JAVASCRIPT

Table des matières

Introduction

Qu'est-ce que le JavaScript?

Le JavaScript est une extension du langage html qui est incluse dans le code. Ce langage de programmation apporte des améliorations au langage html en permettant d'exécuter des commandes.

Ne pas confondre JavaScript et Java.

Contrairement au langage Java, le code JavaScript est directement écrit dans la page html, c'est un langage qui ne permet aucune confidentialité au niveau des codes (ceux-ci sont effectivement visibles).
Un applet Java (le programme) doit être compilé à chaque chargement de la page, d'où un important ralentissement pour les applets Java contrairement au JavaScript.

JavaScript	Java
Langage interprété	Langage compilé
Code intégré au html	Code (applet) à part du document html, appelé à partir de la page
Langage peu typé	Langage fortement typé (déclaration du type de variable)
Liaisons dynamiques : les références des objets sont vérifiées au chargement	Liaisons statiques : Les objets doivent exister au chargement (compilation)
Accessibilité du code	Confidentialité du code
Sûr : ne peut pas écrire sur le disque dur	Sûr : ne peut pas écrire sur le disque dur

Le JavaScript est *case sensitive* (*sensible à la casse*), c'est-à-dire qu'il fait une différence entre un nom contenant ou non des majuscules. Ainsi la fonction *bonjour();* n'est pas la même fonction que *Bonjour()*; .
Enfin, comme en langage C, chaque instruction se termine par un point-virgule (;).

A quoi ressemble un script?

Un script est une portion de code qui vient s'insérer dans une page html. Le code du script n'est toutefois pas visible dans la fenêtre du navigateur car il est compris entre des balises (ou tags) spécifiques qui signalent au navigateur qu'il s'agit d'un script écrit en langage JavaScript.
Les balises annonçant un code JavaScript sont les suivantes :

```
<script type="text/javascript" >
Placez ici le code de votre script
</script>
```

Ajouter des commentaires dans votre code

Comme dans tout langage de programmation, il est bon d'ajouter des commentaires dans les scripts :

- d'une part pour s'y retrouver lorsque l'on voudra revoir le script
- d'autre part pour permettre comprendre le script.

Ne pas confondre les balises de commentaires du langage html et les caractères de commentaires !
Pour écrire des commentaires, JavaScript utilise les conventions utilisées en langage C/C++
Pour mettre en commentaires toute une ligne on utilise le double slash :

```
// Tous les caractères derrière le // sont ignorés
```

Pour mettre en commentaire une partie du texte (éventuellement sur plusieurs lignes) on utilise /* et */ :

```
/* Toutes les lignes comprises entre ces repères sont ignorées par
l'interpréteur de code */
```

La mise en commentaire d'une partie du code est utile pour la mise au point des programmes (permet d'éliminer provisoirement sans effacer).

Un exemple de JavaScript

On va faire afficher un boite de dialogue suite au chargement d'une page html.
Voici la page html correspondante :

```
<html>
<head> <title> Voici une page contenant du JavaScript </title> </head>
<body>
<script type="text/javascript" >
alert("Voici un message d alerte !");
// et voici un commentaire
</script> </body> </html>
```

Testez le résultat d'un tel script lors du chargement de la page ci-dessus

Implantation du code

Où insérer le JavaScript dans votre page html

Il y a plusieurs façon d'inclure du JavaScript dans une page html :

- dans la balise <script>
- dans un fichier externe
- grâce aux événements

Dans la balise script

Le code JavaScript peut être inséré où vous le désirez dans votre page Web, vous devez toutefois veiller à ce que le navigateur soit entièrement chargé votre script avant d'exécuter une instruction. En effet, lorsque le navigateur charge votre page Web, il la traite de haut en bas, de plus vos visiteurs (souvent impatients) peuvent très bien interrompre le chargement d'une page, auquel cas si l'appel d'une fonction se situe avant la fonction dans votre page il est probable que cela génèrera une erreur si cette fonction n'a pas été chargée.
Ainsi, on place généralement le maximum d'éléments du script dans l'en-tête (ce sont les éléments situés entre les balises <head> et </head>). Les évènements JavaScript seront quant à eux placés dans le corps de la page (entre les balises <body> et </body>) comme attributs d'une commande html.

```
<script type="text/javascript" >
Placez ici le code de votre script
</script>
```

L'argument de la balise <script> décrit le langage utilisé.

Dans un fichier externe

Il est possible de mettre les codes de JavaScript en annexe dans un fichier.
Le code à insérer est le suivant :

```
<script type="text/javascript" src="url/fichier.js"> </script>
```

Où url/fichier.js correspond au chemin d'accès au fichier contenant le code en JavaScript, sachant que si celui-ci n'existe pas le navigateur exécutera le code inséré entre les 2 balises.

Grâce aux évènements

On appelle évènement une action de l'utilisateur, comme le clic d'un des boutons de la souris.
Le code dans le cas du résultat d'un événement s'écrit :

```
<balise eventHandler="code JavaScript à insérer">
```

eventHandler représente le nom de l'événement.

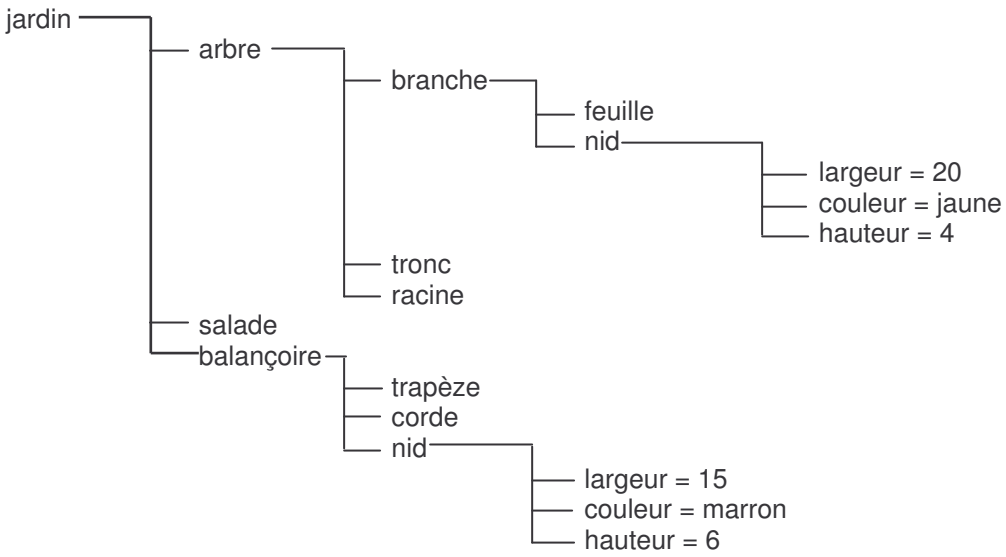
La notion d'objet

Pourquoi parle-t-on d'objets?

JavaScript traite les éléments qui s'affichent dans votre navigateur comme des objets, c'est-à-dire des éléments

- classés selon une hiérarchie pour pouvoir les désigner précisément
- auxquels on associe des propriétés et des méthodes.

Voici un exemple concret :
Imaginez un arbre dans un jardin comportant une branche sur laquelle se trouve un nid. On suppose la hiérarchie d'objets est définie comme suit :



Le nid sur l'arbre est désigné par **jardin.arbre.branche.nid**
Contrairement au nid situé sur la balançoire **jardin.balançoire.nid**
Imaginez maintenant que l'on veuille changer la couleur du nid (dans l'arbre) pour le peindre en vert, il suffirait de taper une commande du genre : **jardin.arbre.branche.nid.couleur = vert;**
C'est donc ainsi que l'on représente les objets en JavaScript, à la seule différence que ce n'est pas un jardin qui contient des objets mais la fenêtre du navigateur...

Les objets du navigateur

La page du navigateur contient des objets, on peut accéder à n'importe lesquels d'entre eux et les manipuler par l'intermédiaire de leurs propriétés ou méthodes.
On commence généralement par l'objet le plus élevé dans la hiérarchie (celui contenant tous les autres) puis on descend dans la hiérarchie jusqu'à arriver à l'objet voulu !
L'objet le plus élevé dans la hiérarchie est l'objet fenêtre (les objets en JavaScript ont leur dénomination en anglais, donc dans le cas présent ***window***)
Dans la fenêtre s'affiche une page, c'est l'objet ***document***
Cette page peut contenir plusieurs objets, comme des formulaires, des images, ...
Un formulaire peut contenir par exemple :

- une case à cocher (appelé *checkbo*x et nommée case) et
- un champ de texte (appelé *textarea* et nommé champ_texte).

