Apprendre Javascript

Tour d'horizon…

Qu'est-ce c'est ?

Javascript est un langage de **scripts** qui s'incorpore au langage **Html** de présentation des pages Web. Les scripts vont permettre de rendre une **page Web** plus **dynamique** :

- en animant certaines zones de la page,
- ou en faisant réagir certains éléments de la page en fonction de certains **événements** provoqués par l'utilisateur (= **client**), ou du navigateur utilisé, … (exemple : sur clic de la souris, changer la couleur du texte, ….)

Etant donné que Javascript peut être considéré comme une extension du langage HTML, la connaissance du HTML est indispensable avant d'aborder ce manuel. Malgré tout, un très léger rappel y sera fait.

• Logiciels nécessaires

- **Pour écrire** des scripts Javascript, un simple éditeur de texte tel le Bloc-notes de Windows est suffisant. Toutefois, de nombreux logiciels freeware ou non, en rendent l'écriture plus agréable (aide intégrée, colorisation des mots-clés, …)
Exemples à ce jour : MSE (Microsoft), 1st page 2000 (EvrSoft), DreamWeaver (Macromedia), …

- **Pour exécuter** les scripts, il faut un navigateur tel que Internet Explorer (IE), Netscape Navigator (NN), Opéra, …

• Historique

JavaScript (initialement appelé LiveScript) a été développé par Netscape puis a été repris par la firme Sun (qui a aussi développé Java).
Les scripts sont des instructions (des lignes de **code**) interprétées par les navigateurs Netscape et Internet Explorer. Javascript n'a cessé d'évoluer avec les versions des navigateurs. Le tableau ci-dessous récapitule quelles versions de Javascript sont reconnues par les 2 navigateurs les plus utilisés à ce jour. Mais n'oublions pas que ce ne sont pas les seuls (Opéra, …).

Navigateurs	Versions Javascript supportées
Netscape 2.0 (NN2) Internet Explorer 3.0 (IE3)	Javascript (baptisé à posteriori 1.0)
Netscape 3.0 (NN3)	Javascript 1.1
Netscape 4.0 (Communicator) (NN4) Internet Explorer 4.0 (IE4)	Javascript 1.2
Netscape 6.0 (NN6)	Javascript 1.5

Nous verrons qu'il existe une fonction Javascript qui permet de tester le type et la version du navigateur utilisée. Elle permettra d'aiguiller l'utilisateur vers une page ou une autre.

- ## Java, Javascript et Jscript

Surtout ne pas confondre Java et Javascript/Jscript bien qu'ils soient tous utilisés pour créer des pages Web évoluées !
Javascript et **Jscript** sont interprétés et exécutés par le navigateur de l'internaute. JavaScript, nous l'avons vu, a été développé par Netscape. Microsoft, quant à lui, a repris Javascript pour le nommer Jscript dans IE. Les différences de langage entre Javascript et Jscript s'accentuant au cours des différentes versions, en 1998, Netscape a confié à l'ECMA (European Computer Manufacturers Association) le soin de créer un standard : l'ECMA-262. Toutefois les éditeurs continuent d'ajouter de nouvelles fonctionnalités à leur navigateur. De ce fait, le standard n'est pas totalement implémenté et des différences encore notables sont encore à signaler. Et même… en supposant que le standard soit respecté, l'ECMA n'a pas imposé le modèle des objets du document (DOM : voir page 44).
Heureusement, la compatibilité est de bon niveau.

Le tableau ci-dessous donne un bref aperçu des différences entre les 2 langages de script et Java, véritable langage de programmation :

Javascript / Jscript	Java
Adapté uniquement à la gestion des pages Web	Gère les pages Web mais aussi permet de créer des applications totalement autonomes et hors Web.
Instructions intégrées dans la page Html : un simple éditeur de textes est nécessaire pour les écrire.	Module (« applet ») appelé dans la page Html : il nécessite un éditeur de texte ET un compilateur.
Instructions interprétées linéairement par le navigateur du client (=l'internaute) au moment de l'exécution : une erreur de syntaxe bloque l'affichage de la page Html. Donc : il est indispensable d'avoir bien testé son script, si possible sur différents navigateurs et différentes versions de navigateurs, avant de le mettre à disposition sur une page publique.	Code **source compilé** avant son exécution : tant que le module n'est pas syntaxiquement correct, il ne peut être appelé dans la page. Aux yeux de l'utilisateur, la page Web s'affichera correctement.
Codes simples → applications relativement simples	Langage de programmation beaucoup plus complexe mais plus performant (accès aux bases de données, …). Tout est théoriquement possible.
Permet d'accéder aux objets du navigateur	N'accède pas aux objets du navigateur
Aucune confidentialité du code : le code source est visible et copiable par tout utilisateur dans le navigateur, mais les droits d'auteur restent applicables.	Confidentialité et sécurité du code après compilation. Le code étant illisible et dans un fichier séparé, seul le concepteur peut modifier le code initial.
Permet d'accéder aux clients Web (navigateur)	Permet d'accéder aux serveurs Web (bases de données par exemple)
L'intégration de scripts ne pénalise pas significativement les temps de téléchargement de la page Web	L'intégration d'applets ralentit le téléchargement de la page.

- ## Syntaxe et Conseils

- Javascript ressemble aux langages C et C++ : à savoir
  - ; à la fin de chaque ligne d'instructions. Cela deviendra vite une habitude bien que Javascript ne l'impose pas.

- **{…}** pour encadrer un bloc d'instructions
- les opérateurs de comparaison et d'affectation sont les mêmes (**&&**, **||**, **==**, …)
- les symboles des commentaires **//** et **/* … */**
- toute une série de mots réservés qui correspondent à des mots-clés du langage.
- …

Toutefois, Javascript est sensible à la casse (caractères minuscules/majuscules) c'est-à-dire que la variable ***NBRE*** n'est pas la même que la variable ***nbre*** ou ***Nbre***

- Difficultés

La difficulté avec Javascript est d'écrire des scripts qui puissent être interprétés et exécutés par n'importe quel navigateur et sur n'importe quelle plate-forme (PC, Mac, Unix, Linux).

JavaScript et HTML

- Quelques rappels de HTML

Une page HTML est constitué de **balises** (=tags) qui correspondent à des instructions de mise en forme de la page HTML. Chaque balise est encadrée par les symboles **<** et **>**. Chaque mise en forme est présentée par une balise de début **<balise>** et une balise de fin **</balise>**.
Exemple : <B> ce texte sera en caractères gras </B> <I> ce texte sera en italiques </I>

Certaine balises sont obligatoires dans toute page Web. La plus petite page possible est la suivante : celle-ci affiche « **Coucou**».

```
<HTML>
<HEAD>  </HEAD>
<BODY> Coucou </BODY>
</HTML>
```

- Notion d'objets

JavaScript est un langage qui manipule les objets. Voici quelques exemples d'objets :
- les balises Html comme les cadres (<Frame>), les formulaires (<Form>) et les champs de texte (<Textarea>, <Input> ; …), les images (<Img>), les applets Java (<Applet>), …
- les spécificités du navigateur (son nom, sa version, …),
- les plug-ins installés sur le poste client (Acrobat, Real Audio, …)
- la date et l'heure du poste-client,
- …

L'ensemble de ces objets constituent le modèle des objets du document : le DOM (Document Object Model). Nous le détaillerons plus loin car sans le connaître, nous allons pouvoir écrire notre premier script.

- Dynamic HTML (DHTML)

Javascript associé aux **événements** et aux **feuilles de style** et aux **calques** permet de dynamiser les objets des pages Web ce qui signifie que des modifications peuvent se faire une fois que la page a fini de se charger, ce que ne permet pas le HTML classique.

Le DHTML est intéressant, mais, malheureusement, à l'heure actuelle, il n'existe pas de norme officielle et suivie par les navigateurs. Chaque navigateur gère le DHTML à sa façon. Ainsi, un script écrit pour un navigateur ne fonctionnera pas sur un autre sans un travail d'adaptation.

IE4 et IE5	compatibilité ascendante
NN4	incompatible avec IE et NN6
NN6 et IE5	compatible avec norme du W3C
NN6	incompatible avec ses versions antérieures

Ainsi, si on tient compte des navigateurs les plus courants à l'heure actuelle, 3 navigateurs seront nécessaires pour tester la bonne exécution des pages (IE4, NN4, IE5 ou NN6).

Un premier script

Ces lignes nous permettront de voir comment on intègre un script dans une page HTML. Nous allons créer un exemple très simple. Dans l'exemple précédent, le message « Coucou » s'affichait dans la page HTML. Dans cet exemple, c'est le clic sur un bouton de la page HTML qui va afficher « Coucou » dans une boîte de message.

- Ecriture de la page HTML

La première chose à faire consiste à créer le bouton en HTML . Ce qui donne :

```
<html>
<head>
</head>
<body bgcolor="#FFFFFF">
   <input type= "Button" name="bt_cliquer" value="Cliquer ici">
</body>
</html>
```

- Déclaration d'un script

Il reste à définir que, sous l'action d'un clic, une boîte dialogue va s'ouvrir. La page Html ci-dessus va être modifiée ainsi :

```
<input type= "Button" name="bt_cliquer" value="Cliquer ici"
onclick="ouvrir_fenetre();">
```

La fonction Javascript appelée **ouvrir_fenetre()** est définie ainsi :

```
<SCRIPT Language="Javascript">
<!--
function ouvrir_fenetre()
{
    window.alert("Coucou");
}
-->
</SCRIPT>
```

- ## Script commenté

Code	Commentaires
<SCRIPT Language="Javascript">	Détermine le langage utilisé. Ce pourrait être "VBScript"
<!-- cache le script aux anciens navigateurs qui sinon vont l'afficher	Les balises Html <!-- et --> encadre du texte qui est dit "commentaire", non interprété par le navigateur.
function ouvrir_fenetre()	La fonction définie s'appelle "ouvrir_fenetre" et ne nécessite aucune valeur préalable appelée **paramètre** "()". Toute description de fonction javascript doit commencer par "function"
{	Les symboles { et } définissent des blocs d'instructions qui seront exécutés séquentiellement.
window.alert("Coucou");	Appel de la méthode "alert" de l'objet window (voir "les méthodes" des objets). Toute instruction javascript doit terminée par un ";" (sauf les instructions itératives et de boucles.)
}	Fin de bloc d'instructions
-->	Fin du bloc de commentaire Html
</SCRIPT>	Balise de fin de script

A l'exécution, un clic sur le bouton **Cliquer ici** fait apparaître la fenêtre suivante :

Remarques :

- Etant donné que la fonction ne contient qu'un seule instruction, on n'aurait pu se passer d'écrire la fonction "**ouvrir_fenetre()**" et écrire beaucoup plus simplement :

```
<input type= "Button" name="bt_cliquer" value="Cliquer ici"
onclick="javascript:window.alert("Coucou");">
```

- Javascript est généralement utilisé côté client (avant que la page ne soit envoyée au serveur) mais il peut également fonctionner côté serveur. Il suffit d'écrire :
  <SCRIPT Language="Javascript" runat=server>

- ## Où placer les scripts ?

