LE JAVASCRIPT

Sommaire

INTRODUCTION

« JavaScript peut raisonnablement se targuer d'être le langage de programmation le plus incompris au monde.

Bien que souvent raillé comme étant un simple jouet, derrière sa simplicité désarmante se cachent certaines fonctionnalités de langage très puissantes. On a vu au cours de l'année 2006 l'apparition d'un certain nombre d'applications JavaScript de tout premier plan, ce qui montre qu'une connaissance approfondie de cette technologie est une compétence importante pour tout développeur Web.

Il peut être utile de commencer avec une idée de l'histoire de ce langage. JavaScript a été créé en 1995 par Brendan Eich, un ingénieur chez Netscape, et fourni pour la première fois avec Netscape 2 début 1996. Il était au départ censé s'appeler LiveScript, mais a été renommé par une décision marketing néfaste dans le but de capitaliser sur la popularité du langage Java de Sun Microsystems, malgré qu'ils n'aient que très peu en commun. Cela n'a jamais cessé d'être une source de confusion.

Microsoft a lancé quelques mois plus tard avec Internet Explorer 3 une version globalement compatible du langage, appelée JScript. Netscape a soumis le langage à l'Ecma International, une organisation de normalisation européenne, ce qui a abouti à la première édition du standard ECMAScript en 1997. Ce standard a reçu une mise à jour importante appelée ECMAScript edition 3 en 1999, et n'a plus bougé depuis, bien qu'une quatrième édition soit en cours de rédaction.

Cette stabilité est une excellente nouvelle pour les développeurs, parce qu'elle a donné aux différentes implémentations tout le temps nécessaire pour s'y adapter.
[…]

Contrairement à la plupart des langages de programmation, JavaScript n'a pas de concept d'entrée ou de sortie. Il est conçu pour s'exécuter comme un langage de script dans un environnement hôte, et c'est à cet environnement de fournir des mécanismes de communication avec le monde extérieur. L'environnement hôte le plus commun est un navigateur, mais d'autres interpréteurs JavaScript existent également. »

https://developer.mozilla.org/fr/Une_r%c3%a9introduction_%c3%a0_JavaScript

1.GENERALITES

1.1.Le langage Javascript

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 faudra donc de bonnes bases en HTML.

Son principal inconvénient est le fait qu'il ne dispose pas de débogueur, la détection des erreurs en devient fastidieuse. Ensuite, il le code est accessible, puisque contenu dans la page envoyée au navigateur.

1.2.Côté pratique

Pour programmer en Javascript, il suffit d'un navigateur web et d'un éditeur de texte, le bloc-notes de Windows est suffisant.
Il est conseillé d'utiliser un éditeur qui propose une coloration syntaxique (par exemple l'excellent Notepad++ qui est gratuit).

2.LE LANGAGE

2.1.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 trois façons d'intégrer du code. La première consiste à le placer entre les balises, tout simplement.

Syntaxe :

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

Exemple 2.1 :

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

Il est 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>`.

Syntaxe :

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

Exemple 2.2 :

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

Plutôt déconseillé, il est aussi possible d'embarquer du code javascript avec la balise HTML :

Exemple 2.3 :

```
<button onclick="alert('Ok')">Envoi</button>
```

2.2.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 le même ligne, puisque l'interpréteur ignore les sauts de lignes.

Exemple 2.3 :

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

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

Exemple 2.4 :

```
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 « */ »

Exemple 2.5 :

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

3.LES VARIABLES

3.1.Les constantes

JS propose quatre types de constantes :

- Les constantes **numériques**, en notation décimale (75,2) ou flottante, c'est-à-dire scientifique (752E-1).
- Les constantes **booléennes** : **true** et **false**.
- Les **chaînes de caractères**, encadrées de guillemets ou de d'apostrophes (‘’).
- La constante **null** qui signifie « rien », « sans valeur », et qui est attribuée au variables non définies.

3.2.Les variables en JS

3.2.1.Les types de variable

Un nom de variable doit commencer par une lettre (alphabet ASCII) ou le signe « _ » et se composer de lettres, de chiffres et des caractères « _ » et « $ » (à l'exclusion du blanc). Le nombre de caractères n'est pas précisé. 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 :

- Les nombres : `number`
- Les chaînes de caractères : `string`
- Les booléens[1] : `boolean`
- Les objets : `object`
- Les fonctions : `function`

Les fonctions seront vues au chapitre 5.LES FONCTIONS, voici déjà un premier exemple : la fonction **`typeof()`**, qui retourne le type d'une variable.

Exemple 3.1 :

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

Si la variable n'est pas affectée d'une valeur, elle recevra alors le type **`undefined`**, comme le montre l'exemple 3.2.

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

Exemple 3.2 :

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

3.2.2.La déclaration et l'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 3.3 :

```
var Numero;
```

Dans les exemples qui suivent, par facilité, l'affectation et la déclaration sont regroupées. Les variables peuvent être déclarées de deux façons :

- De façon explicite, avec le mot-clé « **var** ».

Exemple 3.4 :

```
var Numero = 1 ;
```

- De façon implicite. On écrit directement le nom de la variable suivi de la valeur que l'on lui attribue.

Exemple 3.5 :