La case à cocher est repérée par : **window.document.forms[0].case**
Le champ de texte est repéré par : **window.document.forms[0].champ_texte**
La case à cocher a entre autre une propriété *checke*d, qui retourne la valeur 1 si elle est cochée, 0 dans le cas contraire.
Le champ de texte a comme propriétés parmi d'autres :

- ***name*** : le nom du champ de texte
- ***value*** : le texte contenu dans le champ
- ***size*** : la taille du champ de texte

On peut modifier le texte associé au champ de texte que l'on a nommé champ_text par
window.document.forms[0].champ_texte.value

Les variables

Le concept de variable

Une variable est un "contenant" repéré par son nom, pouvant contenir des données qui pourront être modifiées lors de l'exécution du programme. Une variable est définie par un nom (identificateur), un type et une valeur.
En JavaScript, les noms de variables peuvent être aussi long que l'on désire, mais doivent répondre à certains critères :

- un nom de variable doit commencer par une lettre (majuscule ou minuscule) ou un "_"
- un nom de variable peut comporter des lettres, des chiffres et les caractères _ et & (espaces non autorisés !)
- Les noms de variables ne peuvent pas être les noms suivants (qui sont des noms réservés) :
  - abstract
  - boolean break byte
  - case catch char class
  - default do double
  - else extends
  - false final finally float
  - goto
  - if, implements, import, in, instanceof, int, interface
  - long
  - native, new, null
  - package, private, protected, public
  - return
  - short, static, super, switch, synchronized
  - this, throw, throws, transient, true, try
  - var, void
  - while, with

Nom de variable correct	Nom de variable incorrect	Raison
Variable	Nom de Variable	comporte des espaces
Nom_De_Variable	123	commence par un chiffre
nom_de_variable	toto@mailcity.com	caractère spécial @
nom_de_variable_123	Nom-de-variable	signe - interdit
_707	transient	nom réservé

- Les noms de variables sont sensibles à la casse : JavaScript fait la différence entre un nom en majuscules et un nom en minuscules. Il faut donc veiller à utiliser des noms comportant la même casse !
- Il est conseillé de choisir des noms de variables qui rappelle sa fonction dans le programme. Si plusieurs mots sont utilisés, choisissez un nom utilisant une contraction.

Exemple : la variable contenant le nombre de coups joués pourra être nommée NbreCoupJoue ou nbre_coup_joue

La déclaration de variables

JavaScript étant très souple au niveau de l'écriture (à double-tranchant car il laisse passer des erreurs), la déclaration des variables peut se faire de deux façons :

- de façon explicite, en faisant précéder la variable du mot clé *var* qui permet d'indiquer de façon rigoureuse qu'il s'agit d'une variable :
```
var chaine= "bonjour"
```
- de façon implicite, en écrivant directement le nom de la variable suivie du caractère = et de la valeur à affecter :
```
chaine= "bonjour"
```

Le navigateur détermine seul qu'il s'agit d'une déclaration de variable.

Même si une déclaration implicite est reconnue par le navigateur, il est tout de même plus rigoureux (et plus rapide à l'exécution) de déclarer les variables de façon explicite avec le mot *var*.

On peut déclarer plusieurs variables avec le même mot-clé var.

Exemple : var i = 0, age = 15, nom = "Durand"

Portée (visibilité) des variables

Selon l'endroit où on déclare une variable, celle-ci pourra être accessible (visible) de partout dans le script ou bien que dans une portion déterminée du code, on parle de "portée" d'une variable.

Lorsqu'une variable est déclarée sans le mot clé *var*, c'est-à-dire **de façon implicite**, elle est accessible de partout dans le script (n'importe quelle fonction du script peut faire appel à cette variable). On parle alors de **variable globale**

Lorsque l'on déclare **de façon explicite** une variable en JavaScript (en précédant sa déclaration avec le mot *var*), sa portée dépend de l'endroit où elle est déclarée.

- Une variable déclarée au début du script, c'est-à-dire avant toute fonction sera globale, on pourra alors l'utiliser à partir de n'importe quel endroit dans le script
- Une variable déclarée par le mot clé *var* dans une fonction aura une portée limitée à cette seule fonction, c'est-à-dire qu'elle est inutilisable ailleurs, on parle alors de variable locale

```
var bidule = "dehors";
function bizarre()
{var bidule = "dedans";
document.write (bidule);}
document.write(bidule);          //affiche "dehors"
bizarre();                       //affiche "dedans"
```

Les types de données dans les variables

En JavaScript il n'y a pas besoin de déclarer le type de variables que l'on utilise, contrairement à C ou Java pour lesquels il faut préciser s'il s'agit d'entier *(int),* de nombre à virgule flottante *(float)*, de caractères *(char), ...*

En fait le JavaScript n'autorise la manipulation que de 4 types de données :

- des nombres : entiers ou à virgules
- des chaînes de caractères (string) : une suite de caractères
- des booléens : des variables à deux états permettant de vérifier une condition
- des variables de type *null* : un mot caractéristique pour indiquer qu'il n'y a pas de données

Nombre entier

Un nombre entier est un nombre sans virgule qui peut être exprimé dans différentes bases :

- décimale : l'entier est représenté par une suite de chiffres (de 0 à 9) ne devant pas commencer par 0
- hexadécimale : l'entier est représenté par une suite de chiffres (0 à F ou 0 à f) devant commencer par 0X ou 0x
- octale : l'entier est représenté par une suite de chiffres (de 0 à 7) devant commencer par 0

Nombre à virgule *(float)*

Un nombre à virgule flottante est un nombre à virgule, il peut toutefois être représenté de différentes façons :

- un entier décimal : 895
- un nombre comportant un point (et non une virgule) : 845.32
- une fraction : 27/11
- un nombre exponentiel, c'est-à-dire un nombre (éventuellement à virgule) suivi de la lettre *e* (ou E), puis d'un entier correspondant à la puissance de 10 (signé ou non, c'est-à-dire précédé d'un + ou d'un -)
  - 2.75e-2
  - 35.8E+10
  - .25e-2

Chaîne de caractères *(string)*

Une chaîne de caractère est représentée par une suite de caractères encadrée par des guillemets simples (') ou doubles ("), sachant que les deux types de guillemets ne peuvent être mélangés pour une même chaîne de caractères, ce qui signifie que les guillemets dans une chaîne de caractères existent par paire.
Il existe des caractères spéciaux à utiliser dans les chaînes pour simuler d'une part des caractères non visuels ou pour éviter au navigateur de confondre les caractères d'une chaîne avec ceux du script, ces caractères sont précédés d'un antislash (\) :

- \b : touche de suppression
- \f : formulaire plein
- \n : retour à la ligne
- \r : appui sur la touche *ENTREE*
- \t : tabulation
- \" : guillemets doubles
- \' : guillemets simples
- \\ : caractère antislash

Ainsi, si on veut stocker dans la variable *Titre* la chaîne suivante:

```
Qu'y a-t-il dans "c:\windows\"
```

Il faudra écrire dans le code JavaScript :

```
Titre = "Qu\'y a-t-il dans \"c:\\windows\\\"";  ou
Titre = 'Qu\'y a-t-il dans \"c:\\windows\\\"';
```

Booléens *(booleans)*

Un booléen est une variable spéciale servant à évaluer une condition, il peut donc avoir deux valeurs :

- true : si le résultat est vrai
- false : si le résultat est faux

Chaîne de caractères

L'objet String

string signifie *"chaîne"*, il s'agit en fait de chaîne de caractères.
Les propriétés et méthodes de l'objet String permettent la manipulation de chaînes de caractères.

Les propriétés de l'objet String

L'objet *string* a une seule propriété : la propriété *length* qui permet de retourner la longueur d'une chaîne de caractères.
La syntaxe de la propriété *length* est la suivante :

```
x = nom_de_la_chaine.length;
x = ('chaine de caracteres').length;
```

On peut directement passer la chaîne de caractères comme objet, en délimitant la chaîne par des apostrophes et en plaçant le tout entre parenthèses.
La méthode consistant à appliquer une propriété à une variable de type *string* fonctionne bien évidemment aussi.

Les méthodes de l'objet String

Les méthodes de l'objet *string* permettent de récupérer une portion d'une chaîne de caractère, ou bien de la modifier.
Pour comprendre les méthodes suivantes, il est nécessaire de comprendre comment est stockée une chaîne de caractères.
Il s'agit en fait d'un tableau constitué de *n* caractères *(n* est donc le nombre de caractères), on note 0 la position du premier caractère (celui à l'extrême gauche), puis on les compte de gauche à droite en incrémentant ce nombre :
Chaîne C o m m e n t ça m a r c h e ?
Position des caractères 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
Comme on peut le voir, une chaîne ayant *n* caractères aura son dernier caractère en position *n-1* (ici, pour une chaîne de 18 caractères le dernier élément est à la position 17...).