- C'est le couple de balises <Script> … </Script> qui permet de repérer l'existence d'un script dans une page Html, mais … où le placer ?
Il peut être placé n'importe où dans la page mais si il est placé dans l'entête <HEAD> … </HEAD>, alors il est exécuté avant même que le navigateur ait commencé à afficher le contenu. Cela permet de manipuler le corps du document.
- Lors de l'appel à une fonction dans une page, il est impératif que cette fonction ait été déclarée avant d'être utilisée. La déclaration dans l'entête de la page répond à cette exigence.

- ## Ça ne marche pas !

L'interpréteur de script doit être activé dans les navigateurs NN et IE. Pour le vérifier :

Dans NN	**Editions / Préférences**. Dans la boîte de dialogue, cliquer sur "Avancées". La case placée devant "Activer Javascript" doit être cochée.
Dans IE	**Outils / Options Internet** puis onglet "**Sécurité**" Cliquer sur l'icône "Internet" (=globe terrestre) puis "Personnaliser le niveau". Dans la boîte de dialogue, dérouler la liste jusqu'à la rubrique "Script". Le radio-bouton placé devant "Active Scripting/Activer" doit être coché.

- ## Les commentaires

Nous avons vu comment se présente une fonction Javascript. Celle-ci est très simple, mais elle peut très vite être plus compliquée. Dans ce cas, il est important, voire indispensable, de commenter les instructions. Pourquoi ?
- lorsqu'une fonction a été écrite, il est difficile d'en comprendre certaines subtilités après quelques mois. Un commentaire permet de donner des pistes.
- pour permettre une prise en main et une maintenance rapides par quelqu'un n'a pas écrit la fonction.

Les commentaires devraient être présents dans les pages Html ET les fonctions Javascript. Leur syntaxe est différente.

✓ Les commentaires en Html

<!-- commentaire début commentaire suite commentaire fin **-->**	Sur plusieurs lignes, tout ce qui est compris entre <!-- et --> n'est pas interprété par le navigateur. En Html, les commentaires évitent que certains anciens navigateurs affichent les fonctions qui leur sont inconnues comme étant de l'Html. Dans ce cas, ils les ignorent.

✓ Les commentaires en Javascript

Dans des fonctions Javascript, il existe 2 types de symboles pour désigner du commentaire.

// appel de la méthode "alert"	sur une même ligne, tout ce qui suit les signes // n'est pas analysé par le compilateur de Javascript

/* commentaire début commentaire suite commentaire fin ***/**	Sur plusieurs lignes, tout ce qui est compris entre /* et */ n'est pas analysé par le compilateur. Ceci permet de déterminer des blocs de commentaires.

- ## Utiliser les mêmes scripts dans différentes pages Html

Lorsqu'un même script doit être utilisé dans des pages Html différentes, afin de faciliter sa maintenance, il est conseillé d'écrire ce script dans un fichier texte séparé (suffixé par exemple .JS pour le Javascript).
L'attribut SRC ajouté à la balise <Script> indiquera le nom du fichier qui contient le script. (Les autres paramètres sont ignorés). La balise </SCRIPT> est alors absente.
Exemple :

```
<SCRIPT SRC="MesScripts.js">
```

- En savoir un peu plus …

Certains développeurs lorsqu'il utilisent des spécificités d'une version précisent la version Javascript utilisée (exemple : <Script Language="Javascript1.2>). Mais le plus simple est d'utiliser le paramètre "Javascript" et de tester sur différentes versions de navigateurs.

Fonctions, instructions, variables et opérateurs

Dans l'exemple précédent, nous avons écrit une fonction **ouvrir_fenetre()** qui contient une seule instruction. Nous allons maintenant décrire les composants qui permettent l'élaboration de fonctions plus complexes.

- Rappel

Javascript étant sensible à la casse des caractères, il est important de respecter la position des majuscules dans les instructions qui vont suivre.

- Fonctions

Les fonctions peuvent être assimilées à des boîtes qui regroupent un ensemble d'instructions réutilisables simplement et qui permettent d'obtenir un résultat précis.

- Une fonction peut (mais elle n'est pas obligée) accepter des valeurs (appelées "arguments" ou "**paramètres**"). Si il y en a plusieurs, elles sont séparées par des virgules.
- Elle peut (mais elle n'est pas obligée) renvoyer **une** (et une seule !) valeur issue d'un calcul (instruction **return**)
- Une fonction fait quelque chose d'utile.

✓ Sa syntaxe

- Le nom de la fonction est précédé du mot-clé **function**
- Les instructions qui composent la fonction sont encadrées de **{** et **}**

✓ Exemple d'une fonction avec un argument

```
function calcul_surface_cercle (rayon)
{
  /* cette fonction calcule la surface d'un cercle à partir du
     rayon passé en paramètre */
   var surf= Math.PI * rayon * rayon;
   return surf;      // ou return (surf);
}
```

✓ Exemple d'une fonction à 2 arguments

```
function calcul_surface_rectangle (longueur, largeur)
{
  /* cette fonction calcule la surface d'un rectangle à partir des
     longueur et largeur passées en paramètre  */
   return longueur*largeur;      // calcule la surface et renvoie
                                 // la valeur
}
```

✓ Fonctions préexistantes dans Javascript

Fonctions	Rôle	Exemple
parseInt()	Extrait une valeur **numérique entière** à partir d'une chaîne de caractères	var x= parseInt("234.55 €") // x=234
parseFloat()	Extrait une valeur **numérique décimale** à partir d'une chaîne de caractères	var x= parseInt("234.55 €") // x=234.55
isNaN()	(depuis Javascript 1.1) Teste si une valeur est numérique ou non. Elle retourne la valeur "true" ou "false".	var reponse=isNaN("234.55 €")) // reponse = false
isFinite()	(depuis Javascript 1.3) indique si l'argument est une valeur numérique finie	if (isFinite(x)) return(x); else return (Infinity);
escape()	code un caractère ou une chaîne pour l'insérer dans une URL	str = escape("&"); // str="%26" // codage ASCII
unescape()	retourne la chaîne ASCII de la valeur de codage indiquée (en hexadécimal)	str=unescape("%28"); // str="("

eval()	retourne le résultat numérique d'une paramètre. C'est une fonction très puissante car elle permet à la fois, d'examiner, d'interpréter et d'exécuter une chaîne Javascript comme s'il s'agissait d'un script	var operateur ="*", val1= 10;
		var valeur = "val1 = val1" + operateur + "5";
		var x= eval(valeur); // x=50

Number()	convertit un objet en valeur numérique	var nbre = Number("01/02/94")
String()	convertit un objet en chaîne	maDate=new Date(940000000000); strDate=String(maDate); // strDate devient "Ven 15 Oct 16:06:40 UTC+01001999"

- Les instructions

Une fonction contient une série d'instructions qui permettent de réaliser des boucles (structures itératives), des conditions, des affectations de valeurs, de la gestion des chaînes de caractères, des conversions, … Les instructions les plus importantes sont détaillées ci-après (p 24 et suivantes).

- Les variables et les types de données

Elles correspondent à des zones de mémoire "étiquetées" par un nom, dans lesquelles seront enregistrées les valeurs à traiter. Elles sont créées à l'aide du mot-clé **var** et reçoivent une valeur grâce au symbole d'affectation **=** . En Javascript, il existe 7 types de données déterminés par l'affectation qui en est faite :

Déclaration en Javascript	Type
var chaine1 = "ENITA Bordeaux" ; var chaine2 = "Réunion Salle \"Pomerol\" " ; var chaine3= chaine2 ;	**Chaîne de caractères** certains caractères spéciaux doivent être précédés de \ (voir Annexe : Caractères spéciaux)
var prix = 10.95 ; var indice = -1 ; var i ;	**Numérique** décimal Numérique entier Numérique indéfini (type undefined)
var indicateur = false ; var option = true ;	**Booléen** (false ou true)

Déclaration en Javascript	Type
var dept_enitab = new Array ("des", "dpa", "dbe") ;	**Tableau** de données. Ce qui donnera : dept_enitab [0] = "des" ; dept_enitab [1] = "dpa" ;
var tab1 = new Array (2) ; tab1[0] = new Array (3) ; tab1[1] = new Array (3) ; tab1[0] [0] = "dpa" ; tab1[1] [0] = "des" ; …	**Tableau multidimensionnel :** on crée tab1[0], tab1[1] (2 éléments). Puis on définit tab1[x] comme étant à nouveau des tableaux à 3 éléments. On affecte les valeurs.
var aujourdhui = new Date();	**Date** – retourne la date et l'heure en cours
var maVariable = null ;	**Null** – aucune donnée n'est présente (absence de données). Signification différente de 0 (zéro) ou vide ("").
var pi = Math.PI ; var racine_carree = Math.sqrt (121) ;	**Mathématique**. Les variables reçoivent des valeurs de fonctions mathématiques prédéfinies.

