type d'objet avec une majuscule.

**Exemple 24.1 – Déclaration d'une classe**

```
function Eleve (Age,Sexe,Ville) {
      this.age = Age ;
      this.sexe = Sexe ;
      this.ville = Ville ;
      }
var Laurent = new Eleve(17,'M','Grenoble') ;
```

[15] Il convient de préciser que ce terme n'est en aucun cas péjoratif. Il signifie simplement qu'il est nécessaire de bien maîtriser toutes les sections précédentes avant de se lancer dans la programmation objet.

## 24.3 Utilisation de méthodes

Il est bien entendu possible de déclarer et utiliser des méthodes. Pour cela, il faut déclarer une fonction indépendante de la classe, avant la déclaration du constructeur. Ensuite, on déclare la fonction à l'intérieur du constructeur, en l'affectant à une méthode. Il faut faire attention lors de cette déclaration, car **il ne faut pas mettre les parenthèses des fonctions**[16] ! Voici la syntaxe à suivre :

```
function nom_classe () {
this.méthode = fonction ;
}
```

**Exemple 24.2 – Méthode d'un classe**

```
function affich_infos () {
      document.write (this.nom + " a " + this.age + "
      ans.");
      }
function Eleve (Nom,Age) {
      this.age = Age ;
      this.nom = Nom ;
      this.affich = affich_infos;
      }

var Laurent = new Eleve('Laurent',17) ;
Laurent.affich() ;
```

Il est possible de simplifier l'écriture lorsqu'il y a beaucoup de propriétés, grâce à l'utilisation de `with(objet){}`. Cela ne fonctionne qu'à l'intérieur des méthodes, et non dans le constructeur.

**Exemple 24.3 – Méthode d'un classe**

```
function Eleve (Nom,Age) {
age = Age ;
nom = Nom ;
affich = affich_infos;
}
function affich_infos () {
with (this) {
    document.write (nom + " a " + age + " ans.");
    }
}
var Laurent = new Eleve(17, 'Laurent') ;
Laurent.affich() ;
```

## 24.4 Exercice

Dans cet exercice, le but est de demander 3 informations à l'utilisateur (nom, age, ville) à l'aide d'une boite de dialogue. On regroupe ces informations à l'aide d'un objet – d'une classe crée auparavant – puis on utilise une méthode pour l'affichage dans une zone de texte.

[16] Même si cela semble contradictoire, ceci est important. On ne peut faire des égalités de fonctions.

# 25 EXPRESSIONS REGULIERES

## 25.1 Présentation

Ce chapitre s'adresse à des programmeurs avertis, qui voudront bâtir un site dynamique d'un aspect assez évolué. Les expressions régulières sont assez compliquées à comprendre, et leur utilisation demande une connaissance sans failles de l'objet `String`.

## 25.2 Définition

Les expressions régulières existent dans la plupart des langages de programmation, mais sont peu connues du fait de leur complexité. Elles permettent de réaliser des traitements sur les chaînes de caractères. Ces traitements sont de l'ordre de la recherche de caractères, de l'extraction de sous-chaînes… beaucoup d'autres traitements existent qu'il revient au programmeur d'imaginer et de créer. En réalité, une expression régulière possède un motif, qui est une suite de caractères ordonnés selon un ordre, un nombre d'apparitions, une non apparition…

Une expression régulière est avant tout un objet `RegExp`. Comme tout objet, il se déclare ainsi :

```
var reg = RegExp (pattern, option);
```

Pour la première fois, il faut fournir des paramètres au constructeur. Ces paramètres concernent l'expression régulière que vous utilisez. Le pattern est le motif de l'expression. L'option est une chaîne de caractères, qui – comme son nom l'indique – contient les options de l'expression régulière. Nous allons voir leur constitution ci-après.

Il existe aussi une autre façon de déclarer une expression régulière. Elle est moins utilisée et moins élégante.

```
var reg = /pattern/option;
```

Les deux notations sont absolument équivalentes mais la première est plus explicite, donc conseillée.