Méthode	Description
charAt(chaîne, position) ou chaîne**.charAt**(position)	Retourne le caractère situé à la position donnée en paramètre
indexOf(sous-chaîne, position)	Retourne la position d'une sous-chaîne (lettre ou groupe de lettres) dans une chaîne de caractère, en effectuant la recherche de gauche à droite, à partir de la position spécifiée en paramètre.
lastIndexOf(sous-chaîne, position) similaire à *indexOf()*, à la différence que la recherche se fait de droite à gauche :	Retourne la position d'une sous-chaîne (lettre ou groupe de lettres) dans une chaîne de caractère, en effectuant la recherche de droite à gauche, à partir de la position spécifiée en paramètre.
substring(position1, position2)	Retourne la sous-chaîne (lettre ou groupe de lettres) comprise entre les positions 1 et 2 données en paramètre.

toLowerCase()	Convertit tous les caractères d'une chaîne en minuscules
toUpperCase()	Convertit tous les caractères d'une chaîne en majuscules

D'autres méthodes

Méthode	Description
anchor(nom_a_donner)	Transforme le texte en ancrage html
big()	Augmente la taille de la police
blink()	Transforme la chaîne en texte clignotant
bold()	Met le texte en gras (balise <B>)
fixed	Transforme le texte en `police fixe` (balise <TT>)
fontcolor(couleur)	Modifie la couleur du texte (admet comme argument la couleur en hexadécimal ou en valeur littérale)
fontsize(Size)	Modifie la taille de la police, en affectant la valeur passée en paramètre
italics()	Transforme le texte en italique (balise <I>)
link(URL)	Transforme le texte en hypertexte (balise <A href>)
small()	Diminue la taille de la police
strike()	Transforme le texte en texte barré (balise <strike>)
sub()	Transforme le texte en indice (balise <sub>)
sup()	Transforme le texte en exposant (balise <sup>)

Exemples d'utilisation des méthodes de l'objet String

Méthode charAt()

Il existe deux syntaxes pour la méthode **charAt()**

```
x = "chaine de caractères"; Resultat = x.charAt(position);
Resultat = charAt("chaine de caractères", position);
```

Le paramètre *position* est un entier qui représente la position du caractère à récupérer, il doit être compris entre 0 et *n-1* (où *n* représente la longueur de la chaîne). Dans le cas contraire (le paramètre *position* négatif ou supérieur ou égal à la longueur) **charAt()** renvoie une chaîne de longueur nulle.

Voici quelques exemples :

```
Chaine = 'Comment ça marche?'
var Resultat = charAt(Chaine,0) donne 'C'
var Resultat = charAt("Comment ça marche?", 1) donne 'o'
var Resultat = Chaine.charAt(17) donne '?'
var Resultat = ("Comment ça marche?").charAt(18) donne ""
var Resultat = charAt(Chaine, -1) donne ""
```

Méthode indexOf()

La méthode **indexOf()** permet de rechercher (de gauche à droite) la position d'une sous-chaîne dans une chaîne de caractères.

```
Chaine = "chaine de caractères"; Sous-Chaine = "sous-chaîne de caractères";
Resultat = x.indexOf(position);
```

La position indiquée en argument permet de déterminer la position du caractère à partir duquel la recherche est effectuée. L'argument *position* doit être compris entre 0 et *n*-1. Si cet argument est omis la recherche débutera à la position 0.

Lorsque la recherche est infructueuse, la méthode **indexOf()** renvoie la valeur -1.

Voici quelques exemples :

```
Chaine = 'Comment ça marche?' Sous_Chaine = 'mar'
var Resultat = Chaine.indexOf(Sous_Chaine, 6) donne '11'
var Resultat = Chaine.indexOf(Sous_Chaine) donne '11'
var Resultat = Chaine.indexOf(Sous_Chaine, 11) donne '11'
var Resultat = Chaine.indexOf(Sous_Chaine, 12) donne '-1'
var Resultat = Chaine.indexOf(Sous_Chaine, -1) donne "-1"
var Resultat = Chaine.indexOf(Sous_Chaine, 15) donne "-1"].
var Resultat = Chaine.indexOf(Sous_Chaine, 19) donne "-1"
```

Méthode lastIndexOf()

La méthode **lastIndexOf()** permet de rechercher (de droite à gauche) la position d'une sous-chaîne dans une chaîne de caractères.

```
Chaine = "chaine de caractères"; Sous-Chaine = "sous-chaîne de caractères";
Resultat = x.lastIndexOf(position);
```
La position indiquée en argument permet de déterminer la position du caractère à partir duquel la recherche est effectuée (vers la gauche pour cette méthode). L'argument *position* doit être compris entre 0 et *n*-1. Si cet argument est omis la recherche débutera à partir de la fin de la chaîne.
Lorsque la recherche est infructueuse, la méthode **lastIndexOf()** renvoie la valeur -1.
Voici quelques exemples :
```
Chaine = 'Comment ça marche?' Sous_Chaine = 'mar'
Chaine.lastIndexOf(Sous_Chaine, 6) donne '-1'
Chaine.lastIndexOf(Sous_Chaine) donne '11'
Chaine.lastIndexOf(Sous_Chaine, 11) donne '11'
Chaine.lastIndexOf(Sous_Chaine, 12) donne '11'
Chaine.lastIndexOf(Sous_Chaine, -1) donne "-1"
Chaine.lastIndexOf(Sous_Chaine, 19) donne "-1"
```
Méthode substring()
La méthode **substring()** permet de récupérer une sous-chaîne dans une chaîne de caractères en précisant en paramètres les positions des caractères entre lesquels la sous-chaîne doit être récupérée.
```
Chaine = "chaine de caractères"; Resultat = x.substring(position1,
position2);
```
Les positions indiquées en argument permettent de déterminer les positions des caractères entre lesquels la sous-chaîne doit être récupérée. Les arguments *position1* et *position2* doivent être compris entre 0 et *n*-1.
- Si l'argument *position1* est plus petit que l'argument *position*2, la méthode **substring()** retourne la sous-chaîne commençant à la position 1 et s'arrêtant au caractère situé avant position 2
- Si l'argument *position1* est plus grand que l'argument *position*2, la méthode **substring()** retourne la sous-chaîne commençant à la position 2 et s'arrêtant au caractère situé avant position 1
- Si l'argument *position1* est égal à l'argument *position*2, la méthode **substring()** retourne une chaîne vide

```
Chaine = 'Comment ça marche?'
var Resultat = Chaine.substring(1,5) donne 'omme'
var Resultat = Chaine.substring(6,6) donne ''
var Resultat = Chaine.substring(8,2) donne 'mment '
```
Méthodes toLowerCase() et toUpperCase()
La méthode **toLowerCase()** permet de convertir toutes les lettres d'une chaîne en minuscules, les autres caractères sont laissés tels quels.
```
Chaine = 'Comment ça Marche?'
var Resultat = Chaine.toLowerCase donne 'comment ça marche?'
```
La méthode **toUpperCase()** permet de convertir toutes les lettres d'une chaîne en majuscules, les autres caractères sont laissés tels quels.
Voici quelques exemples :
```
Chaine = 'Comment ça Marche?'
var Resultat = Chaine.toUpperCase donne 'COMMENT çA MARCHE?'
```
Les événements
Qu'appelle-t-on un événement?
Les évènements sont des actions de l'utilisateur qui vont pouvoir donner lieu à une interactivité.
L'événement clic de souris est le seul que le html gère. Grâce à JavaScript il est possible d'associer des fonctions, des méthodes à des événements tels que le passage de la souris au-dessus d'une zone, le changement d'une valeur, ...
Ce sont les gestionnaires d'événements qui permettent d'associer une action à un événement. La syntaxe d'un gestionnaire d'événement est la suivante :
```
onevenement="Action_JavaScript_ou_Fonction();"
```
Les gestionnaires d'événements sont associés à des objets, et leur code s'insère dans la balise de ceux-ci.
Liste de quelques événements

Evénement	Gestionnaire d'événement	Description
Click	onclick	l'utilisateur clique sur l'élément associé à l'événement
Load	onload	le navigateur de l'utilisateur charge la page en cours
Unload	onunload	le navigateur de l'utilisateur quitte la page en cours
MouseOver	onmouseover	l'utilisateur positionne le curseur de la souris au-dessus d'un élément
MouseOut	onmouseout	le curseur de la souris quitte un élément