```
Numero = 1 ;
```

3.2.3.Les noms 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.

Lettre	Mot-clé
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. 3.1 : Noms réservés

3.2.4.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.

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

Certains caractères 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 des balises HTML dans les chaînes de caractères. Ces dernières seront interprétées comme de vraies balises.

Cela sera vu plus en détail dans le chapitre 16.L'OBJET STRING.

3.2.5.Variables globales et variables locales

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

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

4. LES OPERATEURS

Comme tout langage informatique, JS possède des opérateurs pour effectuer les calculs. Dans les exemples suivants , la valeur initiale de x sera toujours égale à 11, et celle de y sera égale à 5.

4.1.Les opérateurs de calcul

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 4.1 : opérateurs de calcul

4.2.Les opérateurs de comparaison

Signe	Nom	Exemple	Résultat
==	Egal	x == 11	true
<	Inférieur	x < 11	false
<=	Inférieur ou égal	x <= 11	true
>	Supérieur	x > 11	false
>=	Supérieur ou égal	x >= 11	true
!=	Différent	x != 11	false

Tab 4.2 : opérateurs de comparaison

Ces opérateur seront utilisés dans les boucles conditionnelles[2]. Le résultat s'exprime alors en valeur booléenne.

[2] Voir pour cela le chapitre 6.STRUCTURES DE CONTROLE

4.3.Les 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 4.3 : opérateurs associatifs

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

4.4.Les opérateurs logiques

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

Tab 4.4 : 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[3].

4.5.Les 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 4.5 : opérateurs d'incrémentation

[3] Il s'agit d'un OU non exclusif, pour les puristes.

5. LES FONCTIONS

5.1.Définition

C'est un groupe d'instruction prédéfini et exécuté lorsqu'il est appelé. Ce groupe d'instructions a un nom et peut être appelé plusieurs fois.

En Javascript, il existe deux types de fonctions :

- les fonctions propres à Javascript
- les fonctions définient par le programmeur.

5.2.Déclaration

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

Syntaxe :

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

Le nom d'une fonction suit les mêmes règles que celui d'une variable[4]. Chaque nom de fonction doit être unique dans un même script. Les parenthèses sont obligatoires même si il n'y a pas d'arguments.

5.3.Les 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>

5.4.L'appel d'une fonction

Syntaxe :

```
nom_de_la_fonction();
```

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

5.5.Fonctions manipulant des valeurs

5.5.1.Passer une valeur à une fonction

On peut 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.

Exemple 5.1 : une fonction affichant le texte dans la page à laquelle on fournit le texte à afficher.

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

On peut passer plusieurs paramètres à une fonction, en séparant les paramètres par des virgules.

Syntaxe :

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

Exemple 5.2 :

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

5.5.2.Retourner une 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.

Exemple 5.3 :

```
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
```

NB : Le mot-clé **return** est facultatif.

5.6.Variables locales et variables globales

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.

Exemple 5.4 :

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

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

Exemple 5.5 :

```
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 5.6 :

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

6. LES STRUCTURES DE CONTROLE

6.1.Les structures algorithmiques

Quelque soit le langage informatique utilisé[5], un programme informatique utilise 3 catégories principales d'instructions. Ce sont :

- les instructions séquentielles.
- les instructions alternatives, ou conditionnelles.
- les instructions itératives, ou répétitives.

Nous verrons chacun des types de structures, en comparant l'algorithme et le code JS.

6.2.La 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
Instruction1
Instruction2
FIN
	{
	Instruction1
	Instruction2
	}

6.3.Les instructions conditionnelles

6.3.1.L'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.

Algorithme	Code JS

SI condition
ALORS Instructions1
SINON Instructions2
FINSI
	If (condition) {
	Instruction1
	}
	else {
	Instruction2
	}

Tab. 6.1 : Expression conditionnelle if…else

Exemple 6.1 :

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

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

Dans l'exemple 6.1, la fonction prompt() permet d'afficher une boîte de dialogue.

6.3.2.L'expression () ? :

Il s'agit d'un moyen rapide de tester une condition et de d'exécuter une instruction selon son résultat.

Algorithme	Code JS
SI condition ALORS Instructions1 SINON Instructions2 FINSI	(condition) ? instruction 1 : instruction 2

Tab. 6.2 : Expression conditionnelle () ? :

Dans cette expression, si la condition est vérifiée, l'instruction 1 est effectuée, sinon, l'instruction 2 est effectuée. Précisons, que toute l'expression doit se trouver sur la même ligne.

Exemple 6.2 :

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

6.4.Les instructions itératives

6.4.1.L'itération `for`

Elle permet de répéter des instructions en s'arrêtant à un certain nombre d'itérations (boucles).

Algorithme	Code JS

POUR i = valeur initiale A i = valeur finale
REPETER instructions
FIN POUR
	For (val initiale ; condition ; incrémentation) {
	Instructions
	}

Tab. 6.3 : Expression itérative for

Exemple 6.3 :

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

6.4.2.L'itération `while`

Elle permet de répéter des instructions tant qu'une condition est vraie.

Algorithme	Code JS

TANT QUE (condition vraie)
REPETER instructions
FIN TANT QUE
	while (condition) {
	instructions
	}

Tab. 6.4 : Expression itérative while

Exemple 6.4 :

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

6.5.Interrompre une boucle

6.5.1.L'instruction `break`

Elle permet de mettre fin prématurément à une instruction itérative `while` ou `for`.

Exemple 6.5 :

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

6.5.2.L'instruction `continue`

Elle permet de mettre fin à une itération et de passer à la boucle suivante.

Exemple 6.6 :

```
for (i = 0; i < 5; i++) {
   if (i == 2) {
   continue ;
   }
   document.write(i + '<br>');
}
```

7. LES BOITES DE MESSAGE

7.1.Généralités

Les boîtes de dialogue, qui sont des méthodes de l'objet window, abordé plus loin dans la partie objet du langage.

7.2.Alert()

Ne comporte qu'un texte informatif et un bouton « OK ».

Syntaxe :

```
alert (paramètres) ;
```

Paramètres :

- Une chaîne de caractère
- Une variable
- Un enchaînement des deux, séparés par le signe "+"

Exemple 7.1 :

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

7.3.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 ».