## 25.3 Paramètres

Les exemples sont regroupés dans une section ultérieure car ils se doivent de contenir les deux paramètres, qui sont traités séparément ci-après.

### 25.3.1 Option

La chaîne option peut prendre 4 valeurs. La première est la chaîne vide `""`, qui signifie l'absence d'option. Les 3 autres valeurs sont détaillées ci-dessous.

- `"g"` : la recherche est globale – sur toute la chaîne -.
- `"i"` : ne pas tenir compte de casse – majuscules/minuscules- .
- `"gi"` : les deux options réunies

### 25.3.2 Pattern

Cette partie est la plus délicate. Le pattern est assez complexe et sa compréhension peut être difficile. Il faut savoir que le pattern correspond aux caractères recherchés – pour une raison ou une autre – dans une chaîne.

La chaîne pattern est extensible à l'infini. Le choix de ce qu'elle contient appartient – du moins en grande partie – au programmeur. Il s'agit du motif de la chaîne, c'est-à-dire des caractères qui inclus ou exclus de la recherche. Cette chaîne pattern contient des caractères spéciaux concaténés. Ces caractères spéciaux concernent le motif lui-même, le caractère à rechercher, le nombre d'occurrences, le groupe de caractères cherché… Leur liste se trouve dans le tableau ci-dessous :

| Motif | Signification |
|---|---|
| ^ | Début du pattern – de la chaîne. |
| $ | Fin du pattern. Interdit tout caractère après. |
| . | N'importe quel caractère. |
| a \| b | Caractère a OU b. |
| * | Caractère précédent présent 0 à x fois. |
| + | Caractère précédent présent 1 à x fois. |
| ? | Caractère précédent présent 0 à 1 fois. |
| {x} | Caractère précédent présent x fois. |
| {x,} | Caractère précédent présent au moins x fois. |
| {x,y} | Caractère précédent présent au entre x et y fois. |
| [abc] | Groupe de caractères : n'importe lequel contenu entre les crochets. |
| [a-z] | N'importe quel caractère alphabétique. |
| [^a-z] | Aucun caractère alphabétique. |
| \\ | Caractère « \ ». |
| \d | Tous les chiffres – équivalent de [0-9] . |
| \D | Aucun chiffre – équivalent de [^0-9]. |
| \b | Limites des mots (espace, tab, …). |
| \s | Tous les caractères d'espacements – équivalent de [\v\r\n\t\f]. |
| \S | Aucun caractère d'espacements – équivalent de [^\v\r\n\t\f]. |
| \w | Tous les caractères alphanumériques dont « _ » – équivalent de [A-Za-z0-9_]. |
| \W | Aucun caractère alphanumérique – équivalent de [^A-Za-z0-9_]. |

**TAB. 13 – Motifs du pattern**

Les différents motifs ne sont pas séparés. On les met les uns à la suite des autres, sans espacements. Il est à préciser que l'on peut très bien mettre une variable ou un mot entre guillemets - sans mise en forme avec ^ $ - comme argument pattern.

### 25.3.3 Exemple

Voici différents exemples de déclaration d'objet `RegExp`. Ces exemples ne sont pas très complexes, ils donnent simplement une idée de la construction des motifs. Le pattern est plus détaillé que les options qui ne sont pas très importantes. Chaque exemple est commenté, pour bien comprendre le but du motif.

**Exemple 25.1 – Notation objet**

```
var exp = new RegExp("^[A-Za-z0-9]{4,}$","i") ;
```

L'exemple ci-dessus présente la recherche d'une chaîne de au moins 4 caractères alphanumériques. `[A-Za-z0-9]` désigne les caractères alphanumériques, et `{4,}` désigne 4 fois ou plus. Les caractères sont au chois majuscules ou minuscules. L'exemple ci-après est équivalent, mais il utilise l'autre notation.