Focus	onfocus	l'utilisateur donne le focus à un élément, c'est-à-dire que cet élément est sélectionné comme étant l'élément actif
Blur	onblur	l'élément perd le focus, c'est-à-dire que l'utilisateur clique hors de cet élément, celui-ci n'est alors plus sélectionné comme étant l'élément actif
Change	onchange	l'utilisateur modifie le contenu d'un champ de données
Select	onselect	l'utilisateur sélectionne un texte (ou une partie d'un texte) dans un champ de type "text" ou "textarea"
Submit	onsubmit	Se produit lorsque l'utilisateur clique sur le bouton de soumission d'un formulaire (le bouton qui permet d'envoyer le formulaire)

Objets auxquels on peut associer des événements

Chaque événement ne peut pas être associé à n'importe quel objet. Un événement *onChange* ne peut pas s'appliquer à un lien hypertexte

Objet	Evénements associables
Lien hypertexte	onclick, onmouseover, onmouseout
Page du navigateur	onload, onunload
Bouton, Case à cocher, Bouton radio, Bouton Reset	onclick
Liste de sélection d'un formulaire	onblur, onchange, onfocus
Bouton Submit	onsubmit
Champ de texte et zone de texte	onblur, onchange, onfocus, onselect

Quelques exemples d'événements

Le mieux pour apprendre à se servir des événements est de s'entraîner à écrire de petits codes...
Pour vous inspirer, regardez les fichiers sources de certaines pages web, mais pensez toujours à respecter les auteurs des codes en ne faisant pas un copier-coller de leurs scripts sans leur accord (il est généralement de bon ton de citer la source du JavaScript que l'on récupère...).

Ouverture d'une boite de dialogue lors d'un click

Le code correspondant à une boîte de dialogue est très simple : `window.alert("Votre Texte");`
Il s'agit donc de le mettre dans la balise d'un lien hypertexte :

```
<html> <head>
<title>Ouverture d'une boite de dialogue lors d'un click</title> </head>
<body>
<a href="JavaScript :;" onclick="window.alert('Message d\'alerte a utiliser
avec moderation');">Cliquez ici !</a>
</body> </html>
```

Le gestionnaire d'événement *onclick* a été inséré dans la balise de lien hypertexte `<a href...>`
Le but du script est de faire apparaître une boîte de dialogue, on ne désire pas que le lien nous entraîne sur une autre page : il faut insérer "JavaScript :;" dans l'attribut *href* pour signaler au navigateur que l'on désire rester sur la page en cours. Ne pas mettre un attribut vide au risque de révéler le contenu de votre répertoire à vos visiteurs.
Remarquez l'emploi de \' dans la phrase "Message d'alerte a utiliser avec modération". Le signe antislash (\) précédant le guillemet permet de signaler au navigateur qu'il ne faut pas l'interpréter comme un délimiteur de chaîne mais comme un simple caractère pour éviter qu'il génère une erreur !

Modification d'une image lors du survol d'un lien par le pointeur de la souris

On peut utiliser le gestionnaire *onmouseover* pour créer un menu interactif qui se modifie au passage de la souris. On peut même ajouter le gestionnaire *onmouseout* pour rétablir l'image originale lorsque le curseur quitte l'image (Son utilisation est limitée aux navigateurs supportant JavaScript 1.1 et supérieur !).

```
<html> <head>
<title>Modification d'une image lors du passage de la souris</title> </head>
<body>
<a href="JavaScript :;" onmouseover="document.img_1.src='image2.gif';"
onmouseout="document.img_1.src='image1.gif';"> <img name="img_1"
src="image1.gif"> </a>
</body> </html>
```

Pour pouvoir associer un événement à une image il faut que celle-ci soit considérée comme un lien, ainsi on place la balise `<img ...>` entre les balises `<a ...> et </a>`

Les opérateurs

Qu'est-ce qu'un opérateur?

Les opérateurs sont des symboles qui permettent de manipuler des variables, ie d'effectuer des opérations, les évaluer...
On distingue plusieurs types d'opérateurs :

- les opérateurs de calcul
- les opérateurs d'assignation
- les opérateurs d'incrémentation
- les opérateurs de comparaison
- les opérateurs logiques
- les opérateurs bit-à-bit
- les opérateurs de rotation de bit

Les opérateurs de calcul

Les opérateurs de calcul permettent de modifier mathématiquement la valeur d'une variable

Opérateur	Dénomination	Effet	Exemple	Résultat (x vaut 7)
+	addition	Ajoute deux valeurs	x+3	10
-	soustraction	Soustrait deux valeurs	x-3	4
*	multiplication	Multiplie deux valeurs	x*3	21
/	division	Divise deux valeurs	x/3	2.3333333
%	modulo	Reste de la division entière	x%3	1
=	affectation	Affecte une valeur à une variable	x=3	Met la valeur 3 dans la variable x

Les opérateurs d'assignation

Ces opérateurs permettent de simplifier des opérations telles qu'ajouter *une valeur dans une variable et stocker le résultat dans la variable*. Une telle opération s'écrirait habituellement de la façon suivante par exemple : *x=x+2* Avec les opérateurs d'assignation il est possible d'écrire cette opération sous la forme suivante : *x+=2* Ainsi, si la valeur de x était 7 avant opération, elle sera de 9 après.
Les autres opérateurs du même type sont les suivants :

Opérateur	Effet	Exemple
+=	addition deux valeurs et stocke le résultat dans la variable (à gauche)	x+=2 équivaut à x=x+2
-=	soustrait deux valeurs et stocke le résultat dans la variable	x-=2 équivaut à x=x-2
*=	multiplie deux valeurs et stocke le résultat dans la variable	x*=2 équivaut à x=x*2
/=	divise deux valeurs et stocke le résultat dans la variable	x/=2 équivaut à x=x/2

Les opérateurs d'incrémentation

Opérateur	Dénomination	Effet	Syntaxe	Résultat (x vaut 7)
++	Incrémentation	Augmente d'une unité la variable	x++	8
--	Décrémentation	Diminue d'une unité la variable	x--	6

Ce type d'opérateur permet de facilement augmenter ou diminuer d'une unité une variable. Ces opérateurs sont très utiles pour des structures telles que des boucles, qui ont besoin d'un compteur (variable qui augmente de un en un).
Un opérateur de type `x++` permet de remplacer des notations telles que `x=x+1` ou bien `x+=1`

Les opérateurs de comparaison

Opérateur	Dénomination	Effet	Exemple	Résultat (x vaut 7)
== A ne pas confondre avec le signe d'affectation (=)	opérateur d'égalité	Compare deux valeurs et vérifie leur égalité	x==3	1 si x est égal à 3, sinon 0
<	opérateur d'infériorité stricte	Vérifie qu'une variable est strictement inférieure à une valeur	x<3	1 si x est inférieur à 3, sinon 0
<=	opérateur d'infériorité	Vérifie qu'une variable est inférieure ou égale à une valeur	x<=3	1 si x est inférieur à 3, sinon 0
>	opérateur de	Vérifie qu'une variable est	x>3	1 si x est supérieur à

	supériorité stricte	strictement supérieure à une valeur		3, sinon
>=	opérateur de supériorité	Vérifie qu'une variable est supérieure ou égale à une valeur	x>=3	1 si x est supérieur ou égal à 3, sinon 0
!=	opérateur de différence	Vérifie qu'une variable est différente d'une valeur	x !=3	1 si x est différent de 3, sinon 0

Les opérateurs logiques (booléens)

Ce type d'opérateur permet de vérifier si plusieurs conditions sont vraies :

Opérateur	Dénomination	Effet	Syntaxe
||	OU logique	Vérifie qu'une des conditions est réalisée	((condition1)||(condition2))
&&	ET logique	Vérifie que toutes les conditions sont réalisées	((condition1)&&(condition2))
!	NON logique	Inverse l'état d'une variable booléenne (retourne la valeur 1 si la variable vaut 0, 0 si elle vaut 1)	(!condition)

Les opérateurs bit-à-bit

Ce type d'opérateur traite ses opérandes comme des données binaires, plutôt que des données décimales, hexadécimales ou octales. Ces opérateurs traitent ces données selon leur représentation binaire mais retournent des valeurs numériques standards dans leur format d'origine.
Les opérateurs suivants effectuent des opérations bit-à-bit, c'est-à-dire avec des bits de même poids
.

Opérateur	Dénomination	Effet	Syntaxe	Résultat (x vaut 7)
&	ET bit-à-bit	Retourne 1 si les deux bits de même poids sont à 1	9 & 12 (1001 & 1100)	8 (1000)
|	OU bit-à-bit	Retourne 1 si l'un ou l'autre des deux bits de même poids est à 1 (ou les deux)	9 | 12 (1001 | 1100)	13 (1101)
^	OU bit-à-bit	Retourne 1 si l'un des deux bits de même poids est à 1 (mais pas les deux)	9 ^ 12 (1001 ^ 1100)	5 (0101)

Les opérateurs de rotation de bit

Ce type d'opérateur traite ses opérandes comme des données binaires d'une longueur de 32 bits, plutôt que des données décimales, hexadécimales ou octales. Ces opérateurs traitent ces données selon leur représentation binaire mais retournent des valeurs numériques standards dans leur format d'origine.
Les opérateurs suivants effectuent des rotations sur les bits, c'est-à-dire qu'il décale chacun des bits d'un nombre de bits vers la gauche ou vers la droite. Le premier opérande désigne la donnée sur laquelle on va faire le décalage, la seconde désigne le nombre de bits duquel elle va être décalée.