Syntaxe :

```
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** ».

Paramètres :

- Le texte à afficher dans la boite de message
- La valeur par défaut à afficher dans la boite d'invite.

Exemple 7.2 :

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

7.4.Confirm()

Il s'agit d'une boite de confirmation, composée d'un texte, d'un bouton « OK » et d'un bouton « Annuler ».

Syntaxe :

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

La méthode retourne **`true`** si on clique sur « OK », et **`false`** si on clique sur « Annuler »

Paramètre :

- Le texte à afficher dans la boite de message

Exemple 7.3 :

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

8. NOTION OBJET

8.1.Le concept objet

Jusque-là, nous avons vu des variables, avec une valeur, indépendantes 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, on peut voir les propriétés et utiliser les méthodes.

8.2.Création d'un objet

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

Syntaxe :

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

Exemple 8.1 :

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

8.3.Accès aux propriétés et aux méthodes

On accède aux propriétés – et aux méthodes – d'un objet en accolant l'objet et sa propriété avec l'opérateur « . ».

Syntaxe :

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

Exemple 8.2: par extrapolation

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

Exemple 8.3:

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

L'objet `document` est intégré au langage, et sa méthode `write()` permet d'écrire dans la page courante.

8.4.L'objet `this`

Il existe un objet très utile en JS objet : `this`. Même s'il sera plus utile en programmation objet, lorsque vous créerez vos classes. Il s'agit d'un objet qui représente l'objet en cours. Il possède alors les propriétés et les méthodes de l'objet.

9. LES FORMULAIRES

9.1.Généralités

9.1.1.Présentation

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.

9.1.2.La balise `form`

Chaque formulaire doit être encadré par les balises `<form>…</form>`. Toutes les balises seront comprises entre ces deux-là.

Si on utilise un langage coté serveur (PHP ou autre) , on a besoin 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.

Exemple 9.1 :

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

Pour que les informations du formulaire soient envoyées par mail, préciser '`mailto:…`' dans l'action.

Exemple 9.2 :

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

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

9.2.Les objet du formulaire

Même si nous n'avons pas encore vu les objets du navigateur[6], nous allons introduire ici l'objet `document`. 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 :

Syntaxe :

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

L'exemple 9.15 montre comment attribuer une valeur à une ligne de texte.

Exemple 9.21 :

```
<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 cet exemple, la fonction est appelée à la fin du chargement grâce au gestionnaire d'événement[7] placé dans la balise `<body>`.

[7] Abordé dans le chapitre 10.LES EVENEMENTS.

10. LES EVENEMENTS

10.1.Généralités

10.1.1.Présentation

Le JS permet de réagir à certaines actions de l'utilisateur. On nomme évènement le mot Event, et gestionnaire d'événement le mot-clé `on*Event*`.

10.1.2.Fonctionnement

Les évènements sont souvent indiqués dans la balise d'un formulaire. (*Cependant les puristes considèrent que ce code est « intrusif » et préfèrent « attacher » les évènements dynamiquement au chargement de la page.*)

Syntaxe :

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

- balise : désigne le nom de la balise HTML qui supporte l'événement.
- onEvent : désigne le gestionnaire d'événement associé à l'événement *Event*.
- 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.

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.

Exemple 10.1 :

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

10.2.Le clic de souris

Gestionnaire d'événement :

onClick

Exemple 10.2 :

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

Balises supportées :
<input type="button">, <input type="checkbox">, <input type="radio">, <input type="reset">, <input type="submit">, <a href..>

10.3.Le chargement

10.3.1.Load

Gestionnaire d'événement :

onLoad

Exemple 10.3 :

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

Balises supportées :
<body>, <frameset>

10.3.2.UnLoad

Gestionnaire d'événement :

onUnLoad

Exemple 10.4 :

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

Balises supportées :
<body>, <frameset>

10.3.3.Error

Lorsque le chargement d'une page ou d'une image s'interrompt en erreur.

Gestionnaire d'événement :

onError

Exemple 10.5 :

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

Balises supportées :
<body>, <frameset>, <img>

10.4.Le passage de la souris

10.4.1.MouseOver

Gestionnaire d'événement :

onMouseOver

Exemple 10.7 :

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

Balises supportées :
<a href…>, <area href…>

10.4.2.MouseOut

Gestionnaire d'événement :

onMouseOut

Exemple 10.8 :

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

Balises supportées :
<a href…>, <area href…>

10.5.Le focus

10.5.1.Focus

Lorsqu un contrôle texte ou une zone d'input accueille le curseur.

Gestionnaire d'événement :

onFocus

Exemple 10.9 :

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

Balises supportées :
<input type="text">, <select>, <textarea>, <input type="password">

10.5.2.Blur

Lorsqu'un contrôle perd le focus.

Gestionnaire d'événement :

onBlur

Exemple 10.10 :

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

Balises supportées :
<input type="text">, <select>, <textarea>, <input type="password">

10.6.Les changements

Lorsque la valeur d'un texte ou d'une option change dans un formulaire

Gestionnaire d'événement :

onChange

Exemple 10.11 :

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

Balises supportées :
<input type="text">, <select>, <textarea>, <input type="password">

10.7.La sélection

Sélection dans un champ de texte.

Gestionnaire d'événement :

onSelect

Exemple 10.12 :

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

Balises supportées :
<input type="text">, <textarea>

10.8.L'envoi

Click sur un bouton « submit » d'un formulaire de type « post » ou « get ».

Gestionnaire d'événement :

```
onSubmit
```

Exemple 10.13 :

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

Balises supportées :
<input type="submit">

11. L'OBJET ARRAY

11.1.Généralités

11.2.Création et affectation d'un tableau

11.2.1 Création d'un tableau

On créé un tableau comme n'importe quel objet de classe. On invoque le constructeur de classe. On indique 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.