**Exemple 25.2 – Notation simplifiée**

```
var exp = /^[A-Za-z0-9]{4,}$/i ;
```

**Exemple 25.3 – Vérification d'e-mail**

```
var exp = new RegExp ("^[A-Za-z0-9]+[A-Za-z0-9\.\-
_]*@[A-Za-z0-9\-_]+\.[A-Za-z0-9\.\-_]{1,}$","");
```

L'exemple ci-dessus présente la vérification d'une adresse e-mail. Elle doit commencer par au moins un caractère alphanumérique : `[A-Za-z0-9]+` . Ensuite elle peut comporter autant de caractères alphanumériques que l'on veut, plus le point, le tiret et l'underscore : `[A-Za-z0-9\.\-_]*`. Tout cela doit être suivi d'un @ : `@`. Ensuite, il peut y avoir n'importe quel nombre de caractères alphanumériques, plus le point, le tiret et l'underscore : `[A-Za-z0-9\-_]+`. Cela doit être suivi d'un point : `\.` . Ce dernier doit être suivi d'au moins deux caractères alphanumériques dont le point et le tiret : `[A-Za-z0-9\.\-_]{1,}`. Il n'y a aucune option.

Ces exemples ne traitent que de la création de l'objet `RegExp`. Ce dernier est bien entendu inutile si on ne l'utilise pas ensuite. C'est le sujet de la section suivante.

## 25.4 Utilisation

### 25.4.1 Méthodes de l'objet RegExp

Il existe trois méthodes de l'objet `RegExp` : `test()`, `exec()` et `compile()`. Elles s'utilisent selon la syntaxe objet habituelle. La première prend en paramètre la chaîne à tester selon le motif de l'expression régulière. Elle renvoie un booléen qui indique si la chaîne correspond au motif ou non. La deuxième méthode prend aussi la chaîne à tester en paramètre. Elle renvoie un tableau des occurrences du motif à tester. La dernière permet de modifier le motif de l'expression régulière, sans créer un nouvel objet.

**Exemple 25.4 – RegExp.test()**

```
adresse = prompt ("Votre mail ?","mail@fai.com") ;
var exp = new RegExp ("^[A-Za-z0-9]+[A-Za-z0-9\.\-
_]*@[A-Za-z0-9\-_]+\.[A-Za-z0-9\.\-_]{1,}$","");
document.write ( exp.test(adresse) );
```

L'exemple précédent propose de tester la correction d'un mail. On retrouve le motif déjà expliqué dans la section précédente. La méthode `test()` est appliquée à l'adresse e-mail. Le reste est – ou devrait être – familier.

### 25.4.2 Méthodes de l'objet String

Les méthodes de l'objet `String` vont aussi aider à la manipulation des expressions régulières. 3 nouvelles méthodes de l'objet `String`[17] sont introduites ici. A ces trois méthodes, il est possible de rajouter la plupart des méthodes `String`. Ces méthodes nouvelles sont les suivantes :

- `search()` : trouver les occurrences répondant aux critères du motif.
- `replace ()` : trouver remplacer les occurrences répondant aux critères du motif.
- `match()` : trouver les occurrences répondant aux critères du motif.

Ces trois méthodes prennent en paramètre un objet `RegExp`. Voici la syntaxe correspondante :

```
var reg = RegExp (pattern, option);
chaîne.méthode(reg);
```

[17] Elles n'ont pas été introduites précédemment du fait de leur inutilité en dehors des expressions régulières.

Il est à noter que la méthode `split()` possède une syntaxe similaire. Les méthodes `split()` et `match()` renvoient chacune un tableau de toutes les occurrences trouvées.

De plus, il est possible d'indiquer simplement le motif, avec la notation peu utilisée des slash, comme paramètre, comme ceci :

```
chaîne.méthode(/pattern/option);
```

**Exemple 25.5 – String.replace()**

```
var nom = prompt('Votre nom?\nMettez les
accents…','nom') ;
nom = nom.replace (/[éèêë]/g,"e") ;
nom = nom.replace (/[àäâ]/g,"a") ;
nom = nom.replace (/[üûù]/g,"u") ;
nom = nom.replace (/[òöô]/g,"o") ;
nom = nom.replace (/[ìïî]/g,"i") ;
alert(nom) ;
```

L'exemple ci-dessus propose de supprimer les accents d'une chaîne. On remplace tout simplement les lettres accentuées par la lettre sans accent.