✓ Conseils et remarques

- Eviter tout signe diacritique dans les noms de variables (caractères accentués, cédille, …)
- Pas d'espace dans les noms de variable
- Javascript est sensible à la casse utilisée (la variable "toto" est différente de la variable "Toto").
- Le mot-clé **var** est facultatif mais fortement conseillé pour éviter tout problème de confusion.
- = est le symbole d'affectation ; == est le symbole de comparaison (voir "opérateurs" page 46).

• Portée des variables

✓ Variable locale

Une variable déclarée dans une fonction par le mot-clé **var** aura une portée limitée à cette seule fonction. On ne pourra donc pas utiliser sa valeur ailleurs dans le script. C'est une **variable locale** à la fonction.

Exemple : var ecole ="ENITA Bordeaux"; // n'est connue que la fonction dans laquelle elle se
// trouve

✓ Variable globale

- Si la variable est déclarée **sans** utiliser le mot **var**, sa portée sera **globale** (une fois que la fonction aura été exécutée).

Exemple : ecole="ENITA Bordeaux" ; // est connue de toutes les fonctions. Elle pourra être
// utilisée dans d'autres fonctions

- De même, une variable déclarée tout au début du script, **en dehors et avant toute fonction**, est toujours globale, qu'elle soit déclarée **avec** ou **sans var**.

Exemple :

```
<SCRIPT LANGUAGE="javascript"
ecole="ENITA Bordeaux"   // est connue de toutes les fonctions
function signer()
{
    var titre="Directeur";
    titre = titre + ecole;
```

```
}
```

Conseil : Pour la facilité de gestion des variables, on ne peut que conseiller de les déclarer en début de script (comme dans la plupart des langages de programmation).

- ## Les opérateurs

Les opérateurs vont permettre de manipuler les variables (comparaison, affectation, calculs, …). Il en existe de différentes catégories. Tout comme en mathématique, certains signes ont des priorités de calcul par rapport à d'autres, les opérateurs Javascript ont des priorités décrites en annexe p 48.

✓ Les opérateurs de calcul + - * /

Ils servent aux calculs : * (multiplier), + (additionner), - (soustraire), / (diviser), = (affecter) , % (prendre le reste de la division = modulo).

Exemples : a * 5.2 ; b / (c + 4) 10 % 2

✓ Les opérateurs de comparaison ==, <, …

Ils sont utilisés dans les instructions de conditions (if …. {. .. }).

Exemples : if (x == 0) signifie si x = 0
if (x != 0) signifie si x différent de 0

Les autres opérateurs sont listés en annexe page 46.

✓ Les opérateurs associatifs += -= *= /=

Ils permettent à la fois le calcul et l'affectation de la valeur calculée à une variable.

Exemples : x + = y équivalent de x = x + y

✓ Les opérateurs logiques && et ||

Ils servent à comparer 2 expressions.

Exemples : if (x > 10) && (y < b) { … } si x > 10 et y < b alors …
if (x < 0) || (z == 1) { … } si x < 0 ou z = 1 alors …

✓ Les opérateurs unaires + et -

L'opérateur – est courant puisqu'il sert à inverser la valeur donnée :

Exemples : -x si x = 2 alors x = -2
-y si y = -4 alors y = 4

✓ Les opérateurs d'incrémentation ++ et --

Ils servent à augmenter (ou diminuer) de 1 la valeur d'une variable. Ils sont très utilisés dans les structures de boucles.

Exemples : i++ équivalent de i = i + 1
i-- équivalent de i = i - 1

✓ L'opérateur d'affectation =

Il sert à donner une valeur à une variable.

Exemples : x = 10 x prend la valeur 10
y = a = w = 3 y, a et w sont initialisés à 3

✓ L'opérateur de concaténation de chaînes de caractères +

Il sert à associer des chaînes de caractères pour en former une nouvelle.
Exemples : departement = "DES" ;
chaine1 = "- ENITA Bordeaux - " + departement + " – " ;

Remarque : Le signe + sert aussi à additionner des nombres. Javascript reconnaît le type de variable utilisé. Toute constante encadrée de " ou de ' et toute donnée contenant un caractère autre qu'un caractère admis dans les chiffres (chiffres, virgule, point, signe unaire, …) est considérée comme une chaîne de caractères.

✓ L'opérateur de données binaires

Ces opérateurs sont rarement utilisés. Ils sont décrits en annexe page 46.

✓ Autres opérateurs

Il existe d'autres opérateurs que nous détaillerons plus loin :
new, **delete**, **void**, **typeof**, **in**, **instanceof** et **this**
Les 3 derniers s'appliquent aux objets.

Les événements

- Généralités

C'est la notion d'événements gérés par du Javascript qui rendent les pages Html animées et attractives. En effet selon l'action de l'internaute, des actions seront déclenchées et la page va s'animer d'une manière ou d'une autre. En bref, c'est la clé de l'interactivité autrement appelée DHTML (Dynamic HTML).

En Html classique, un seul événement est géré : c'est le clic de la souris sur un lien (balise …). Javascript va permettre d'en gérer beaucoup d'autres.

Les événements sont associés aux fonctions, aux méthodes et aux formulaires.

- Les événements et les gestionnaires d'événements

Les événements correspondent aux actions réalisées par l'utilisateur. Les fonctions à réaliser selon les événements seront déclenchées par le gestionnaire d'événements.

Description	Evénements	Gestionnaire d'événement
Quand l'utilisateur clique sur un bouton, un lien ou tout autre élément…	Click	onClick="fonction()"
Quand la page a fini d'être chargée par le navigateur…	Load	onLoad="fonction()"
Quand l'utilisateur quitte la page…	Unload	onUnload="fonction()"
Quand le pointeur de la souris survole (sans clic) un lien ou tout autre élément…	MouseOver	onMouseOver="fonction()"
Quand le pointeur de la souris quitte un lien ou tout autre élément… A partir de Javascript 1.1 (donc pas sous IE 3.0 et Netscape 2).	MouseOut	onMouseOut="fonction()"

Quand un élément d'un formulaire devient la zone d'entrée active. On dit qu'il reçoit le focus.	Focus	onFocus="fonction()"
Quand un élément d'un formulaire a perdu le focus (n'est plus l'élément actif). En général, cela se produit quand l'utilisateur clique en dehors de l'élément actif.	Blur	onBlur="fonction()"
Quand la valeur d'un champ de formulaire est modifiée.	Change	onChange="fonction()"
Quand l'utilisateur sélectionne tout ou partie d'une valeur d'un champ dans un élément de formulaire.	Select	onSelect="fonction()"
Quand l'utilisateur clique sur le bouton "Submit" pour envoyer un formulaire au serveur...	Submit	onSubmit="fonction()"

Exemples : "Click" :

```
onClick = "alert ('Vous avez cliqué sur le bouton')"
```

au clic de l'utilisateur, une boite d'alerte s'ouvre avec le message indiqué.

Exemples : "Load - Unload"

Ces 2 événements sont en général appelés dans les balises <BODY> ou <FRAMESET>.

```
<BODY onLoad = "alert ('Bienvenue !')" onUnload="alert ('A bientôt !')">
```

Quand la page est chargée, l'internaute verra une boîte d'alerte "Bienvenue !" et quand il la quittera, il verra "A bientôt !".

Exemples : "mouseOver - mouseOut" :

```
<IMG SRC="logo.jpg" onMouseover="alert ('Le curseur de la souris est passée sur le logo') ">
```

Cet événement est pratique pour afficher des explications soit dans la barre de statut ou pour afficher un texte qui correspond à la zone survolée, …
L'événement **onmouseOut** est généralement associé à un **onmouseOver**. Il se produit lorsque le pointeur quitte la zone "sensible" (lien ou image).

On peut aussi coder en faisant intervenir la fonction sur un lien hypertexte :

```
<A HREF="#" onMouseover="alert ('Le curseur de la souris est passée sur le logo') > lien </A>
```

Dans ce cas, l'internaute peut cliquer sur le lien. Pour gérer correctement ce "clic", il est conseillé de réaliser un lien sur **"#"** qui aura pour seule conséquence, de réafficher la page en cours.
Autre écriture quasiment équivalente à la précédente :

```
<A HREF="javascript:void(0)" onMouseover="alert ('Le curseur de la souris est passée sur le logo') > lien </A>
```

La différence vient de l'appel à une fonction Javascript qui ne fait …. rien. En effet, l'opérateur **void** avec le paramètre 0 indique qu'aucune fonction n'est exécutée quand l'internaute clique sur l'image.

C'est grâce à ces 2 événements **mouseOver** et **mouseOut** que l'on peut gérer l'inversion d'images (une image quand le curseur de la souris page sur le lien, une autre image quand il quitte ce lien).

Exemples : "focus – blur - change" :

```
<INPUT type="text" onFocus="calcul1()" onChange="verif ()" onBlur="calcul2 ()" >
```

La fonction calcul1() est appelée dès que le curseur se trouve dans cette zone de texte. Si la valeur de la zone a été modifiée, la fonction verif() est appelée, puis quand le curseur a quitté cette zone, c'est la fonction calcul2() qui est exécutée.

Exemples : "select" :

```
<INPUT type="text" onSelect="copie();" >
```

Si tout ou partie de la valeur est sélectionnée (mise en surbrillance) dans une zone Text ou Textarea, la fonction copie() est appelée.

Les entrées / sorties

- **Les saisies de données** (entrée) vont être réalisées par l'intermédiaire des formulaires (voir objet **form**)

- **Les édition des résultats** (sorties) vont s'inscrire dans de nouvelles fenêtres (voir objet **window**) ou dans le document en cours (voir objet **document**).

Structures algorithmiques

Instructions de condition : Si … alors … Si … alors … sinon …

Si… Alors … [Sinon …]	Exemple

if (condition)
{
instructions;
… ;
}
	if (x > 1) // si x > 1
	{
	if (y == "ha") // si y = 0
	{
	alert ("la surface est de " + x + " hectares") ;
	// affichage du résultat
	}
	}

else
{
instructions ;
}
	// Instructions facultatives

Remarque : Il est tout à fait possible de faire des conditions imbriquées. Pour des facilités de lecture du script, il est conseillé de décaler vers la droite les blocs **if** internes comme dans l'exemple ci-dessus.