Syntaxe :

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

Exemple 11.1 :

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

11.2.2 Affectation d'un tableau

Le numéro commence à 0 et finit au nombre maximum moins 1.

Syntaxe :

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

Exemple 11.2 :

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

L'exemple suivant entre une série de nombre en ajoutant 1 à chaque fois.

Exemple 11.3 :

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

11.3.Accès aux données d'un tableau

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 la rentrer comme paramètre d'une fonction.

Syntaxe :

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

Exemple 11.4 :

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

11.4.Tableau à 2 dimensions

Pour créer un tableau à 2 dimensions, on utilise l'astuce suivante : on déclare chaque élément comme un nouveau tableau.

Syntaxe :

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

Exemple 11.5 :

```
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 11.6 :

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

Avec ce système, on peut créer un tableau à 3,4 dimensions ou plus, bien que l'utilité en soit quelque peu douteuse…

11.5.Propriétés et méthodes

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

11.5.1.Propriété

L'objet Array ne possède qu'une propriété - `length` - qui désigne le nombre d'éléments du tableau.

Syntaxe :

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

Exemple 11.7 :

```
var MonTableau = new Array (2) ;
```

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

11.5.2.Méthodes

L'objet Array possède plusieurs méthodes qui permettent de manier les éléments du tableau.

Syntaxe :

```
tableau.méthode() ;
```

- `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 11.8 :

```
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(' ')) ;
```

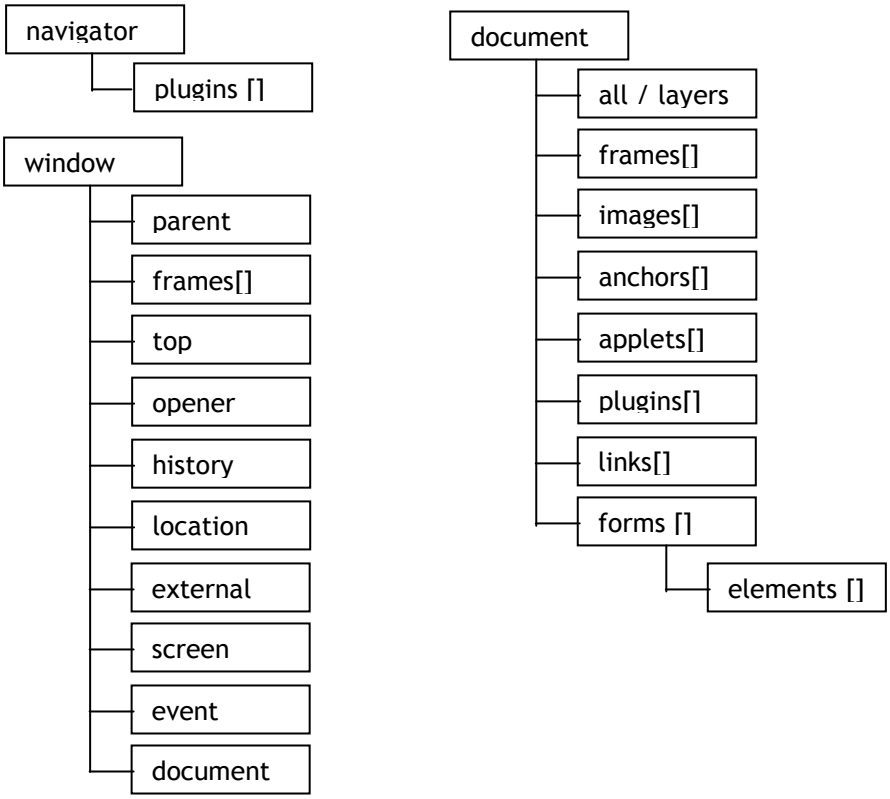
12.LES OBJETS DU NAVIGATEUR

12.1.Généralités

Nous avons vu les classes prédéfinies en JS. Il existe aussi plusieurs objets en JS rattachés à la fenêtre, à la page et au navigateur. Il sont appelés **`window`**, **`document`** et **`navigator`**. Ce sont 3 classes distinctes.

12.2.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.



13.L'OBJET NAVIGATOR

L'objet `navigator` possède toutes les caractéristiques du navigateur ainsi que certaines de l'ordinateur. 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.

13.1.Les propriétés de `navigator`

Toutes ces propriétés ne font que donner des informations sur votre type de navigateur. Pour y accéder, on suit la syntaxe habituelle d'un objet.

Syntaxe :

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

- `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.
- `userAgen`t : informations de appCodeName et de appVersion réunies.

NB : Certaines propriétés ne fonctionnet pas avec tous les navigateurs

- `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.
- `Language` : code langue du browser. (Netscape)

14.L'OBJET WINDOW

Fenêtre du browser en cours d'exécution.

14.1.Les propriétés de `window`

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.

Syntaxe :

```
window.propriété;
```

- `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.

14.2.Les méthodes de `window`

14.2.1.Généralités

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.

14.2.2.Liste des méthodes

- `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.

14.3.Les évènements de `window`

On peut rattacher certains évènements à l'objet window. Il seront placés dans la balise `<body>`, grâce au gestionnaire d'événement `onEvent`.

Syntaxe :

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

- `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.

Pour plus de précisions, consulter le chapitre 7, relatif aux évènements de Javascript.

14.4.Les objets de `window`

L'objet `window` contient plusieurs autres objets assez réduits que je présente dans cette partie.

Syntaxe :

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

14.4.1.L'objet `frames`

Tableau contenant toutes les frames déclarées dans la page. Il ne possède ni propriétés ni méthodes.

Syntaxe :