Opérateur	Dénomination	Effet	Syntaxe	Résultat (x vaut 7)
<<	Rotation à gauche	Décale les bits vers la gauche (multiplie par 2 à chaque décalage). Les zéros qui sortent à gauche sont perdus, tandis que des zéros sont insérés à droite	6 << 1 (110 << 1)	12 (1100)
>>	Rotation à droite avec conservation du signe	Décale les bits vers la droite (divise par 2 à chaque décalage). Les zéros qui sortent à droite sont perdus, tandis que le bit non-nul de poids plus fort est recopié à gauche	6 >> 1 (0110 >> 1)	3 (0011)
>>>	Rotation droite et remplissage de zéros	Décale les bits vers la droite (divise par 2 à chaque décalage). Les zéros qui sortent à droite sont perdus, tandis que des zéros sont insérés à gauche	6 >>> 1 (0110 >>> 1)	3 (0011)

Les priorités

Lorsque l'on associe plusieurs opérateurs, il faut que le navigateur sache dans quel ordre les traiter, voici donc dans **l'ordre décroissant** les priorités de tous les opérateurs :

()	parenthèses
++ -- !	opérateurs unaires
* / %	mult,div,modulo
+ -	addition,soustraction
<< >>	décalage de bits
< > <= >=	opérateurs relationnels
== !=	égalité
&	ET binaire
^	OU exclusif binaire(XOR)
|	OU binaire
&&	ET logique
||	OU logique
= += -= *= /= %= ^=	Affectations diverses

Cela signifie que , dans une expression complexe, où apparaissent plusieurs opérateurs, JavaScript les interprètera en tenant compte de leur priorité. Si plusieurs priorités se trouvent dans la même expression et sont de même niveau ce sera l'opérateur le plus à gauche qui sera effectué en premier.
Grâce aux parenthèses qui sont toujours prioritaires, vous pouvez définir vous-même l'ordre de calcul des opérateurs. Donc utilisez le plus souvent les parenthèses qui rendront votre programme plus clair, plus lisible.

Structures conditionnelles

Qu'est-ce qu'une structure conditionnelle?

On appelle *structure conditionnelle* les instructions qui permettent de tester si une condition est vraie ou non, ce qui permet de donner de l'interactivité à vos scripts par exemple.

L'instruction if

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

```
if (condition testée)
{liste d'instructions}
```

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

L'instruction if ... else

L'instruction *if* dans sa forme basique ne permet de tester qu'une condition, or la plupart du temps on aimerait pouvoir choisir les instructions à exécuter **en cas de non-réalisation de la condition**...
L'expression *if ... else* permet d'exécuter une autre série d'instruction en cas de non-réalisation de la condition.
La syntaxe de cette expression est la suivante :

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

Il est possible de faire un test avec la structure suivante :

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

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

Les instructions if…else peuvent être imbriquées

Les boucles

Les boucles sont des structures qui permettent d'exécuter plusieurs fois la même série d'instructions jusqu'à ce qu'une condition ne soit plus réalisée.

La façon la plus commune de faire une boucle est de créer un compteur (une variable qui s'incrémente, c'est-à-dire qui augmente de 1 à chaque tour de boucle) et de faire arrêter la boucle lorsque le compteur dépasse une certaine valeur.

La boucle for

L'instruction *for* permet d'exécuter plusieurs fois la même série d'instructions : c'est une boucle ! Dans sa syntaxe, il faut préciser le nom de la variable qui sert de compteur (éventuellement sa valeur de départ), la condition sur la variable pour laquelle la boucle s'arrête (basiquement une condition qui teste si la valeur du compteur dépasse une limite) et enfin une instruction qui incrémente (ou décrémente) le compteur.

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

Par exemple :

```
for(i=1; i<6; i++)
{alert(i)}
```

Cette boucle affiche 5 fois la valeur de i, c'est-à-dire 1, 2, 3, 4 et 5 Elle commence à i = 1, vérifie que i est bien inférieur à 6, etc... jusqu'à atteindre la valeur i=6, pour laquelle la condition ne sera plus réalisée, la boucle s'interrompra et le programme continuera son cours.

- il faudra toujours vérifier que la boucle a bien une condition de sortie (le compteur s'incrémente correctement)
- une instruction `alert(i);` dans une boucle est un bon moyen pour vérifier la valeur du compteur pas à pas !
- il faut bien compter le nombre de fois que l'on veut faire exécuter la boucle :

`for (i=0;i<10;i++)`	exécute 10 fois la boucle (i de 0 à 9)
`for (i=0;i<=10;i++)`	exécute 11 fois la boucle (i de 0 à 10)
`for (i=1;i<10;i++)`	exécute 9 fois la boucle (i de 1 à 9)
`for (i=1;i<=10;i++)`	exécute 10 fois la boucle (i de 1 à 10)

L'instruction while

L'instruction *while* représente un autre moyen d'exécuter plusieurs fois la même série d'instructions.

```
while (condition testée)
{liste d'instructions}
```

Cette instruction exécute la liste d'instructions **tant que** *(while* signifie *tant qu*e) la condition est réalisée.
La condition de sortie pouvant être n'importe quelle structure conditionnelle, les risques de boucle infinie (boucle dont la condition est toujours vraie) existent, c'est-à-dire qu'elle risque de provoquer un plantage du navigateur !

Saut inconditionnel

Il peut être nécessaire de faire sauter à la boucle une ou plusieurs valeurs sans pour autant mettre fin à celle-ci.
La syntaxe de cette expression est `continue` (cette instruction se place dans une boucle !), on l'associe généralement à une structure conditionnelle, sinon les lignes situées entre cette instruction et la fin de la boucle seraient obsolètes.
Exemple : Imaginons que l'on veuille imprimer pour x allant de 1 à 10 la valeur de 1/(x-7). Il est évident que pour x=7 il y aura une erreur. Grâce à l'instruction `continue` il est possible de traiter cette valeur à part puis de continuer la boucle !

```
x=1
while (x<=10)
{if (x == 7)
{alert('division par 0');
continue;}
a = 1/(x-7);
alert(x);
x++}
```

Il y avait une erreur dans ce script... peut-être ne l'avez-vous pas vue : Lorsque x est égal à 7, le compteur ne s'incrémente plus, il reste constamment à la valeur 7, il aurait fallu écrire :

```
x=1
while (x<=10)
{if (x == 7)
{alert('division par 0');
x++;
continue;}
a = 1/(x-7);
alert(x);
x++}
```

Arrêt inconditionnel

A l'inverse, il peut être voulu d'arrêter prématurément la boucle, pour une autre condition que celle précisé dans l'en-tête de la boucle. L'instruction `break` permet d'arrêter une boucle (`for` ou bien `while`).
Il s'agit, tout comme `continue`, de l'associer à une structure conditionnelle, sans laquelle la boucle ne ferait jamais plus d'un tour ! Dans l'exemple de tout à l'heure, par exemple si l'on ne savait pas à quel moment le dénominateur (x-7) s'annule (pour des équations plus compliquées par exemple) il serait possible de faire arrêter la boucle en cas d'annulation du dénominateur, pour éviter une division par zéro !

```
for (x=1; x<=10; x++)
{a = x-7;
if (a == 0)
{alert('division par 0');
break; }
alert(1/a);}
```

Les fonctions

La notion de fonction

On appelle *fonction* un sous-programme qui permet d'effectuer un ensemble d'instruction par simple appel de la fonction dans le corps du programme principal. Les fonctions permettent d'exécuter dans plusieurs parties du programme une série d'instructions, cela permet une simplicité du code et donc une taille de programme minimale. JavaScript contient des fonctions prédéfinies qui peuvent s'appliquer pour un ou plusieurs types d'objets spécifiques mais on peut définir ses propres fonctions.