## 25.5 Exemple concret

Ci-dessous est proposé un exemple dans lequel vous entrez une suite de mots séparés par des caractères de ponctuation. En cliquant sur un bouton, la liste de tous les noms est affichée. De plus, deux méthodes sont proposées. Cet exemple vous montre aussi comment intégrer le traitement au code avec une fonction.

**Exemple 25.6 – Exemple concret**

```
<HTML>
<HEAD>
<TITLE>New Document</TITLE>
<script language="Javascript">
function TraitBySplit()  {
      var chaine = document.form1.input.value; //récupération de la chaine
      var exp = new RegExp ('[.,;:/!? ]',"g"); //motif avec la ponctuation
      tab = chaine.split(exp); /*séparation de la chaine */
      var result = "Voici les noms :"; //affichage
      for (i = 0 ; i < tab.length ; i++)  {
            result = result + "\n" + tab[i] ;
            }
      document.form1.output.value = result;
}
function TraitByMatch()  {
//récupération de la chaine
      var chaine = document.form1.input.value;
      var exp = new RegExp ('[A-Za-zìïîòöôüûùàäâéèêë]+',"g"); /* motif
avec les lettres */
      tab = chaine.match(exp); // séparation de la chaine
      var result = "Voici les noms :"; //affichage
      for (i = 0 ; i < tab.length ; i++)  {
            result = result + "\n" + tab[i] ;
            }
      document.form1.output.value = result;
}
</script>
</HEAD>
<BODY>
<center>
<form name="form1">
<textarea name="input" rows=5 cols=50>Entrez une suite de noms séparés
indifféremment par les signes de ponctuation .,;:/!? et
espace</textarea><br/>
<input type="button" name="match" value="Avec match()"
onClick="TraitByMatch();"> 
<input type="button" name="split" value="Avec split()"
onClick="TraitBySplit();"><br/>
<textarea name="output" rows=5 cols=50>Résultat</textarea><br/>
</form>
</center>
</BODY>
</HTML>
```

## 25.6 Exercice

Faire un formulaire avec 3 lignes de texte et un bouton. Dans les lignes de texte, l'internaute entre son adresse, code postal et ville. En cliquant sur le bouton, on teste leur validité.

# 26 FONCTIONS ET METHODES

## 26.1 Présentation

Ce chapitre présente – un peu tardivement – les fonctions intégrées au langage, et ne dépendant d'aucun objet. Seront aussi présentées aussi les méthodes et propriétés que l'on peut associer à tous les objets.

## 26.2 Fonctions du langage

### 26.2.1 Escape()

Cette fonction encode les caractères spéciaux d'une chaîne, selon le code `%xx`, et retourne cette chaîne encodée. Voici la syntaxe à utiliser :

```
chaine1 = escape (chaine2)
```

**Exemple 26.1 – Escape()**

```
var chaine = "Voici des caractères spéciaux !" ;
document.write( escape(chaine) ) ;
```

### 26.2.2 Unescape()

Cette fonction décode les caractères spéciaux codé par `escape()`, et retourne cette chaîne encodée.

```
chaine1 = escape (chaine2) ;
```

**Exemple 26.2 – Unescape()**

```
var chaine = "Voici des caractères spéciaux !" ;
var chaine2 = escape(chaine) ;
document.write( unescape(chaine2) ) ;
```

### 26.2.3 ParseFloat()

Cette fonction convertit une chaîne passée en paramètre en nombre décimal. Renvoie `NaN` si la conversion est impossible.

```
decimal = parseFloat (chaine) ;
```

**Exemple 26.3 – ParseFloat()**

```
var chaine = "Voici des caractères spéciaux !" ;
var chaine2 = "35.7" ;
document.write( parseFloat(chaine) + "<br/>" ) ;
document.write( parseFloat(chaine2) ) ;
```