```
variable = window.frames[i] ;
```

14.4.2.L'objet `parent`

Ppage principale, qui contient la déclaration de toutes les frames. Il possède les mêmes attributs que l'objet `window`.

Syntaxe :

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

14.4.3.L'objet `top`

Ppage de plus haut niveau. Il possède les mêmes attributs que l'objet `window`.

Syntaxe :

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

14.4.4.L'objet `opener`

Page responsable de l'ouverture de la page courante. Possède les mêmes attributs que l'objet `window`.

Syntaxe :

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

14.4.5.L'objet `history`

Historique des pages consultées. Il possède une propriété et 3 méthodes. Voici la propriété et les méthodes de cet objet, qui se déclarent selon la syntaxe objet.

Syntaxe :

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

- `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.

14.4.6.L'objet `location`

Informations contenues dans l'URL de la page en cours.
L'objet `window.location` renvoie l'URL complète de la page en cours.

Syntaxe :

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

Les propriétés suivantes renvoient des informations plus précises concernant l'URL.

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

Il existe deux propriétés de l'objet window.location, qui concernent toutes deux le rechargement de la page.

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

14.4.7.L'objet `screen`

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

Syntaxe :

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

- `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.

14.4.8.L'objet `event`

Il s'agit d'un objet propre à Microsoft Internet Explorer. Il renvoie le type d'événement qui a eu lieu.

Syntaxe :

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

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

15.L'OBJET DOCUMENT

Il s'agit 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."`

15.1.Les propriétés de `document`

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

Syntaxe :

```
document.propriété;
```

- `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.

Exemple 15.1 :

```
<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>
```

15.2.Les méthodes de `document`

Syntaxe :

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

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

- 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.

Exemple 15.2.1 :

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

Exemple 15.2.2 :

```
<html>
<style>
#div1{
background-color:#00ff00;
width:300px;height:100px;
}
</style>
</head>
<body>
<input type="button" value="Changer la couleur de fond en vert"
onclick="document.getElementById('div1').style.backgroundColor='#00ff00';"
/>
<br />
<input type="button" value="Changer la couleur de fond en rouge"
onclick="document.getElementById('div1').style.backgroundColor='#ff0000';"
/>
<br />
<input type="button" value="Augmenter la largeur"
onclick="document.getElementById('div1').style.width=
(parseInt(document.getElementById('div1').clientWidth)+25)+'px';" />
<br />
<input type="button" value="Diminuer la largeur"
onclick="document.getElementById('div1').style.width=
(parseInt(document.getElementById('div1').clientWidth)-25)+'px';" />
<br />
<input type="button" value="Augmenter la hauteur"
onclick="document.getElementById('div1').style.height=
(parseInt(document.getElementById('div1').clientHeight)+25)+'px';" />
<br />
```