La déclaration d'une fonction

Avant d'être utilisée, une fonction doit être définie car pour l'appeler dans le corps du programme il faut que le navigateur la connaisse, c'est-à-dire qu'il connaisse son nom, ses arguments et les instructions qu'elle contient. La définition d'une fonction s'appelle *"déclaration"*. La déclaration d'une fonction se fait grâce au mot clé function :

```
function Nom_De_La_Fonction(argument1, argument2, ...)
{liste d'instructions}
```

- le mot clé function est suivi du nom que l'on donne à la fonction
- le nom de la fonction suit les mêmes règles que les noms de variables :
  - le nom doit commencer par une lettre
  - un nom de fonction peut comporter des lettres, des chiffres et les caractères _ et & (espaces non autorisés !)
  - le nom de la fonction, comme celui des variables est sensible à la casse (différenciation entre les minuscules et majuscules)
- Les arguments sont facultatifs, mais s'il n'y a pas d'arguments, les parenthèses doivent rester présentes
- Ne pas oublier de refermer les accolades
- Le nombre d'accolades ouvertes (fonction, boucles et autres) doit être égal au nombre de parenthèses fermées !
- La même chose s'applique pour les parenthèses, les crochets ou les guillemets !

Une fonction ne s'exécutera pas tant que l'on ne fait pas appel à elle quelque part dans la page !

Appel de fonction

Pour exécuter une fonction, il suffit de faire appel à elle en écrivant son nom (une fois de plus en respectant la casse) suivie d'une parenthèse ouverte (éventuellement des arguments) puis d'une parenthèse fermée :

```
Nom_De_La_Fonction();
```

- le point virgule signifie la fin d'une instruction et permet au navigateur de distinguer les différents blocs d'instructions
- si des arguments ont été définis dans la déclaration de la fonction, il faudra les inclure lors de l'appel de la fonction (le même nombre d'arguments séparés par des virgules !)

Une fonction doit être déclarée avant d'être appelée. Le navigateur traitant la page de haut en bas, on déclare généralement les fonctions dans des balises script situées dans l'en-tête de la page, entre les balises <head> et </head>).
Grâce au gestionnaire d'évènement `onload` (à placer dans la balise body) il est possible d'exécuter une fonction au chargement de la page, comme par exemple l'initialisation des variables pour votre script, et/ou le test du navigateur pour savoir si celui-ci est apte à faire fonctionner le script.
Il s'utilise de la manière suivante :

```
<html> <head>
<script type=«text/javascript» >
<!--function Chargement() { alert('Bienvenue sur le site');}   //-->
</script> </head>
<body onload="Chargement();">...
</body> </html>
```

Les paramètres d'une fonction

Il est possible de passer des paramètres à une fonction, c'est-à-dire lui fournir une valeur ou le nom d'une variable afin que la fonction puisse effectuer des opérations sur ces paramètres ou bien grâce à ces paramètres.
Lorsque vous passez plusieurs paramètres à une fonction il faut les séparer par des virgules, aussi bien dans la déclaration que dans l'appel et il faudra bien passer le bon nombre de paramètres lors de l'appel.
Imaginons que l'on veuille créer une page Web qui affiche une boîte de dialogue qui affiche un texte différent selon le lien sur lequel on appuie.
La méthode de base consiste à faire une fonction pour chaque texte à afficher :

```
<html> <head>
<script type=«text/javascript» >
<!--function Affiche1() { alert('Texte1'); }
    function Affiche2() { alert('Texte2'); }   //--> </script> </head>
<body >
<a href="JavaScript :;" onclick="Affiche1();">Texte1</a>
<a href="JavaScript :;" onclick="Affiche2();">Texte2</a>
</body> </html>
```

Il existe toutefois une méthode qui consiste à créer une fonction qui a comme paramètre le texte à afficher :

```
<html> <head>
<script type=«text/javascript» >
<!--function Affiche(Texte) { alert(Texte); }   //--> </script> </head>
<body >
<a href="JavaScript :;" onclick="Affiche('Texte1');">Texte1</a>
<a href="JavaScript :;" onclick="Affiche('Texte2');">Texte2</a>
</body> </html>
```

Aucune différence visuelle mais vous n'avez plus qu'une seule fonction qui peut vous afficher n'importe quel texte...

Travailler sur des variables dans les fonctions

Lorsque vous manipulerez des variables dans des fonctions, il vous arrivera de constater que vous avez beau modifier la variable dans la fonction celle-ci retrouve sa valeur d'origine dès que l'on sort de la fonction...
Cela est du à la portée des variables, c'est-à-dire si elles ont été définies comme variables globales ou locales.

- Une variable déclarée implicitement (non précédée du mot var) sera globale , c'est-à-dire accessible après exécution de la fonction
- Une variable déclarée explicitement (précédée du mot var) dans la fonction sera locale , c'est-à-dire accessible uniquement dans la fonction, toute référence à cette variable hors de la fonction provoquera une erreur (variable inconnue)...

Le mot-clé this

Lorsque vous faîtes appel à une fonction à partir d'un objet, par exemple un formulaire, le mot clé `this` fait référence à l'objet en cours et vous évite d'avoir à définir l'objet en tapant *window.objet1.objet2...*
Pour passer l'objet en cours en paramètre d'une fonction, il suffit de taper `nom_de_la_fonction(this)` pour pouvoir manipuler cet objet à partir de la fonction.
Pour manipuler les propriétés de l'objet il suffira de taper `this.propriete` (où `propriete` représente bien sûr le nom de la propriété).

Les méthodes

Qu'appelle-t-on une méthode?

Une méthode est une fonction associée à un objet, c'est-à-dire une action que l'on peut faire exécuter à un objet. Les méthodes des objets du navigateur sont des fonctions définies à l'avance par html, on ne peut donc pas les modifier. Il est toutefois possible de créer une méthode personnelle pour un objet que l'on a créé soi-même. Prenons par exemple une page html, elle est composée d'un objet appelé *document*. L'objet *document* a par exemple la méthode `write()` qui lui est associée et qui permet de modifier le contenu de la page html en affichant du texte.
Une méthode s'appelle comme une propriété, c'est-à-dire de la manière suivante :

```
window.objet1.objet2.methode().
```

Dans le cas de la méthode `write()`, l'appel se fait comme suit :

```
window.document.write()
```

La méthode write()

La méthode `write()` de l'objet `document` permet de modifier de façon dynamique le contenu d'une page html, on peut l'utiliser de différentes façons :

- en passant directement le texte en paramètres : `document.write("bonjour");` qui aura pour effet de concaténer la chaîne 'bonjour' à l'endroit où est placé le script
- en passant le texte par l'intermédiaire d'une variable : `Chaine='bonjour'; document.write(Chaine);` qui aura pour effet de concaténer la chaîne 'bonjour' (contenue dans la variable *Chaine*)à l'endroit où est placé le script
- en utilisant les deux : `Chaine='bonjour'; document.write('je vous passe le' + Chaine);` qui aura pour effet de concaténer la chaîne 'bonjour' (contenue dans la variable *Chaine*) à la suite de la chaîne de caractère 'je vous passe le' dans la page html

Il est possible d'utiliser des balises html à l'intérieur même de la méthode `write()` :

```
document.write('<font color="#ff0000">Bonjour</font>');
```

La méthode writeln()

La méthode `writeln()` fonctionne exactement comme la méthode `write()` à la seule différence qu'elle ajoute un retour chariot à la fin de la chaîne. Or un retour chariot (en html) est ignoré par le navigateur (un retour à la ligne se fait avec la balise
). Cette méthode n'a donc un avantage que lorsqu'elle est utilisée entre les balises <PRE> et </PRE> qui formatent le texte comme dans un fichier texte (et donc qui prend en compte le caractère de retour à la ligne).

Objets du navigateur

Présentation des objets du navigateur

Lorsque vous ouvrez une page Web, le navigateur crée des objets prédéfinis correspondant à la page Web, à l'état du navigateur, et peuvent donner beaucoup d'informations qui vous seront utiles.
Les objets de base du navigateur sont les suivants :

- navigator : qui contient des informations sur le navigateur de celui qui visite la page
- window : c'est l'objet où s'affiche la page, il contient donc des propriétés concernant la fenêtre elle-même mais aussi tous les objets-enfants contenus dans celle-ci
- location : contient des informations relatives à l'adresse de la page à l'écran
- history : c'est l'historique, c'est-à-dire la liste de liens qui ont été visités précédemment
- document : il contient les propriétés sur le contenu du document (couleur d'arrière plan, titre, ...)

Ces objets sont largement dépendant du contenu de la page. En effet, mis à part des objets tels que *navigator* qui sont figés pour un utilisateur donné, le contenu des autres objets variera suivant le contenu de la page, car suivant la page les objets présents dans celles-ci (sous-objets des objets décrits précédemment) sont différents.