Instructions de boucle : Pour …

Pour …	Exemple : somme des 10 premiers nombres entiers

for (expression_initiale ; condition_de_sortie ;
expression de progression)
{
instructions;
}
	for (i=1 ; i <= 10 ; i++) // pour i variant de 1 à 10
	// (inclus) par pas de 1
	{
	somme = somme + i;
	}

Instructions de boucle : Tant que …

Tant que …	Exemple : factorielle 10

while (condition)
{
  instructions;
}
	var n = 10 ;
	var somme = 1 ;
	while (n > 0) // tant que n > 0 (inclus) par pas de 1
	{
	  somme = somme * n;
	  n-- ;
	}

Remarque : Dans une boucle "Tant que …", les instructions situées dans la boucle peuvent ne **jamais être exécutées** si la condition n'est pas remplie.

Instructions de boucle : Faire … tant que …

Faire … tant que …	Exemple : liste des options de 3ème année

do
{
  instructions;
}
while (condition)
	var texte=""; // initialisation de la chaîne
	var n=0;
	var option_3a = new Array ("agrotic", "forêt", "oeno", "fin") ; // création tableau des options de 3ème année
	
	do
	{
	  texte = texte + option_3a [n++] + "\n"; //concatène
	  // les options et ajoute un retour ligne \n
	}
	while (option_3a[n] != "fin") // tant que différent
	  // du mot "fin"
	alert (texte);

Remarque : Dans une boucle "Faire … tant que …", les instructions situées dans la boucle sont **exécutées au moins une fois**. même si la condition n'est pas remplie.

Instructions de branchements multiples : si … si … si …

si … si … si …	Exemple : liste des options de 3ème année

switch (expression)
{
  case valeur1 : instructions;
  break;
  case valeur2 : instructions;
  break;}
default : instructions;
}
	switch (choix_menu)
	{
	  case 1 :
	  case 2 : appel_fct_menu1_2(); // si menu=1 ou 2
	  break;
	  case 3 : appel_fct_menu3();
	  break;
	  default : alert ("vous n'avez pas choisi de menu");
	}

Remarques :

- L'expression peut être une simple variable (voir ce chapitre) ou une expression à analyser.
- Plusieurs "case" peuvent être empilés. Dans l'exemple ci-dessus, la fonction appel_fct_menu__2() est appelée si choix_menu vaut 1 ou 2.
- Chaque "case" est terminé par une instruction break, pour éviter que le compilateur n'exécute la suite des "case"
- L'instruction "default" est facultative. Elle sert à traiter tous les cas non précisés dans la liste des "case".
- Dans le cas d'une variable alphanumérique, on écrirait : case "L"

Rupture de structure, ou le mauvais usage de break et continue

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

L'instruction **continue** permet, quant à elle, de sauter à l'occurrence suivante de la boucle **for** ou **while**. Elle permet donc de sauter un bloc d'instructions et de continuer ensuite le bouclage (sans sortir de celui-ci comme le fait **break**).

Remarque :
Ces instructions **bafouent les règles élémentaires** de la programmation structurée. Leur usage ne s'impose que dans des cas très particuliers qui sortent du cadre de l'objectif de ce cours. **Nous en proscrirons donc l'usage.**

Récursivité

Comme la plupart des langages, Javascript permet d'écrire des fonctions récursives, c'est-à-dire des fonctions qui s'appellent elles-mêmes ou qui appellent la fonction de départ. Le passage des paramètres se faisant par recopie des valeurs, les valeurs des variables ne sont pas écrasées.

Attention, ce genre de structure itérative « déguisée » est réservée aux programmeurs avertis.

Exemple : somme des n premiers entiers

```
// si n vaut 1 alors la somme du premier entier vaut 1
// sinon elle vaut la somme de n + la somme des n- 1 premiers
entiers
function somme_n_entiers( n)
{
    return n == 1 ? 1 : n + somme_n_entiers ( n - 1);
}

{
…
   somme = somme_n_entiers (10);    // la somme des
                                    // 10 premiers entiers vaut "somme"
}
```

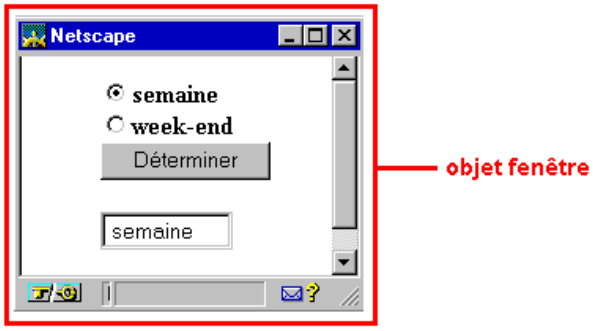
La théorie des objets et le DOM

Nous avons vu ce que peuvent être des objets (voir page 4).

La hiérarchie des objets

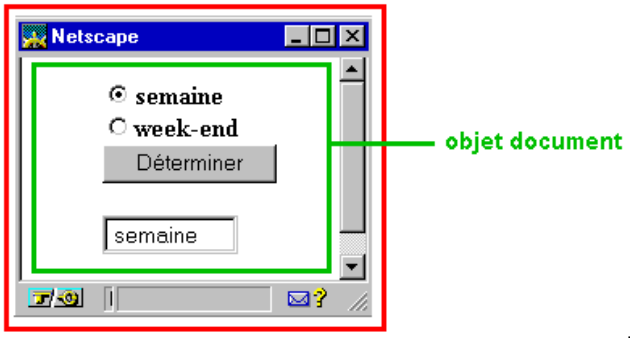
Une page Web est en fait constituée d'une série d'éléments qui peuvent être imbriqués les uns dans les autres. Javascript va diviser cette page en **objets** et surtout va vous permettre d'accéder à ces objets, d'en retirer des informations et de les manipuler.

Pour définir quels objets sont gérés par Javascript, nous allons supposer que vous avez chargé la page suivante dans votre navigateur.

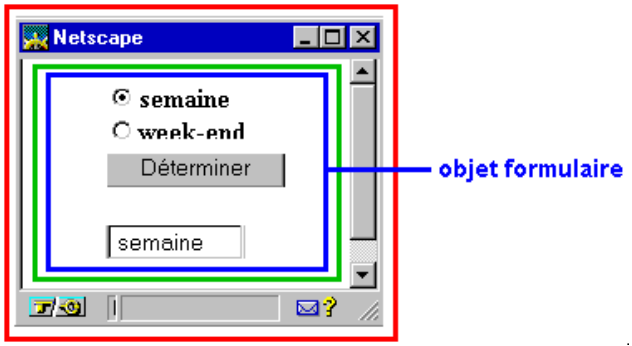


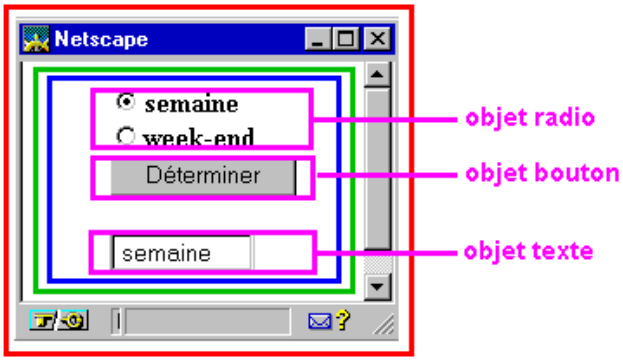
Cette page s'affiche dans une fenêtre du navigateur. C'est l'objet window.

Dans cette fenêtre, une page Html a été chargée. C'est l'objet document. On constate qu'il est inclus dans l'objet fenêtre. On commence à voir la notion de la hiérarchie des objets Javacript.



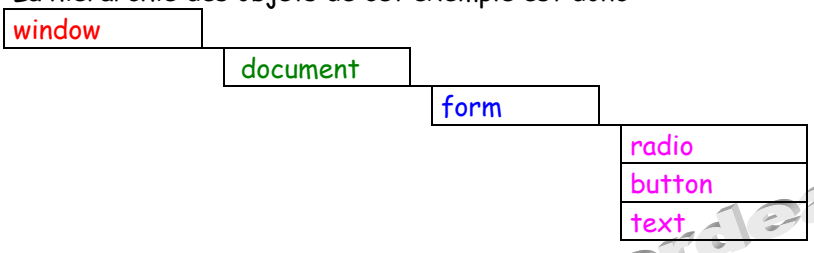
Dans ce document, on y trouve un formulaire de saisie au sens Html. C'est l'objet form contenu dans l'objet fenêtre.





Dans ce formulaire, on trouve deux radio-boutons, un bouton classique et une zone de texte. Ce sont respectivement l'objet radio, l'objet button, l'objet text.

La hiérarchie des objets de cet exemple est donc :



Pour accéder à un objet, il faudra donner le chemin complet de l'objet en allant du contenant le plus extérieur à l'objet référencé.

Exemple :

L'accès au radio-bouton "semaine" sera déterminé par :

```
window.document.form.radio[0]
ou
document.form.radio[0]
```

Comme l'objet **window** occupe la première place dans la hiérarchie, il est repris par défaut par Javascript et devient donc facultatif.

Chaque objet est numéroté. "semaine" étant le premier élément des radio-boutons, il sera appelé radio[0]. Le suivant "week-end" sera radio[1].

Les propriétés des objets et leurs attributs

Une propriété est une valeur, une caractéristique, une description de l'objet. Par exemple, l'objet "livre" a comme propriétés son auteur, sa maison d'édition, son titre, son numéro ISBN, etc. Les objets Javascript ont des propriétés personnalisées selon leur type. Les valeurs sont appelées **attributs**.

Exemple : les radio-boutons peuvent être cochés ou non. La propriété est "checked" et sa valeur (=**attribut**) peut être "true" (bouton coché) ou "false" (bouton non coché). Pour tester si le radio-bouton "semaine" est coché, on écrit :

```
if (document.form.radio[0].checked == true)
{
    ...
}
```

Les méthodes des objets

Non seulement chaque objet peut avoir des propriétés (valeurs particulières) mais il peut également avoir des **fonctions** qui lui sont propres, appelées **méthodes** (ou même **comportements**). L'appel de la méthode se fait selon la notation :

nom_de_l'objet.nom_de_la_méthode()

Exemple :
Il est possible d'écrire du texte dans un document. La méthode s'appelle "write()". Pour afficher du texte dans un document, on écrit :

document.write("<CENTER>Ici, l'ENITA de Bordeaux ! </CENTER><HR> Bienvenue !");

Ce qui donnera dans le navigateur :

Ici, l'ENITA de Bordeaux

Bienvenue !