```
<input type="button" value="Diminuer la hauteur"
onclick="document.getElementById('div1').style.height=
(parseInt(document.getElementById('div1').clientHeight)-25)+'px';" />
<br />
<input type="button" value="Augmenter la taille de la police"
onclick="document.getElementById('div1').style.fontSize=
(parseInt(document.getElementById('div1').style.fontSize)+3)+'px';" />
<br />
<input type="button" value="Diminuer la taille de la police"
onclick="document.getElementById('div1').style.fontSize=
(parseInt(document.getElementById('div1').style.fontSize)-3)+'px';" />
<br />
<div id="div1" style="font-size:12px;">
Ceci est un  test
</div>
</body>
</html>
```

15.3.Les évènements de `document`

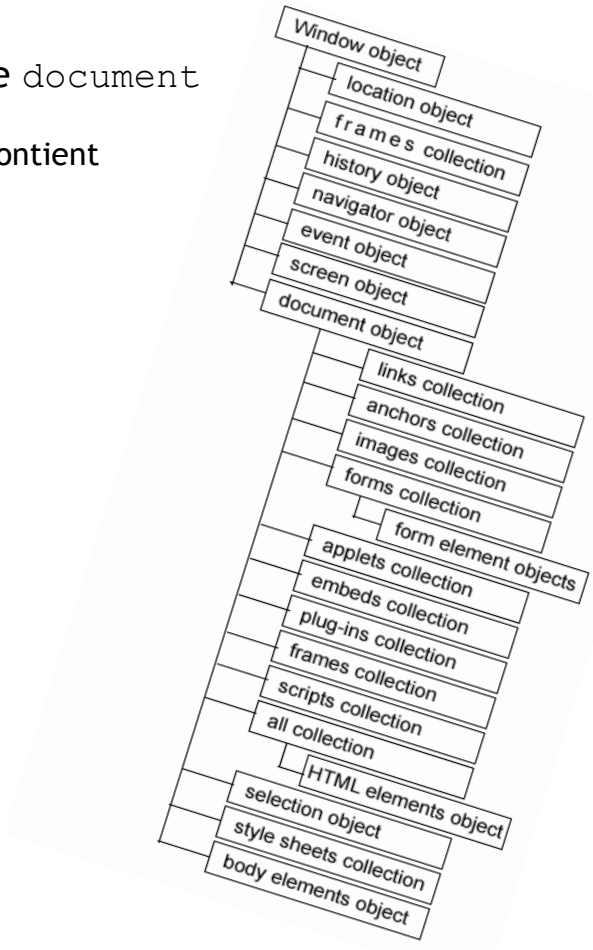
On peut rattacher certains évènements à l'objet `document`.

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

Pour plus de précisions, consulter le chapitre 7, relatif aux évènements de Javascript.

15.4.Les objets de `document`

L'objet `document` contient plusieurs autres objets.



Syntaxe :

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

15.4.1.L'objet `all`

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>`.

Syntaxe :

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

15.4.2.L'objet `layers`

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

Syntaxe :

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

15.4.3.L'objet `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 de formulaire.

Syntaxe :

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

15.4.4.L'objet `anchors`

Tableau contenant toutes les ancres – les balises `<a>` - de la page courante. Il ne possède ni propriétés ni méthodes.

Syntaxe :

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

15.4.5.L'objet `images`

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.

Syntaxe :

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

15.4.6.L'objet `applets`

Ttableau contenant toutes les applets java déclarés dans la page courante. Il ne possède ni propriétés ni méthodes.

Syntaxe :

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

15.4.7.L'objet `plugins`

Tableau `plugins`, qui n'est reconnu que par Netscape. Il s'agit de la liste de tous les plugins installés.

15.4.8.L'objet `frames`

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

Syntaxe :

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

16.LES OBJET DU NOYAU JAVASCRIPT

16.1.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++.

16.2.Quelques précisions

Certains de ces objets seront déjà définis, comme l'objet `Math`. 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, mais qu'il faut appeler selon la syntaxe habituelle. Certains objets `Image` existent déjà dans votre page, sans que vous le sachiez. Il s'agit des balises `<img>`. Ces dernières n'ont donc pas besoin de faire appel au constructeur.

17.L'OBJET MATH

17.1.Fonctions

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

Syntaxe :

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

17.1.1.Fonctions diverses

- 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.

17.1.2.Fonctions trigonométriques

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

17.1.3.Fonctions logarithmiques

- exp(x)
- log(x)

17.2.Constantes

- math.PI
- math.LN2
- math.LN10
- math.E
- math.LOG2E
- math.LOG10E

17.3.Simplification

Si vous utilisez beaucoup l'objet math, 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).**

Syntaxe :

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

Exemple 17.1 :

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

18.L'OBJET STRING

18.1.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`.

18.2.La propriété

Il s'agit de la longueur de la chaîne, "**`length`**".

Syntaxe :

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

Exemple 18.1 :

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

18.3.Les méthodes

18.3.1.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.

Syntaxe :

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

Exemple 18.2 :

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

18.3.2.FromCharCode ()

Cette méthode renvoie les nombre ASCII passés en paramètres sous forme de chaîne de caractère.

Syntaxe :

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

Exemple 18.3 :

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

18.3.3.CharCodeAt ()

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

Syntaxe :

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

Exemple 18.4 :

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

18.3.4.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.

Syntaxe :

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

Exemple 18.5 :

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

18.3.5.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.

Syntaxe :

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

Exemple 18.6 :

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

18.3.6.Substring ()

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

Syntaxe :

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

Exemple 18.7 :

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

18.3.7.Substr ()

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

Syntaxe :

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

Exemple 18.8 :

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

18.3.8.Slice ()

Equivalent à substring(). La fin est facultative.

Syntaxe :

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

Exemple 18.9 :

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

18.3.9.Split ()

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

Syntaxe :

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

Exemple 18.10 :

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

18.3.10.Concat ()

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

Syntaxe :

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

Exemple 18.11 :

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

18.3.11.ToLowerCase ()

Cette méthode renvoie la chaîne avec tous les caractères en minuscules

Syntaxe :

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

Exemple 18.12 :

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

18.3.12.ToUpperCase ()

Cette méthode renvoie la chaîne avec tous les caractères en majuscules

Syntaxe :

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

Exemple 18.13 :

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

18.3.13.FontColor ()

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

Syntaxe :

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

Exemple 18.14 :

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

18.3.14.FontSize ()

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

Syntaxe :

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

Exemple 18.15 :

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

18.3.15.Fixed ()

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

Syntaxe :

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

Exemple 18.16 :

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

18.3.16.Bold ()

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

Syntaxe :

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

Exemple 18.17 :

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

18.3.17.Strike ()

Cette méthode renvoie le texte de l'objet barré.

Syntaxe :

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

Exemple 18.18 :

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

18.3.18.Sub ()

Cette méthode renvoie le texte de l'objet en y ajoutant les balises _….

Syntaxe :

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

Exemple 18.19 :

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

18.3.19.Big ()

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

Syntaxe :

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

Exemple 18.20 :

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

18.3.20.Sup ()

Cette méthode renvoie le texte de l'objet en y ajoutant les balises […].

Syntaxe :

```
variable = chaine.sup()
```

Exemple 18.21 :

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

18.3.21.Blink ()

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

Syntaxe :

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

Exemple 18.22 :

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

18.2.22.Small ()

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

Syntaxe :

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

Exemple 18.23 :

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

18.3.23.Italics ()

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

Syntaxe :

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

Exemple 18.24 :

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

18.3.24.Link ()

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

Syntaxe :

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

Exemple 18.25 :

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

18.3.25.Anchor ()

Cette méthode crée un ancre,et renvoie le texte de l'objet en y ajoutant les balises ….

Syntaxe :

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

Exemple 18.26 :

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

18.4.Manipulations sur les chaînes

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

18.4.1.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 variables comme toute variable.

Exemple 18.27 :

```
x = "Maman" ;
```

```
y = 'Maman' ;
```

Dans l'exemple 12.8, les chaînes x et y sont équivalentes. Elles contiennent la même donnée.

18.4.2.Concaténation

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

Exemple 18.28 :

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

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

Exemple 18.29 :

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

18.4.3.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. 12.1 : 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 18.30 :

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

19.L'OBJET DATE

19.1.Généralités

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

Syntaxe :

```
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. Je ne mets pas d'exemple à chaque méthode, je ferai un exemple général et concret à la fin du chapitre.

19.2.Les méthodes

19.2.1.Get

Une fois la variable Date créée, il faut lui donner les informations sur la date et l'heure actuelle. Les fonctions suivantes remplissent la variable – qui est une chaîne – Date avec les données courantes. Elles utilisent toutes la même syntaxe, ce sont des méthodes objet.