Les objets du navigateur sont classés hiérarchiquement

Les objets du navigateur sont classés dans une hiérarchie décrivant la page affichée à l'écran et permettant d'accéder à n'importe quel objet grâce à une désignation dépendant de la hiérarchie (du sommet on descend l'arborescence).
Dans cette hiérarchie, les descendants d'un objet sont des propriétés de ces objets mais peuvent aussi être des objets qui contiennent eux même des propriétés...
Voyons voir à quoi ressemble cette hiérarchie :

Niveau1	Niveau2	Niveau3	Commentaire
navigator			Informations sur le browser utilisé
window			Gestion de la fenêtre d'affichage
	parent, frames, self, top		Désignation de la sous-fenêtre
	location		Informations sur l'emplacement de la page
	history		Accès à l'historique (sites précédemment visités)
	document		Informations sur contenu de la fenêtre (éléments composant la page)
		images	Référence des images présentes dans la page
		forms	Référence des formulaires présents dans la page
		links	Référence des liens présents dans la page
		anchors	Référence des ancrages présents dans la page

Comment accéder à un objet?

Pour accéder à un objet du navigateur, il faut parcourir la hiérarchie du navigateur, en partant du sommet (l'objet `window`), puis en parcourant tous les maillons jusqu'à atteindre l'objet désiré. La syntaxe est `window.objet1.objet2.objet3.objet_vise` (trois objets intermédiaires `objet1 objet2 objet3` mais ce nombre peut varier de 0 à un très grand nombre d'objets, suivant l'imbrication de vos objets dans la page...).
Pour accéder modifier une propriété de l'objet visé il suffit de rajouter un point, puis le nom de la propriété.
Certaines propriétés sont modifiables, c'est-à-dire que dynamiquement on va pouvoir modifier un élément (du texte, une image, ...) ; d'autres par contre sont en lecture seule, c'est-à-dire qu'elles permettent uniquement de récupérer des informations (on ne peut pas les modifier...)

L'objet window

Les particularités de l'objet window

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

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

Les propriétés de l'objet window

propriété	description	lecture seule
defaultstatus	message qui s'affiche par défaut dans la barre d'état du navigateur	non, modifiable à tout moment
frames	tableau qui contient les cadres présents dans la fenêtre	Tous les éléments de *frames* sont en lecture seule
length	nombre de cadres (nombre d'éléments du tableau *frames*	Lecture seule
name	nom de la fenêtre dans laquelle on se trouve	Lecture seule
parent	fenêtre qui englobe celle dans laquelle on se trouve (s'il y en a une..)	Lecture seule, contient des propriétés
self	Synonyme de la fenêtre actuelle (redondance ou précision?)	Lecture seule, contient des propriétés
status	message temporaire qui s'affiche dans la barre d'état du navigateur suite à un événement	non, modifiable à tout moment, vous devez retournez la valeur *true* pour l'utiliser avec *onMouseOver*
top	fenêtre de plus haut niveau, celle qui contient tous les cadres (frames)	Lecture seule, contient des propriétés
window	fenêtre actuelle...	Lecture seule, contient des propriétés

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

Les méthodes de l'objet window

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

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

Les méthodes *alert()*, *confirm()* et *prompt()* sont des méthodes qui font apparaître une boîte de dialogue.

Les méthodes open(), et close()

Les méthodes *open()* et *close()* sont destinées à permettre l'ouverture et la fermeture de fenêtres.
La méthode *open()*admet un nombre important de paramètres. La méthode *open()* permet d'ouvrir une fenêtre :

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

URL désigne l'url de la page qui sera affichée dans la nouvelle fenêtre, c'est-à-dire l'emplacement physique de celle-ci.
Les options de la fenêtre sont les suivantes :

Option	Description
directory = *yes/no*	Affiche ou non les boutons de navigation
location = *yes/no*	Affiche ou non la barre d'adresse
menubar = *yes/no*	Affiche ou non la barre de menu (fichier, edition, ...)
resizable = *yes/no*	Définit si la taille de la fenêtre est modifiable ou non
scrollbars = *yes/no*	Affiche ou non les ascenceurs (barres de défilement)
status = *yes/no*	Affiche ou non la barre d'état
toolbar = *yes/no*	Affiche ou non la barre d'outils
width = largeur (en pixels)	Définit la largeur
height = hauteur (en pixels)	Définit la hauteur

Cette méthode est utilisable avec n'importe quel gestionnaire d'événement, directement dans le code à exécuter ou bien dans une fonction.

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

La méthode *close()* permet de fermer une fenêtre. Elle requiert le nom de la fenêtre comme argument.
On peut créer un bouton (image, hypertexte, ou bouton de formulaire) qui permettra de fermer une fenêtre.
Pour un lien hypertexte, le code sera :

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

Pour un bouton (image), le code sera :

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

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

```
<a href="JavaScript :;" onmouseover="self.close('nom_de_la_fenetre_');" ><img
src="nom_de_l_image"></a>
```

Boîtes de dialogue

Qu'est-ce qu'une boîte de dialogue?

Une boîte de dialogue est une fenêtre qui s'affiche au premier plan suite à un événement, et qui permet

- d'avertir l'utilisateur
- le confronter à un choix
- lui demander de compléter un champ pour récupérer une information

Ce type de boîte oblige une action de la part de l'utilisateur. Les boîtes de dialogues sont un moyen simple de débuguer (repérer les erreurs), en affichant à un point donné une fenêtre contenant la valeur d'une variable.
JavaScript en propose trois différentes dont l'utilisation se rapporte pour chacune à une de celles décrites ci-dessus. Ce sont des méthodes de l'objet *windo*w.

La méthode alert()

La méthode `alert()` permet d'afficher dans une boîte composée d'une fenêtre et d'un bouton *OK* le texte qu'on lui fournit en paramètre. Dès que cette boîte est affichée, l'utilisateur n'a d'autre alternative que de cliquer sur le bouton OK.
Son unique paramètre est une chaîne de caractère, on peut donc lui fournir directement cette chaîne de caractères entre guillemets, lui fournir une variable dont il affichera le contenu, ou bien mêler les deux en concaténant les chaînes grâce à l'opérateur +.

```
alert(nom_de_la_variable);
alert('Chaîne de caractères');
alert('Chaîne de caractères' + nom_de_la_variable);
```

La chaîne de caractère peut (et doit dans certains cas) contenir des caractères marqués d'un antislash (\). Par exemple, si vous voulez écrire :

```
Message d'alerte :
 Au feu ! !
```

Il faudra écrire le script suivant :

```
alert('Message d\'alerte \n Au feu ! !');
```

La méthode confirm()

La méthode `confirm()` est similaire à la méthode alert(), si ce n'est qu'elle permet un choix entre "OK" et "Annuler".
Lorsque l'utilisateur appuie sur "OK" la méthode renvoie la valeur `true`. Elle renvoie `false` dans le cas contraire...
Comme alert(), elle admet un seul paramètre : une chaîne de caractères...

```
confirm('Chaîne de caractères');
```

La méthode prompt()

La méthode `prompt()` est un peu plus évoluée que les deux précédentes puisqu'elle fournit un moyen simple de récupérer une information provenant de l'utilisateur, on parle alors de boîte d'invite.
La méthode `prompt()` requiert deux arguments :

- le texte d'invite
- la chaîne de caractères par défaut dans le champ de saisie

```
prompt('Posez ici votre question','chaîne par défaut');
```

Cette boîte d'invite retourne la valeur de la chaîne saisie par l'utilisateur, elle retourne la valeur `null` si aucun texte n'est saisi...

L'objet navigator

Les particularités de l'objet navigator

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

Les propriétés de l'objet navigator

Les propriétés de l'objet *navigator* sont peu nombreuses, elles permettent en fait de retourner des portions de l'information sur votre navigateur qui est en fait une chaîne de caractères.

Propriété	Description	Pour votre navigateur
navigator.appCodeName	retourne le code du navigateur.	Un navigateur a pour nom de code *Mozilla*
navigator.appName	retourne le nom du navigateur (la plupart du temps la marque).	Cette propriété est utile pour différencier les navigateurs de Netscape et de Microsoft
navigator.appVersion	retourne la version du navigateur. Cette propriété prend la forme suivante : Numéro de version(plateforme (système d'exploitation), nationalité)	Elle est utile pour connaître le système d'exploitation de l'utilisateur, mais surtout, associée avec la propriété *navigator.appName,* elle permet de connaître les fonctionnalités que supporte le navigateur de votre visiteur.
navigator.userAgent	retourne la chaîne de caractère qui comprend toutes les informations.	

Les propriétés ci-dessus offrent un moyen pratique de récupérer une partie de cette information (cette chaîne étant de longueur variable, il ne serait pas évident d'en récupérer une portion sans les autres propriétés...)

L'objet history

Les particularités de l'objet history

L'objet history est une propriété de l'objet document. Il contient l'historique du navigateur, c'est-à-dire l'ensemble des URL (adresses des pages) visitées par l'utilisateur. Ces adresses sont accessibles par le navigateur en cliquant sur les boutons suivants et précédent. L'objet *history* offre des objets et des méthodes qui permettent de naviguer dans cette liste d'adresse directement à partir de la page en cours.