Le DOM

Nous avons vu que les objets peuvent avoir des propriétés, des méthodes et réagir à des événements. L'ensemble constitue le modèle des objets du document (model object document = **DOM**). Ce modèle est légèrement différent selon le navigateur utilisé. Vous trouverez une présentation succincte du DOM Javascript pour NN et IE en annexe page 44. Mais comme ils évoluent rapidement, pour une consultation précise, il est conseillé d'aller consulter la documentation en ligne de chacun des éditeurs.

Création et manipulation d'objets

Si certains objets du DOM sont prédéfinis, certains autres devront être créés avant d'être utilisés. Il est aussi possible de créer ses propres objets. L'opérateur **new** permet de créer de nouveaux objets ou d'utiliser des objets déjà existants. La plupart des objets Javascript possède 2 syntaxes pour les créer. Soit en utilisant l'opérateur **new**, soit en utilisant une syntaxe propre à l'objet à créer.

Ici, nous n'allons décrire que les objets préexistants qui nécessitent l'opérateur **new** pour être construits.

- Objets déjà existants dans Javascript : opérateur new

  - ✓ **Gestion des dates : new Date**

Javascript possède un objet particulier appelé **Date** dont le but est de permettre la manipulation de données relatives à la date et l'heure. Avant de l'utiliser, il faudra créer ce que l'on pourrait appeler "une copie de l'objet Date" (**instance**) de l'une des manières décrites au chapitre "Gérer le temps : l'objet Date" p 38 dont la plus simple est celle qui donne la date et l'heure en cours.

maDate = new Date()

Une fois l'objet **Date** créé, il est possible de le manipuler pour en extraire le jour, le mois, l'année, l'heure, … à l'aide de ses méthodes (fonctions) et de ses propriétés prédéfinies. Par exemple :

var mois = maDate.getMonth()

✓ Les tableaux : new Array

De même que pour la date, il est nécessaire de créer une instance de l'objet par :

```
monTableau = new Array ("2", "4", "ENITA");
```

ou

```
mon Tableau = new Array (4);          // nbre d'éléments du tableau
```

✓ New Option

Les objets **Option** peuvent être ajoutés dans les listes des <SELECT> par :

```
var nom_option = new Option (texte, valeur, defaultselection, selection);
```

Avec :

- **texte** : texte associé affiché pour l'option dans la liste,
- **valeur** : correspond à la valeur envoyé lors du submit,
- **defaultselection** :booléen indiquant si l'objet est sélectionné par défaut au chargement,
- **selection** lors de l'insertion.

Ceci permet donc de générer des listes déroulantes dynamiquement.

• Création d'un objet personnalisé : opérateurs new et this

Rappel : Un objet correspond à un élément auquel sont associées des **méthodes** (=fonctions) et/ou des **propriétés** qui prendront des valeurs appelées **attributs**.

Ainsi un étudiant peut être défini comme un objet (ne pas y voir de critique !). Il peut être distingué par son nom, son e-mail et sa promo. Ceci s'écrira ainsi :

```
function etudiant (unNom, unEmail, unePromo)
{
    this.nom = unNom;
    this.email = unEmail;
    this.promo = unePromo;
}
```

L'opérateur Javascript **this** permet de créer une ou plusieurs instances de ce type d'objet.

Pour créer un objet de ce type, on écrira :

```
Armand = new etudiant ("Armand Claude",
 "a-claude@enitab.fr","1999-2001");
```

ainsi : Armand.promo vaut "1999-2001"
On appelle cette instruction : **constructeur** ou **constructor**.

Cette fonction **etudiant()** ne comporte que des propriétés. On va lui ajouter une fonction qui permettra de dire en quelle année de cursus, se trouve l'étudiant.

```
function annee (promo)
{
    if (promo == "1999-2001")  an = 3;
    if (promo == "2000-2002")  an = 2;
    if (promo == "2001-2003")  an = 1;
    return an;
}
```

La fonction **etudiant()** devient:

```
function etudiant (unNom, unEmail, unePromo)
{
    this.nom = unNom;
    this.email = unEmail;
    this.promo = unePromo;
    this.an_cursus = annee(promo);
}
```

La page HTML qui affiche les coordonnées scolaires de Armand se présente ainsi :

```
<body>

<script language="javascript">
Armand = new etudiant ("Armand Claude", "a-
claude@enitab.fr","2000-2002");
document.write ("-- " + Armand.nom + " - " + Armand.email +
  " - " + Armand.promo + " - " + Armand.an_cursus + "<HR>");
</script>

</body>
```


- Opérateurs sur les objets : for … in, with, in, instanceof

  ✓ **for … in**

Cette instruction permet d'explorer les propriétés d'un objet. Cela peut être très utile pour vérifier les valeurs des propriétés d'un objet du document Html en cours de création.
Exemple : le code ci-dessous affiche toutes les propriétés et attributs d'une case à cocher appelée "checkbox" située dans un formulaire baptisé "form1".

```
<script language="javascript">
  var resultat="";
  for (var propriete in form1.checkbox)
  {
      resultat+= "propriété : " + propriete + " = " +
              form1.checkbox[propriete] + "<BR>";
  }
  document.write(resultat + "<HR>");  // affiche le nom des
       // propriétés séparées par une ligne
</script>}
```

✓ with

Cet opérateur sert uniquement à épargner de la frappe de code. Les 2 écritures ci-dessous sont strictement équivalentes :

sans opérateur with	avec opérateur with

document.write (document.monForm.bouton1.value)
document.write (document.monForm.bouton2.value)
document.write (document.monForm.bouton3.value)
	with (document.monForm)
	{
	document.write (bouton1.value)
	document.write (bouton2.value)
	document.write (bouton3.value)
	}

✓ typeof

Cet opérateur peut être appliqué à n'importe quel objet Javascript. Il retourne le type de l'objet.

Exemples : typeof "ENITA"; // renvoie "String"
typeof 2.56; // renvoie Number
typeof Math; // renvoie Object

✓ in et instanceof

Ces 2 nouveaux opérateurs sont inclus à partir de Javascript 1.4, aussi aucun navigateur Netscape ne les prend en charge pour l'instant.

Exemple d'utilisation de **in** :

```
var a = ("PI" in Math);          // a contient la valeur true puisque PI appartient bien à l'objet Math.
```

Exemple d'utilisation de **instanceof** :

```
var b = new String;
var c = (b instanceof String);  // c contient la valeur true puisque b est du type String
```

Les objets dans Javascript

Afficher du texte : objet Document

- Méthodes

Instruction	Description	Exemple
write("texte");	affiche le texte et les balises Html dans la page en cours	<BODY> <SCRIPT LANGUAGE="Javascript"> <!--var texte = "Zoé"; document.write ("<CENTER> Bonjour " + texte + "<HR></CENTER>"); //--> </SCRIPT> </BODY>
print();		

Il existe d'autres méthodes de mise en forme des caractères, mais toutes ne sont pas utilisables quelle que soit la version du navigateur. Elles ont la même action que les balises qui portent le même nom.
Exemple : la méthode **bold()** écrit la chaîne de caractères en gras. Les 4 instructions Javascript suivantes sont équivalentes :

```
str="Enita Bordeaux";
document.write("<B>"+str+"</B>");
document.write("<B> Enita Bordeaux </B>");
document.write(str.bold());
document.write("Enita Bordeaux ".bold());
```

On trouve ainsi **big(),small(),blink(), bold(), fixed(), italics(), fontcolor("couleur"), fontsize(taille), strike(), sub(), sup().**

- Propriétés

On peut également mettre en forme le document en utilisant des propriétés. Mais n'oublions pas que ce qui suit est optionnel et que l'on peut utiliser l'instruction **document.write()** de façon classique.
Exemple : pour changer la couleur du fond du document, on peut écrire au choix :

```
document.write("<BODY BGCOLOR="#FFFFFF");
document.bgColor="#FFFFFF";
```

On trouve ainsi **bgColor, fgColor, alinkColor, linkColor, vlinkColor.**

Tester le navigateur : objet Navigator

- Création

L'objet est créé automatiquement par le navigateur. Il permet de détecter certains paramètres du navigateur de l'internaute tels que le nom et la version du navigateur employé, … Ce sera très utile pour adapter les scripts au navigateur et à la version de celui-ci.

- Propriétés

Instruction	Description	Exemple
appCodeName	Retourne le nom de code du navigateur. Ce sera "Mozilla" pour Netscape et aussi pour Microsoft qui a repris ce code (~ inutile)	alert("le code de votre navigateur est " + navigator.appCodeName); // = "Mozilla"
appName	Retourne le nom ou la marque du navigateur (soit "Netscape", soit "Microsoft Internet Explorer")	alert("La marque du browser est " +navigator.appName);
appVersion	Retourne la version du navigateur, le système d'exploitation et le code de nationalité de la version	alert (navigator.appVersion); // ="2.0 (Win95; I)" version 2.0 sous Win95 Internationale
userAgent	retourne la chaîne de caractère qui comprend toutes les informations ci-dessus. La chaîne étant de longueur variable, il ne serait pas évident d'en récupérer une portion sans les autres propriétés...)	