Syntaxe :

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

- `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().

19.2.2.Set

Il est aussi possible de remplir la variable Date avec nos propres données. Les fonctions suivantes remplissent la variable – qui est une chaîne – Date avec les données que vous voulez. Elles utilisent toujours la même syntaxe, ce sont des méthodes objet.

Syntaxe :

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

- `setYear()` : Assigne les 2 derniers chiffres de l'année, sous forme d'un entier supérieur à 1900.
- `setYear()` : 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().

19.2.3.L'heure

L'heure est très utile en JS, elle possède donc plusieurs méthodes utiles. La syntaxe est toujours la même.

Syntaxe :

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

- `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().

19.3.Exemple concret

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 19.1 :

```
<HTML>
<HEAD>
<script language="Javascript">
function GetTime () { //la fonction que l'on doit appeler
      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
      if (hours < 10) hours = "0" + hours; //on rajoute un 0
      if (min < 10) min = "0" + min;       //si le chiffre est
      if (sec < 10) sec = "0" + sec;       //inférieur à 10
      // 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.L'OBJET IMAGE

20.1.Rappel HTML

- 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.

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.

20.2.L'objet

La classe Image correspond à la balise HTML `<img>`.
Les images sont stockées dans le tableau `images[]` de l'objet `document`.

Grâce à un objet Image, on peut précharger 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 20.3.Les propriétés.

20.3.Les propriétés

20.3.1.Syntaxe

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

Syntaxe :

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

20.3.2.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.

Exemple 20.1 :

```
<img src="0.gif" name="image1"
```

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

20.3.3.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 20.2 :

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

20.3.4.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 20.3 :

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

20.3.5.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 20.4 :

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

20.3.6.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 20.5 :

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

20.3.7.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 20.6 :

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

20.3.8.Alt

Elle contient le texte qui affiché avant la 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 20.7 :

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

20.3.9.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 20.8 :

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

20.4.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 20.9 :

```
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 20.10 :

```
<html>
<head>
<script language="Javascript">
var tab = new Array (4) ; //tableau
for (i = 0; i < 4; i++) {//remplissage d'images
      tab[i] = new Image();
      tab[i].src = i + ".gif";
      }
function f() { //fonction d'affichage
      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 14.3 permet de charger 5 images et des les afficher dans 5 balises `<img>` différentes grâce au tableau `images[]`.

20.5.Exemple concret : cliquer pour changer d'image

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, on utilise un compteur.

Exemple 20.11 :

```
<HTML>
<HEAD>
<TITLE>New Document</TITLE>
<script language="Javascript">
//création d'un tableau de 4 éléments
var tab_images = new Array(4);
for (i = 0; i < tab_images.length; i++)  { // pour chaque élément
     tab_images[i] = new Image(); /*on crée un objet Image
     et on lui donne un fichier à charger */
     tab_images[i].src = i + ".gif";
          }
var nb = 0; //on initialise un compteur

function changer()  { // la fonction qui sera appelée.
     nb++; //incrémenter le compteur

     if (nb == tab_images.length)  { //si on est 4
          nb = 0; //on remet à 0 le compteur
          }
     //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();"> <!-- le bouton avec l'évènement -->
</BODY>
</HTML>
```

21.LA PROGRAMMATION MULTI-CADRES[8]

21.1.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 cour HTML.

21.2.Généralités

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 21.1 :

```
<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.

21.3.Liens 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.

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` ».

[8] Par programmation multi-cadres, j'entend la programmation avec frames.

Exemple 21.2 :

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

21.4.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` - que nous avons vu précédemment - nous fournit de quoi le faire. Il contient l'objet `parent`, qui possède les mêmes propriétés que lui.

21.4.1.L'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.

Syntaxe :

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

Exemple 21.3 :

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

Exemple 21.4 :

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

Dans l'exemple 20.4, on accède à la valeur du bouton appelé « Button1 » du formulaire nommé « form1 ».

21.4.2.Le nom de la 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.

Syntaxe :

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

Exemple 21.5 :

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

Exemple 21.6 :

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

21.5.Exemple concret

L'exemple 21.7 permet de faire un compteur de secondes impair sur une frame et pair sur l'autre.

Exemple 21.7 :

Exemple 21.7.1.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 21.7.2.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>
```

Exemple 21.7.htm

```
<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="exemple 21.7.1.htm" name="frame1">
      <frame src="exemple 21.7.2.htm" name="frame2">
</frameset>
</HTML>
```

22.LES COOKIES

22.1.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[9].

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.

22.2.Créer un 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 que l'on veut.

Syntaxe :

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

Dans les informations, il y a tout d'abord la chaîne que l'on souhaite conserver. Ensuite, on met 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.

Syntaxe :

```
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.

[9] 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.

- Le champ information : il s'agit de ce que vous voulez stocker 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-virgule, 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 22.1 :

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

Dans l'exemple 9.1, 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.

22.3.Récupérer un cookie

La récupération d'un cookie est un peu plus complexe. Le but est de trouver le type d'information que l'on cherche, et ensuite de lire la valeur. La maîtrise de la classe 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.

Syntaxe :

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

Ensuite, on cherche le nom de la « variable » dans la chaîne du cookie. Ensuite, 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, un exemple de récupération d' un cookie.

Exemple 22.2 :

```
//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) ;
               }
          }

     }
```

22.4.Modifier un 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.

22.5.Supprimer un cookie

Pour supprimer un cookie, il faut tout simplement le recréer, avec la même valeur, et lui donner une date d'expiration passée.