Les propriétés et les méthodes de l'objet history

Les propriétés et les méthodes de l'objet *history* sont peu nombreuses, elles peuvent néanmoins être très utile pour aider à la navigation.

- **la propriété *length*** permet de connaître le nombre d'objets dans l'historique
- **la méthode *back*** permet d'aller à l'URL précédent dans l'historique
- **la méthode *forward*** permet d'aller à l'URL suivant dans l'historique
- **la méthode *go(variable)*** permet d'aller à un des URL de l'historique. Le paramètre *variable* est un nombre entier (positif ou négatif) qui détermine le nombre de pages relatif auquel se trouve l'URL désiré. Il est possible d'entrer une chaîne de caractères en paramètre, auquel cas le navigateur cherchera la page de l'historique la plus proche contenant cette chaîne

L'objet Date

Les particularités de l'objet Date

L'objet *Date* permet de travailler avec toutes les variables qui concernent les dates et la gestion du temps. Il s'agit d'un objet inclus de façon native dans JavaScript, et que l'on peut toujours utiliser.
La syntaxe pour créer un objet-date peut être une des suivantes :

```
Nom_de_l_objet = new Date()
```
cette syntaxe permet de stocker la date et l'heure actuelle.

```
Nom_de_l_objet = new Date("mois jour, année heures :minutes :secondes")
```
les paramètres sont une chaîne de caractères sous cette notation .

```
Nom_de_l_objet = new Date(year, month, day)
```
les paramètres sont trois entiers séparés par des virgules.
Les paramètres omis sont mis à zéro par défaut 3.

```
Nom_de_l_objet = new Date(year, month, day, hours, minutes, seconds)
```
les paramètres sont six entiers séparés par des virgules.
Les paramètres omis sont mis à zéro par défaut.

Les dates en JavaScript sont stockées de la même manière que dans le langage Java, c'est-à-dire qu'il s'agit du nombre de secondes depuis le 1er janvier 1970. Ainsi, toute date antérieure au 1er janvier 1970 fournira une valeur erronée. Si on veut manipuler des dates antérieures il s'agit de créer vous-même un objet date spécifique...

Les méthodes de l'objet Date

La date est stockée dans une variable sous la forme d'une chaîne qui contient le jour, le mois, l'année, l'heure, les minutes, et les secondes. Il est donc difficile d'accéder à un seul élément d'un objet *date*, étant donné que chacun des éléments peut avoir une taille variable. Les méthodes de l'objet *Date* fournissent un moyen simple d'accéder à un seul élément, ou bien de le modifier. Leur syntaxe est la suivante : `Objet_Date.Methode()`

Connaître la date

Les méthodes dont le nom commence par le radical *get* (qui signifie *récupérer*) permettent de récupérer une valeur :

Méthode	Description	Type de valeurs retournée
getDate()	valeur du jour du mois	entier (entre 1 et 31) qui correspond au jour du mois
getDay()	valeur du jour de la semaine pour la date spécifiée...	entier qui correspond au jour de la semaine : 0 : dimanche 1 : lundi …
getHours()	valeur de l'heure	entier (entre 0 et 23) qui correspond à l'heure
getMinutes()	valeur des minutes	entier (entre 0 et 59) qui correspond aux minutes
getSeconds()	valeur des secondes	entier (entre 0 et 59) qui correspond aux secondes
getMonth()	numéro du mois	entier (entre 0 et 11) qui correspond au mois : 0 : janvier 1 : février ...
getTime()	nombre de secondes depuis le 1er janvier 1970	entier. Cette méthode est très utile pour passer d'une date à une autre, soustraire ou ajouter deux dates, ...
getTimezoneOffset()	différence entre l'heure locale et l'heure GMT (Greenwich Meridian Time)	entier, il représente le nombre de minutes de décalage

Modifier le format de la date

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

Méthode	Description	Type de valeurs retournée
toGMTString()	Permet de convertir une date en une chaîne de caractères au format GMT	chaîne de caractère du type : Wed, 28 Jul 1999 15 :15 :20 GMT
toLocaleString()	Permet de convertir une date en une chaîne de caractères au format local	chaîne de caractère dont la syntaxe dépend du système, par exemple : 28/07/99 15 :15 :20

Modifier la date

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

Méthode	Description	Type de valeur en paramètre
setDate(X)	Permet de fixer la valeur du jour du mois	entier (entre 1 et 31) qui correspond au jour du mois
setDay(X)	Permet de fixer la valeur du jour de la semaine	entier qui correspond au jour de la semaine : 0 : dimanche 1 : lundi ...
setHour(X)	Permet de fixer la valeur de l'heure	entier (entre 0 et 23) qui correspond à l'heure
setMinutes(X)	Permet de fixer la valeur des minutes	entier (entre 0 et 59) qui correspond aux minutes

setMonth(X)	Permet de fixer le numéro du mois	entier (entre 0 et 11) qui correspond au mois : 0 : janvier 1 : février ...
setTime(X)	Permet d'assigner la date	entier représentant le nombre de secondes depuis le 1er janvier 1970

L'objet Math

L'objet Math est un objet qui a de nombreuses méthodes et propriétés permettant de manipuler des nombres et qui contient des fonctions mathématiques courantes

Les méthodes et propriétés standards de l'objet Math

Méthode	Description	Exemple
abs()	Retourne la valeur absolue d'un nombre	x = Math.abs(3.26) donne x = 3.26 x = Math.abs(-3.26) donne x = 3.26
ceil()	Retourne l'entier supérieur ou égal à la valeur donnée en paramètre	x = Math.ceil(6.01) donne x = 7 x = Math.ceil(3.99) donne x = 4
floor()	Retourne l'entier inférieur ou égal à la valeur donnée en paramètre	x = Math.floor(6.01) donne x = 6 x = Math.floor(3.99) donne x = 3
round()	Arrondit à l'entier le plus proche la valeur donnée en paramètre	x = Math.round(6.01) donne x = 6 x = Math.round(3.8) donne x = 4 x = Math.round(3.5) donne x = 4
max(Nombre1, Nombre2**)**	Retourne le plus grand des deux entiers donnés en paramètre	x = Math.max(6,7.25) donne x = 7.25 x = Math.max(-8.21,-3.65) donne x = -3.65 x = Math.max(5,5) donne x = 5
min(Nombre1, Nombre2**)**	Retourne le plus petit des deux entiers donnés en paramètre	x = Math.min(6,7.25) donne x = 6 x = Math.min(-8.21,-3.65) donne x = -8.21 x = Math.min(5,5) donne x = 5
pow(Valeur1, Valeur2**)**	Retourne le nombre *Valeur1* à la puissance *Valeur2*	x =Math.pow(3,3) donne x = 27 x = Math.pow(9,0.5) (racine carrée) donne x = 3
random()	Retourne un nombre aléatoire compris entre 0 et 1	x = Math.random() peut donner x = 0.6489534931546957
sqrt(Valeur**)**	Retourne la racine carrée du nombre passé en paramètre	x = Math.sqrt(9) donne x = 3

Logarithmes et exponentielle

Méthode	Description
Math. E	Propriété qui retourne le nombre d'Euler (environ 2.718)
Math.exp(valeur**)**	Retourne l'exponentielle de la valeur entrée en paramètre
Math.LN2	Retourne le logarithme népérien de 2
Math.LN10	Retourne le logarithme népérien de 10
Math.log(valeur**)**	Retourne le logarithme de la valeur entrée en paramètre
Math.LOG2E	Propriété qui retourne la valeur du logarithme du nombre d'Euler en base 2
Math.SQRT1_2	Propriété qui retourne la valeur de 1 sur racine de 2 (0.707)
Math.SQRT2	Racine de 2 (1.414)

Trigonométrie

Méthode	Description
Math.PI	Retourne la valeur du nombre PI, soit environ 3.1415927
Math.sin(valeur**)**	Retourne le sinus de la valeur entrée en paramètre (doit être donnée en radians)
Math.asin(valeur**)**	Retourne l'arcsinus de la valeur entrée en paramètre (doit être donnée en radians)
Math.cos(valeur**)**	Retourne le cosinus de la valeur entrée en paramètre (doit être donnée en radians)
Math.acos(valeur**)**	Retourne l'arccosinus de la valeur entrée en paramètre (doit être donnée en radians)
Math.tan(valeur**)**	Retourne la tangente de la valeur entrée en paramètre (doit être donnée en radians)
Math.atan(valeur**)**	Retourne l'arctangente de la valeur entrée en paramètre (doit être donnée en radians)