Voici une grille avec les navigateurs, les valeurs de propriétés et quelques fonctionalités associées à ceux-ci:

Navigateur	Version	navigator.appName	navigator.appVersion	Javascript
Netscape Navigator	2	Netscape	2.0 (Win95; I)	1.0
Netscape Navigator	3	Netscape	3.xx (Win95; I)	1.1
Netscape Communicator	4	Netscape	4.xx (Win95; I)	1.1
Microsoft Internet Explorer	3	Microsoft Internet Explorer	2.0 (compatible; MSIE 3.0x; Windows 95)	1.0
Microsoft Internet Explorer	4	Microsoft Internet Explorer	4.0x (compatible; MSIE 4.0x; Windows 95)	1.2
Microsoft Internet Explorer	5	Microsoft Internet Explorer	5.xx (compatible; MSIE 5.0x; Windows 95)	1.2

- Exemples

Netscape 3 ou 4 ?	var nm=navigator.appName+navigator.appVersion; if(nm.indexOf("Netscape3.")>-1||nm.indexOf("Netscape4.")>-1) {…}
Navigateur sous Windows ?	if (navigator.appVersion.IndexOf('Win')>-1) {…}
Explorer 4.0 ?	var ms = navigator.appVersion.indexOf("MSIE") if (ie4 = (ms>0) && (parseInt(navigator.appVersion.substring(ms+5, ms+6)) >= 4)

Javascript 1.0 ou 1.1 ?	var test=navigator.useAgent; if(test.indexOf("2.") != -1) {...}

La compatibilité des pages Javascript avec les différents types et versions en circulation pouvant affecter certains affichages des pages, il est souhaitable d'avertir l'internaute de la version de développement.

Ainsi, le script suivant informe les utilisateurs de Explorer 3.0 qu'ils risquent rencontrer quelques désagréments.

```
<SCRIPT LANGUAGE = "JavaScript">
<!--
    var name = navigator.appName ;
    if (name == 'Microsoft Internet Explorer') {
        document.write('Attention! Vous utilisez Microsoft Explorer
3.0.') <BR>');
         document.write('Avec ce navigateur, certains scripts
peuvent ne pas fonctionner correctement');
    }
//-->
</SCRIPT>
```


Connaître la résolution de l'écran : objet Screen

La présentation d'une page web dépend de la plac que l'on peut lui accorder. Ainsi un internaute qui possède un écran de résolution 800x600 ne pourra voir la même chose qu'un internaute qui travaille sur une résolution supérieure. Il peut être intéressant de tester la résolution grâce à l'objet **Screen**.

Propriétés	Description
Screen.height	donne la hauteur de l'écran en pixel
Sreen.width	donne la largeur de l'écran en pixel
Screen.pixelDepth	Donne la résolution des couleurs (bits / pixel)

La fonction affiche les propriétés de l'affichage courant :

```
function screen_properties()
{
  document.examples.results.value = "("+screen.width+" x
    "+screen.height+") pixels, "+
    screen.pixelDepth +" bit depth, "+
    screen.colorDepth +" bit color palette depth.";
} // end function screen_properties
```

Gérer les fenêtres et les cadres : objet window

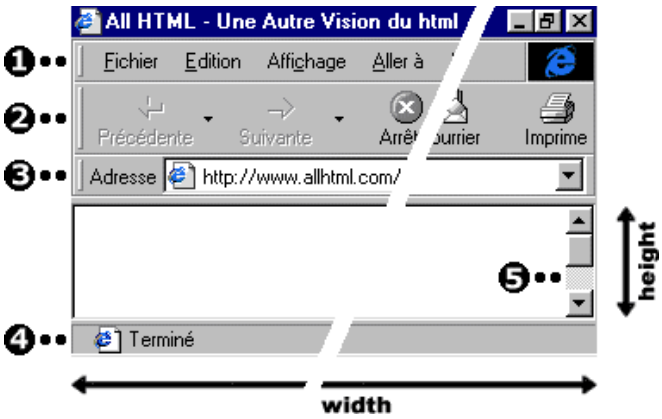
L'objet **window** est compatible avec les navigateurs NN et IE : c'est l'objet maître du JavaScript. Sa référence en devient implicite.
Ainsi, écrire **window.alert("Coucou !")** est équivalent à écrire **alert("Coucou !")**

Il se définit comme une fenêtre, où l'objet **document** est considéré comme le contenu de cette même fenêtre. L'utilisation la plus courante est la création d'une fenêtre volante (**popup**).

- Méthodes

Instruction	Description	Exemple
alert("message")	Permet d'afficher une boîte de dialogue à un bouton "Ok"	alert ("Coucou !");
confirm("message")	Permet d'afficher une boîte de dialogue à 2 boutons "ok" et "annuler"	confirm ("Coucou !");
prompt("message","valeur par défaut")	Affiche une boîte de dialogue dans laquelle l'utilisateur peut saisir l'info demandée	var x=prompt("nom de l'école?","enitab");
open(nom_page, titre_fenetre, parametres)	Ouvre une nouvelle fenêtre	(voir exemple ci-dessous)
close()	Fermer la fenêtre spécifiée	var wnd=open("page.htm","nom_page"); wnd.close();
back()	retourne à l'URL précédente (historique) du navigateur - équivalent à history.go(-1);	window.back();
forward()	URL suivante (historique) du navigateur - équivalent à history.go(1);	window.forward();
focus()	Active la fenêtre spécifiée.	var wnd=open("page.htm","nom_page"); wnd.focus();
blur()	Désactiver la fenêtre spécifiée	
setTimeout()	Active une fonction "chronomètre" (en ms)	
clearTimeout()	Efface la fonction "chronomètre"	

Exemple d'utilisation de la méthode open()



1> menubar (défaut : no).
2> toolbar (défaut : no).
3> location (défaut : no).
4> status (défaut : no).
5> scrollbars (défaut : no)

top : écart en pixels entre le haut de l'écran et le haut de la fenêtre
left : écart en pixels entre la gauche de l'écran et le bord gauche de la fenêtre

La syntaxe générale d'ouverture d'une fenêtre est la suivante :

```
window.open ('nom_page.htm',        // nom du fichier qui s'affiche dans la nouvelle page
             'nom_de_la_fenetre',
             'width=nb_pixels,      // largeur de la fenêtre
             height= nb_pixels,     // hauteur
             top= nb_pixels, // écart entre le haut de l'écran et le haut de la fenêtre
             left= nb_pixels,// écart entre la gauche de l'écran et le bord de la fenêtre
```

```
menubar= yes ou no,      // barre de menu
toolbar= yes ou no,      // barre d'outils
location= yes ou no,     // barre d'URL
status= yes ou no,       // barre d'état
scrollbars= yes ou no,   // barres de défilement (ne s'affichent que si nécessaire)
directories= yes ou no,  // affichage des boutons d'accès rapide
resizable= yes ou no ')">        // dimensions de la fenêtre modifiables
```

Exemples d'une fenêtre avec une barre de menu

```
window.open('fenetre.htm', 'Exemple',
    'width=250,height=300,top=120,left=120,menubar=yes')"> //
sans espaces ni passage à la ligne
```

Exemples d'ouverture d'une fenêtre vierge avec une barre d'outils

```
window.open('', 'Exemple',
    'width=250,height=300,top=120,left=120,toolbar=yes')"> //
sans espaces ni passage à la ligne
```

L'usage des nouvelles fenêtres est sympathique pour afficher des informations complémentaires sans surcharger la page (ou fenêtre) de départ. Cependant, si l'utilisateur ne ferme pas ces nouvelles fenêtres, celles-ci restent ouvertes (et oui !) et le navigateur se retrouve avec plusieurs dizaines de fenêtres ouvertes ce qui ralentit le système et peut finir par le faire planter.
Veillez donc à toujours faire fermer ces nouvelles fenêtres par **window.close();**

- Les propriétés

Instruction	Description	Exemple
defaultStatus	message par défaut qui s'affiche dans la barre d'état.	
frames	(si cadres dans page) tableau qui contient les cadres présents dans la fenêtre	window.frames[n] // n représente le numéro du cadre ou window.nom_du_cadre // nom donné dans la balise **frameset**
self ou **window**	(si cadres dans page) renvoie à la fenêtre en cours	var nom_fenetre=self.name;
name	nom de la fenêtre en cours	
parent	(si cadres dans page) réfère à la page parent	
top	(si cadres dans page) réfère à la fenêtre de plus haut niveau.	
status	valeur du texte affiché dans la barre d'état de la fenêtre.	"self.status='Votre texte'; return true;" Il est indispensable d'ajouter return true;
length	nombre de cadres dans la page	

Gérer les chaînes de caractères : objets String et RegExp

Un certain nombre de fonctions existent pour manipuler les chaînes de caractères. En outre, il existe un objet **RegExp** (expression régulière) qui permet d'effectuer des recherches, des remplacements, des comptages dans des chaînes de caractères.
Certaines fonctions de chaînes peuvent avoir pour paramètre une expression régulière. Ce sont des expressions qui servent pour la recherche et le remplacement de chaînes de caractères.

- Chaînes de caractères : objet String

Rappel : pour concaténer 2 chaînes de caractères, on utilise l'opérateur **+**.
Le 1er caractère d'une chaîne a pour rang 0.