### 26.2.4 ParseInt()

Cette fonction convertit une chaîne passée en paramètre en nombre entier. Renvoie `NaN` si la conversion est impossible. Le paramètre facultatif base permet de faire une conversion en une autre base que décimale. Voici la syntaxe :

```
decimal = parseInt (chaine, base) ;
```

**Exemple 26.4 – ParseInt()**

```
var chaine = "35.7" ;
document.write( parseInt(chaine) + "<br/>" ) ;
document.write( parseInt(chaine, 8) ) ;
```

### 26.2.5 IsFinite()

Cette fonction renvoie `true` si le nombre est fini, sinon, elle renvoie `false`. Voici la syntaxe à suivre :

```
booleen = isFinite (nombre) ;
```

**Exemple 26.5 – IsFinite()**

```
var chaine = "35.7" ;
var chaine2 = "Math";
document.write( isFinite(chaine) + "<br/>") ;
document.write( isFinite(chaine2) ) ;
```

### 26.2.6 IsNaN()

Cette fonction renvoie `true` si la chaîne n'est pas un nombre, sinon, elle renvoie `false`.

```
booleen = isNaN (chaine);
```

**Exemple 26.6 – IsNaN()**

```
var chaine = "35.7" ;
var chaine2 = "Math";
document.write( isNaN(chaine) + "<br/>") ;
document.write( isNaN(chaine2) ) ;
```

## 26.3 Méthodes et propriétés des objets

Les objets de Javascript ou créés par le programmeur possèdent deux propriétés et deux méthodes en commun. Elles permettent de manipuler ces objets et de connaître certaines de leurs caractéristiques.

### 26.3.1 Prototype

Cette propriété permet de rajouter une propriété ou une méthode à un objet que créé ou qui existe dans JS. On l'utilise selon la syntaxe suivante :

```
classe.prototype.nom = valeur ;
```

L'exemple suivant rajoute une propriété et une méthode à l'objet `Array`. Elle permettent de mettre un commentaire au tableau et de renvoyer le premier élément du tableau – ce qui n'est pas très utile en soit.

**Exemple 26.7 – Object.prototype**

```
var tab = new Array(5);
for (i = 0; i < 5; i++) tab[i] = i+1;
//on rajoute un commentaire
Array.prototype.comment = null;
tab.comment = "Tableau de 5 chiffres";
//fonction retournant le premier élément
function FirstElement () {
      return this[0];
      }
Array.prototype.firstElement = FirstElement;
document.write (tab.comment + " : premier élément :
" + tab.firstElement());
```

### 26.3.2 Constructor

Cette propriété renvoie tout simplement le nom du constructeur de l'objet. Voici la syntaxe à suivre :

```
variable = objet.constructor ;
```

**Exemple 26.8 – Object.constructor**

```
var tab = new Array(5);
var s = "string";
var d = new Date ()
var e = new RegExp();
document.write ("Constructeur Array : " +
tab.constructor + "<br/>");
document.write ("Constructeur String : " +
s.constructor + "<br/>");
document.write ("Constructeur Date : " + d.constructor
+ "<br/>");
document.write ("Constructeur RegExp : " +
e.constructor + "<br/>");
```

### 26.3.3 ValueOf()

Cette méthode renvoie tout simplement la valeur de l'objet. Il faut utiliser la syntaxe suivante :

```
variable = objet.valueOf ;
```

**Exemple 26.9 – Object.valueOf()**

```
var tab = new Array(5);
for (i = 0; i < 5; i++) tab[i] = i + 1;
var s = "string";
document.write ("Valeur du Tableau : " + tab.valueOf()
+ "<br/>");
document.write ("Valeur de la Chaîne : " + s.valueOf()
+ "<br/>");
```

### 26.3.4 ToString()

Cette méthode retourne la description de l'objet. Voici la syntaxe à suivre :

```
variable = objet.toString ;
```

**Exemple 26.10 – Object.toString()**

```
var tab = new Array(5);
for (i = 0; i < 5; i++) tab[i] = i + 1;
var s = "string";
document.write ("Description du Tableau : " +
tab.toString() + "<br/>");
document.write ("Description de la Chaîne : " +
s.toString() + "<br/>");
```

## 26.4 Exercice

Dans cet exercice, il faut rajouter 3 méthodes à la classe `String`. La première renverra une chaîne avec la valeur et le constructeur. La seconde renverra la chaîne codée. Et la dernière renverra le dernier caractère de la chaîne.