23.LA PROGRAMMATION OBJET

23.1.Présentation

Bien que le Javascript ne soit pas un langage orienté objet, il donne la possibilité de créer ses propres objets.

23.2.Déclaration d'une 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 on l'a vu, il désigne l'objet en cours.

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 23.1 :

```
function Eleve (Age,Sexe,Ville) {
    this.age = Age ;
    this.sexe = Sexe ;
    this.ville = Ville ;
    }

var Laurent = new Eleve(17,'M','Grenoble') ;
```

23.3.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. 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**[10] !

Syntaxe :

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

Exemple 23.2 :

```
function Eleve (Nom,Age) {
    this.age = Age ;
    this.nom = Nom ;
    this.affich = affich_infos;
    }
function affich_infos () {
    document.write (this.nom + " a " + this.age + "
    ans.");
    }
var Laurent = new Eleve(17, 'Laurent') ;
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 23.3 :

```
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() ;
```

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

24.LES EXPRESSIONS REGULIERES

24.1.Présentation

Les expressions régulières sont assez compliquées à comprendre. Cependant, une fois que le principe est compris, on aura souvent intérêt à aller chercher des expressions régulières toutes faites afin d'éviter de réinventer la roue. Il ne faudra pas dans ce cas s'abstenir de tester correctement ce qui est récupéré.

24.2.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 à vous 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 :

Syntaxe :

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

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

Syntaxe :

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

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

24.3.Paramètres d’une expression régulière

24.3.1.L’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

24.3.2.Le 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 que vous cherchez – pour une raison ou une autre – dans une chaîne.

La chaîne pattern est extensible à l’infini. Le choix de ce qu’elle contient vous appartient. Il s’agit du motif de la chaîne, c’est-à-dire des caractères que vous choisirez d’inclure ou d’exclure de votre 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 Tab.24.1.

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 caractères 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.24.1 : Motifs des expressions régulières.

Les différents motifs ne sont pas séparés. On les mets 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.

24.3.3.Exemples

Exemple 24.1 :

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

L'exemple 24.1 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.

Exemple 24.2 :

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

L'exemple 24.2 est équivalent à l'exemple 24.1, mais il utilise l'autre notation.

Exemple 24.3 :

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

Cet exemple 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.

24.4.Utilisation d'une expression régulière

24.4.1.Les méthode 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 en créer un nouveau.

Exemple 24.3 :

```
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 24.3 propose de tester la correction de votre mail. On retrouve le motif de l'exemple 24.2. La méthode test() est appliquée à l'adresse e-mail. Le reste vous est – ou devrait vous être – familier.

24.4.2.Les méthode de l'objet String

Les méthodes de l'objet String vont aussi aider à la manipulation des expressions régulières. Nous introduirons ici 3 nouvelles méthodes de l'objet String[11]. A ces trois méthodes, nous rajouterons la plupart des méthodes String.

- 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 ci-après.

Syntaxe :

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

[11] 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. De plus, il est possible d'indiquer simplement le motif, avec la notation peu utilisée des slash, comme paramètre.

Syntaxe :

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

Les méthodes split() et match() renvoient chacune un tableau de toutes les occurrences trouvées.

Exemple 24.4 :

```
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 24.4 propose de supprimer les accents d'une chaîne. On remplace tout simplement les lettres accentuées par la lettre sans accent.

24.5.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 24.5 :

```
<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()  {
      var chaine = document.form1.input.value;//récupération de la chaine
      //motif avec les lettres
      var exp = new RegExp ('[A-Za-zìïîòöôüûùàäâéèêë]+',"g");
      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.ANNEXE : FONCTIONS ET PROPRIÉTÉS

25.1.Présentation

Fonctions intégrées au langage ne dépendant d'aucun objet.

25.2.Les fonctions du langage

25.2.1.Escape()

Cette fonction encode les caractères spéciaux d'une chaîne, selon le code %xx, et retourne cette chaîne encodée.

Syntaxe :

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

Exemple 25.1 :

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

25.2.2.Unescape()

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

Syntaxe :

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

Exemple 25.2 :

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

25.2.3.ParseFloat ()

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

Syntaxe :

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

Exemple 25.3 :

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

25.2.4.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.

Syntaxe :

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

Exemple 25.4 :

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

25.2.5.IsFinite ()

Cette fonction renvoie true si le nombre est fini, sinon, elle renvoie false.

Syntaxe :

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

Exemple 25.5 :

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

25.2.6.IsNaN ()

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

Syntaxe :

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

Exemple 25.6 :

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

25.3.Méthodes et propriétés des objets

Les objets de Javascript ou que vous avez créés possèdent deux propriétés et deux méthodes en commun. Elles permettent de manipuler ces objets et de connaître certaines de leur caractéristiques.

25.3.1.Prototype

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

Syntaxe :

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

L'exemple 25.7 rajoute une propriété et une méthode à l'objet Array.

Exemple 25.7 :

```
var tab = new Array(5);
for (i = 0; i < 5; i++) tab[i] = i+1;

Array.prototype.comment = null; // on rajoute un commentaire
tab.comment = "Tableau de 5 chiffres";

function FirstElement () { /* fonction retournant le premier
élément */
      return this[0];
      }
Array.prototype.firstElement = FirstElement;
document.write (tab.comment + " : premier élément : " +
tab.firstElement());
```

25.3.2.Constructor

Cette propriété renvoie le constructeur de l'objet.

Syntaxe :

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

Exemple 25.8 :

```
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/>");
```

25.3.3.ValueOf()

Cette méthode renvoie tout simplement la valeur de l'objet.

Syntaxe :

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

Exemple 25.9 :

```
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/>");
```

25.3.4.ToString()

Cette méthode retourne la description de l'objet.

Syntaxe :

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

Exemple 25.9 :

```
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/>");
```