Instruction	Description	Exemple
length	Propriété qui donne la longueur de la chaîne de caractères, mais aussi le nombre de formulaires, de frames, de radio-boutons, de liens,…	var str="ENITA Bx"; var nb_car=str.length; //=8

charAt(x)	Méthode qui permet d'accéder à un caractère isolé d'une chaîne à partir de sa position dans la chaîne.	var str="ENITA Bx";
		var x=str.charAt(1) //="N"
		autre écriture :
		var x=charAt("ENITA Bx",1);

ch.charCodeAt(x)	Méthode qui permet d'accéder au code ASCII d'un caractère isolé d'une chaîne à partir de sa position dans la chaîne.	var str="ENITA Bx"; var x=str.charCodeAt(4) //=65 (Code ASCII de « A »=65)

indexOf(chaine)	Méthode qui renvoie la position d'une chaîne partielle à partir d'une position déterminée n (facultatif si recherche à partir du début de la chaîne donc 0). Retourne –1 si pas trouvé.	var str="ENITA BORDEAUX";
		var x=str.indexOf("TA"); //=3
		var x=str.indexOf("A", 2); //=4
		var x=str.indexOf("W"); //=-1

lastIndexOf(chaine, position)	La méthode est identique à la précédente mais elle recherche la position de droite à gauche de la chaîne mais renvoie la position à partir de la gauche	var str="ENITA BORDEAUX";
		var x=str.lastIndexOf("A", 2); //=-1
		var x=str.lastIndexOf("A"); //=11

substring(x,y)	Méthode qui renvoie la sous-chaîne située entre la position x et la position y-1.	var str="ENITA BORDEAUX";
		var x=str.substring(3, 7); //="TA B"
		var x=str.substring(0, 4); //="ENIT"

toLowerCase()	Transforme toutes les lettres en minuscules.	var str="ENITA BORDEAUX"; var x=str.toLowerCase(); //="enita bordeaux"
toUpperCase()	Transforme toutes les lettres en Majuscules.	var str="Enita Bordeaux"; var x=str.toLowerCase(); //="ENITA BORDEAUX"

split(separateur)	A partir du séparateur donné en paramètre, la fonction retourne un tableau de variables	var str="Enita;Bordeaux";
		var x=str.split(";")
		alert(x[0] + "\n" + x[1]); // Enita
		// Bordeaux

replace(exp_reg, chaine)	Cherche une correspondance entre une expression régulière et une chaîne et remplace la sous-chaîne trouvée par la nouvelle sous-chaîne.	var str="Enita Bordeaux"; var x=str.replace(/a Bordeaux/,"ab") // Enitab

search(exp_reg)	Recherche dans une chaîne la correspondance avec une expression régulière. Renvoie l'index de l'expression trouvée dans la chaîne. (0 si 1er caractère, -1 si pas trouvé.)	var str="Enita Bordeaux";
		var x=str.search(/En/); // =0
		var x=str.search(/a B/); // =4
		var reg=/eauz/
		var x=str.search(reg); // =-1

match(exp_reg)	Recherche une expression régulière dans une chaîne de caractères. Retourne null si l'expression n'est pas trouvée.	var str="Enita Bordeaux";
		var x=str.match(/En/); // ="En"
		var x=str.match(/a B/); // ="a B"
		var reg=/eauz/
		var x=str.search(reg); // =null

- Expressions régulières : objet RegExp

Cet objet possède des propriétés et des méthodes qui permettent l'utilisation d'expressions régulières pour effectuer une recherche et/ou un remplacement dans des chaînes de caractères.
Il se crée ainsi : **/modèle_de_chaine/indicateurs_facultatifs** (syntaxe empruntée au Perl)
ou **var reg = new RegExp ("modèle_de_chaine", "indicateurs_facultatifs");**
Exemple de base : /enit/gi enit = modèle de chaîne gi = indicateurs
ou var reg = new RegExp("enit","gi");

✓ **Les caractères spéciaux et les indicateurs**

L'objet RegExp donne à certains caractères des fonctionnalités particulières. Les voici classés par rôle:

Caractères spéciaux

Caractère spécial	Description	Exemples
. (point)	Recherche n'importe quel caractère unique à l'exception du caractère de début de ligne.	var str1="Enita Bordeaux"; /B..d/ est trouvé dans str1 /B.d/ n'est pas trouvé dans str1
****	si l'antislash \ précède un autre caractère, il enlève le rôle particulier de ce caractère pour qu'il soit interprété comme un caractère non spécial (voir ci-dessous : **^, $, *, +, ?, ., |, \, /**)	var reg = /a*/ // pour rechercher "a*", il faut écrire /a*/ car * est un caractère spécial (voir ci-dessous). 'a*' recherche 0 ou plusieurs a.
\f	Recherche le "form feed" (saut de page).	
\n	Recherche le "line feed" (interligne, saut de ligne).	
\r	Recherche le "carriage return" (retour à la ligne).	
\t	Recherche une tabulation.	
\v	Recherche une tabulation verticale.	
[\b]	Recherche le retour arrière (backspace). Ne pas confondre avec \b	

Caractère spécial	Description	Exemples
\ooctal \xhex	\ooctal est une valeur octale et \xhex est une valeur hexadécimale (permet d'introduire un code ASCII dans une expressions régulière).	

Classe de caractères

Caractère spécial	Description	Exemples
[xyz] [x-z]	Recherche n'importe quel caractère de ceux spécifiés.	var str1="Enita Bordeaux-Gradignan"; var str2="Bordeaux"; /[jklmn]/ ou /[j-n]/ est trouvé dans str1
[^xyz]	Recherche n'importe quel caractère qui ne font pas partie de ceux spécifiés.	var str1="Enita Bordeaux-Gradignan"; var str2="Bordeaux"; /[^jklmn]/ ou /[^j-n]/ est trouvé dans str2
\d	Recherche les nombres. Equivalent à [0-9]	
\D	Recherche tous les caractères sauf les nombres. Equivalent à [^0-9]	
\w	Recherche n'importe quel caractère alphanumérique, y compris le souligné (_). Equivalent à [A-Za-z0-9_].	
\W	Recherche n'importe quel caractère non alphanumérique, y compris le souligné (_). Equivalent à [^A-Za-z0-9_].	
\s	Recherche un espace, un form feed, un line feed ou une tabulation. Equivalent à [\f\n\v\t\r].	
\S	Recherche un autre caractère qu'un espace. Equivalent à [^ \f\n\v\t\r].	
*	Recherche le caractère précédent 0 ou plusieurs fois (mais pas une seule fois).	var str1="Enita Bordeaux"; var str2="Bordeaux"; /a*/ est trouvé dans str1
+	Recherche le caractère précédent 1 ou plusieurs fois.	var str1="Enita Bordeaux"; var str2="Bordeaux"; /a+/ est trouvé dans str1 et str2
?	Recherche le caractère précédent 0 ou 1 fois.	var str1="Enita;Bordeaux"; var str2="Bordeaux"; /ta?/ est trouvé dans str1 et str2 /a?/ est trouvé dans str2 /a?;/ est trouvé dans str1
{n}	Quand n est un entier positif, il recherche exactement n occurrences du caractère précédent.	var str1="Enita Bordeaux-Gradignan"; var str2="Bordeaux"; /a{2}/ est trouvé dans str1 mais ne prend que les 2 premiers "a" de str1

Caractère spécial	Description	Exemples
{n,}	Quand n est un entier positif, il recherche au moins n occurrences du caractère précédent.	var str1="Enita Bordeaux-Gradignan"; var str2="Bordeaux"; /a{2}/ est trouvé dans str1 et prend tous les "a" de str1
{n,m}	Quand n et m sont des entiers positifs, recherche au moins n et au plus m occurrences du caractère précédent.	var str1="Enita Bordeaux-Gradignan"; var str2="Bordeaux"; /a{2,3}/ est trouvé dans str1 et prend les 3 premiers "a" de str1
a|b	entre 2 caractères il recherche l'un ou l'autre des caractères.	var str1="Enita"; var str2="Bordeaux"; /n|w/ est trouvé dans str1

Position

Caractère spécial	Description	Exemples
^	Correspond au début de la ligne.	var str1="Enita Bordeaux"; var str2="Bordeaux"; /^Bo/ est trouvé dans str2
$	Correspond à la fin de la ligne	var str1="Enita Bordeaux"; var str2="Enita Bordeaux – BP201"; /$aux/ est trouvé dans str1
\b	Recherche dans l'extrémité d'un mot. A ne pas confondre avec [\b]	var str1="Enita Bordeaux"; var str2="Bordeaux"; /x\b/ est trouvé dans str1 et str2 /\bE\w/ est trouvé dans str1
\B	Recherche à l'intérieur d'un mot.	var str1="Enita Bordeaux"; var str2="Bordeaux"; /\w\Bi/ correspond à 'ni' dans str1

Mémorisation des correspondances trouvées

Caractère spécial	Description	Exemples
( )	quand un modèle de recherche est encadré de ( ), sa valeur peut être récupérée dans une variable appelée $1, $2, … (voir ci-dessous)	var str1="Enita Bordeaux"; var reg = /(\w\B)i(\w)/ correspond à 'nit' dans str1 $1 = "n" $2 = "t"

Autre exemple : soit le modèle :
Reg3=/\w+\s(\w+)\s(\d+)/
Quand une correspondance est détectée dans la chaîne, chacune des sous-chaînes trouvées et encadrées de parenthèses peut être mémorisée dans $1,..,$9.
Ainsi la chaîne ch='dupont albert 56' sera transformée par ch.replace(Reg3,'$2 $1') en ch='56 albert'.

En plus des caractères spéciaux ci-dessus, on peut ajouter **2 indicateurs** :

indicateur (facultatif)	description
g	"global match" : recherche sur toute la chaîne de caractères
i	"ignore case" : ne fait plus la distinction entre majuscules et minuscules
gi	combine les 2 premiers paramètres

✓ Les propriétés

Propriétés	Description	Exemple
input		var reg=/nit/; var presence=reg.test (="Enita"); // = true
lastparen ou **$+**		
rightContext ou **$'**		
global	retourne "true" si l'indicateur **g** a été utilisé, sinon retourne "false"	var reg = /nit/gi reg.global vaut true;
lastIndex		
leftContext ou **$\Q**		
source		
ignoreCase	retourne "true" si l'indicateur **i** a été utilisé, sinon retourne "false"	var reg = /nit/gi reg.ignoreCase vaut true;
lastMatch ou **$&**		
multiline		
$1, ...$9	Correspondance avec les sous-chaînes entre parenthèses. Le nombre de sous-chaînes est illimité mais RegExp ne peut en traiter que 9.	exemple ci-dessous

✓ Les méthodes

Instruction	Description	Exemple
test(chaine)	Similaire à la méthode String.search(), à la différence qu'elle retourne true ou false	var str1="Enita Bordeaux"; var reg=/nit/; var presence=reg.test (str1); // true
exec(chaine)	Proche de la méthode String.match(). Recherche le modèle dans la chaîne	voir exemple ci-dessous

Exemple :

```
var s = "une CHAÎNE de caractères."
var re = new RegExp("ne\\b","gi");
while (true) {
  res = re.exec(s);
  if ( res == null ) break;
  document.writeln("<P>J'ai trouvé \""+res[0]+"\" en position "+
                res.index+" dans \""+res.input+"\"");
}
```

donne :

J'ai trouvé "ne" en position 1 dans "une CHAÎNE de caractères."
J'ai trouvé "NE" en position 8 dans "une CHAÎNE de caractères."