# 27 LISTE DES EXEMPLES

# 28 LISTE DES TABLEAUX

# 29 INDEX

# 30 LIENS

## 30.1 Programmation web

| | |
|---|---|
| www.moteurprog.com | Site regroupant de nombreux tutoriaux ainsi qu'un annuaire de sites de programmation. |
| www.developpez.com | Site de programmation en général. De nombreux cours et tutoriaux sur les langages web. |
| www.toutenligne.com | Possède plusieurs tutoriaux sur la programmation Javascript, HTML, PHP, CSS… |

## 30.2 Javascript

| | |
|---|---|
| www.toutjavascript.com | Très bon site sur le Javascript, regroupant tutoriaux, scripts et forums. |
| www.commentcamarche.net | Encyclopédie informatique libre. Possède un bon cours Javascript. |
| www.javascriptfr.com | Site comprenant beaucoup de script et un forum. |

# 31 REMERCIEMENTS

L'auteur tient à remercier un certain nombre de personnes et organismes qui l'ont aidé dans sa tâche.

Je remercie tout particulièrement Philippe Jossa et M. Allardin qui m'ont donné cette passion de la programmation. Merci aussi à Arnaud Guillaume et Dimitri Masson pour m'avoir aidé à progresser dans cette voie. Merci à Jérémie Garceries pour ses conseils avisés. Enfin, merci à tout le club informatique du Lycée du Grésivaudan de Meylan (Isère), tout particulièrement la 1°S9 et la TS7 pour leur aide précieuse.

Je remercie tout autant les divers sites que j'ai consultés, qui sont cités dans la section 29 BIBLIOGRAPHIE.

Merci aussi à Packard Bell, constructeur de mon PC, à Aiwa, constructeur de ma chaîne Hi-fi, à Wanadoo, pour ma connexion Internet, à EDF, pour leur électricité, à Microsoft, pour son Windows XP et pour le logiciel Word, à ES-Computing, pour le logiciel Editplus 2, à Adobe, pour le logiciel de création de fichier PDF et enfin à Nullsoft, pour le logiciel Winamp.

Merci à tous les artistes qui m'ont accompagné pendant la rédaction de ce document : Archive, Beach Boys, Cranberries, Hoobastank, Led Zeppelin, Limp Bizkit, Muse, Nickleback, Nirvana, Placebo, Police, Queen, The Rasmus, REM, Scorpions, Sublime, Supertramp, Téléphone, The White Stripes et tant d'autres…

# 32 HISTORIQUE

| | | |
|---|---|---|
| 23/06/2004 | Version 1.0.0 | Première version du cours. |
| 01/07/2004 | Version 1.1.0 | Ajout de la partie Historique, quelques corrections. |
| 10/07/2004 | Version 1.2.0 | Nouveau look du document et nombreux changement grammaticaux. |
| 12/07/2004 | Version 1.3.0 | Ajouts des sections Table des illustration, Tables des Exemples, et Index. |
| 13/07/2004 | Version 1.3.1 | Ajout de la section Liens et nouvelles corrections. |
| 15/04/2006 | Version 2.0.0 | Changements esthétiques. |
| 16/04/2006 | Version 2.0.1 | Mises à jour sommaires, corrections orthographiques. |
| 17/04/2006 | Version 2.1.0 | Améliorations des exemples et exercices. |

La version la plus récente de ce document peut être demandé à l'adresse e-mail suivante : michael.muraz@insa-lyon.fr