Autre exemple :

```
re = /(\w+)\s(\w+)/;
str = "Jean Dupont";
newstr=str.replace(re, "Nom: $2 Prénom: $1");
document.writeln("<P>"+newstr)
```

donne :

Nom: Dupont Prénom: Jean

En savoir plus : http://developer.netscape.com/docs/manuals/js/client/jsref/regexp.htm

Saisir des données dans un formulaire : objet Form

- ## Généralités

Avec Javascript, les formulaires vont devenir plus interactifs. En effet, avant d'envoyer les données au serveur, il sera facile de contrôler la validité des données (absence de saisie, donnée conforme à un masque de saisie, …).
Les objets des formulaires sur lesquels un contrôle ou une fonction devra agir, nécessitent un nom. Chaque objet sera appelé **contrôle** (=widget) et sera représenté par des balises Html <INPUT … Name=… >, <SELECT Name=…> ou <TEXTAREA Name=…>. Et comme nous avons vue la hiérarchie des objets, le formulaire lui-même devra être baptisé.

- ## Accéder à un formulaire

Elle se fait par les balises **<FORM Name="nom_formulaire" …>** et **</FORM>**. En Javascript, l'attribut **NAME** a toute son importance pour désigner le chemin complet de ses contrôles.
Le formulaire étant un élément de l'objet document, pour accéder à **"nom_formulaire"**, il faut écrire :

```
document.forms["nom_formulaire"]
ou
document.forms[0]
ou
document.nom_formulaire
```

Remarque : Les attributs **ACTION** et **METHOD** sont facultatifs si on ne fait pas appel au serveur.

- ## Accéder et manipuler à un élément du formulaire

Pour accéder à une zone de texte qui serait baptisée "zt" dans le formulaire précédent par l'instruction Html suivante :
<INPUT TYPE="text" Name="zt">
on écrit :

```
document.forms["nom_formulaire"].elements["zt"]
ou
document.forms["nom_formulaire "].elements[0]
ou
document.forms["nom_formulaire "].zt
```

Ensuite, on peut manipuler cet objet.
Exemple : placer la valeur "Enitab" dans la zone de texte "zt"

```
document.forms["nom_formulaire"].elements["zt"].value="Enitab"
```

Autre exemple : donner le focus à l'élément "zt"

```
document.forms["nom_formulaire "].elements["zt"].focus()
```

- Le contrôle "zone de texte" : <INPUT TYPE="text">

La zone de texte est l'élément d'entrée/sortie par excellence de Javascript. La syntaxe Html est :
<INPUT TYPE="text" Name="nom" SIZE=x MAXLENGTH=y>
pour un champ de saisie d'une seule ligne, de longueur x et de longueur maximale de y.

L'objet **zone de texte** possède trois propriétés :

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

- Les radio-boutons et les cases à cocher : <INPUT TYPE="radio | checkbox">

Généralement, les radio-boutons sont utilisés pour sélectionner une option et une seule parmi plusieurs. Les cases à cocher servent à indiquer si l'option est choisie ou non. Leurs syntaxes respectives en Html sont :
<INPUT TYPE="radio" Name="nom" >
<INPUT TYPE="checkbox" Name="nom" >
Si on ajoute **CHECKED**, cela signifie que le contrôle s'affiche dans l'état "sélectionné".

Propriété	Description
name	indique le nom du contrôle. Dans le cas des radio-boutons, tous les boutons portent le même nom. Chacun est alors repéré par un numéro d'index.
index	l'index ou le rang du bouton radio en commençant par 0.
checked	indique l'état en cours de l'élément radio
defaultchecked	indique l'état du bouton sélectionné par défaut.
value	indique la valeur de l'élément radio.
form	
type	type du champ *text, button, radio, checkbox, submit, reset*

Exemple avec des radio-boutons:

```
<HTML>
<HEAD>
<SCRIPT language="javascript">
function choix (nom_form)
{
    if (nom_form.choix3a[0].checked) { alert("Option choisie : " + nom_form.choix3a[0].value) };
    if (nom_form.choix3a[1].checked) { alert("Option choisie : " + nom_form.choix3a[1].value) };
    if (nom_form.choix3a[2].checked) { alert("Option choisie : " + nom_form.choix3a[2].value) };
}
</SCRIPT>
</HEAD>
<BODY>
<H2>Options de 3ème années choisies : </H2>
<FORM NAME="form3a">
    <INPUT TYPE="radio" NAME="choix3a" VALUE="Viti">Viticulture<BR>
    <INPUT TYPE="radio" NAME="choix3a" VALUE="Eco">Economie<BR>
```

```
    <INPUT TYPE="radio" NAME="choix3a" VALUE="Foret">Forêt<BR>
        <INPUT TYPE="button" VALUE="Votre choix de 3ème année ?" onClick="choix (form3a)">
</FORM>
</BODY>
</HTML>
```

Dans le formulaire form3a, on a créé 3 radio-boutons auxquels on a donné le même nom. Le bouton de type "**Button**" déclenche la fonction **choix()**. Elle teste quel radio-bouton est coché : soit choix3a[0], choix3a[1], choix3a[2]. Si la propriété **checked** (=sélectionné) est vraie (true), alors un message affiche de choix sélectionné dans une boite d'alerte.

- Liste de sélection ou liste déroulante : <SELECT> <OPTION>

Ce contrôle permet de proposer diverses options sous la forme d'une liste déroulante. L'utilisateur y fait son choix qui reste alors affiché. La syntaxe Html est :

```
<SELECT Name ="nom">
    <OPTION Value = "valeur1">La 1ère valeur
    <OPTION Value = "valeur2">La 2ème valeur
</SELECT>
```

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

Exemple : reprend l'exemple donné dans la partie "radio-boutons"

```
<HTML>
<HEAD>
<Script language="javascript">
function choix(nom_form)
{
  alert("Elément " + (nom_form.choix3a.selectedIndex + 1) + " choisi : + "<BR>" +
        "sa valeur (value=) est " + nom_form.choix3a[nom_form.choix3a.selectedIndex].value
        "<BR>sa valeur affichée est : " + nom_form.choix3a[nom_form.choix3a.selectedIndex].text);
}
</SCRIPT>
</HEAD>
<BODY>
<H2>Options de 3ème années choisies : </H2>
<FORM NAME="form3a">
<SELECT NAME="choix3a">
    <OPTION VALUE="Viti "> Viticulture
    <OPTION VALUE="Eco">Economie
    <OPTION VALUE="Foret">Forêt
</SELECT>
```

```
        <INPUT TYPE="button" VALUE=" Votre choix de 3ème année ?" onClick="choix(form3a)">
</FORM>
</BODY>
</HTML>
```

Dans le formulaire **form3a**, on a créé une liste de sélection (balise <SELECT></SELECT>). Les différents éléments de la liste sont définis par les balises <OPTION>. Le bouton de type "**Button**" déclenche la fonction **choix()**. Elle teste quelle option a été sélectionnée. La propriété **selectedIndex** renvoie l'index de l'élément sélectionné, et comme il commence à 0, il ne faut pas oublier d'ajouter 1 pour retrouver le juste rang de l'élément affiché dans la boîte d'alerte. Le message termine par la valeur de l'élément choisi (value).
A l'utilisation : si l'utilisateur choisit dans la liste déroulante, l'option "Economie", à l'écran s'affiche la boîte d'alerte suivante :

Elément 2 choisi : sa valeur (value=) est Eco sa valeur affichée est : Economie

Gestion dynamique d'une liste de sélection
Accès à la i^{éme} option de la liste nommée « liste ».

```
var a ;
a=liste.options[i].text ;
```

Création d'une nouvelle option dans la liste.
Il faut avant tout créer une nouvelle option :

```
var o ;
o=new Option(<libellé>,<valeur>);
```

Le paramètre <libellé> détermine le texte qui apparaîtra dans la liste.
Ensuite, il faut ajouter cette nouvelle option en fin de liste (ici la liste de sélection se nomme « liste » :

```
liste.options[liste.options.length]=o;
```

Suppression de la i^{ème} option de la liste :

```
list.options[i]=null
```

Cas particulier, la suppression de toutes les options de la liste :

```
liste.options.length=0 ;
```

- ## Le contrôle textarea : <TEXTAREA>

Il s'agit en fait d'une zone de texte de plusieurs lignes. Sa syntaxe Html est :
<TEXTAREA NAME="nom" ROWS=x COLS=y> … </TEXTAREA>
où ROWS=x représente le nombre de lignes et COLS=y représente le nombre de colonnes.

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

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

Remarque : Si le texte comporte plusieurs ligne, en Javascript, on utilisera **\r\n** pour passer à la ligne.
Comme par exemple dans l'expression document.Form.Text.value = 'Check\r\nthis\r\nout'.

- Les contrôle Reset et Submit : <INPUT TYPE="reset | submit">

Reset permet d'annuler les modifications apportées à tous les contrôles d'un formulaire et de revenir aux valeurs par défaut. Submit valide le formulaire et envoie les données saisies à l'URL désigné dans l'attribut ACTION de la balise <FORM>. Les syntaxes Html sont respectivement :
<INPUT TYPE="reset" NAME="nom" VALUE "texte">
<INPUT TYPE="submit" NAME="nom" VALUE "texte">
où VALUE donne le texte qui sera lisible sur du bouton.

Une seule méthode est associée aux 2 contrôles. C'est la méthode **onClick()**. Elle peut servir, par exemple, pour faire afficher une autre valeur que celle par défaut.

- Le contrôle Hidden (caché) : <INPUT TYPE="hidden">

Ce contrôle permet d'entrer dans le script des données qui n'apparaîtront pas à l'écran, donc cachés.
Sa syntaxe Html est :
<INPUT TYPE="hidden" NAME="nom" VALUE "les données cachées">

Gérer l'historique : 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 adresses des pages (URL) visitées par l'utilisateur. Ces adresses sont accessibles par le navigateur en cliquant sur les boutons suivant et précédent.

Propriété	Description
length	donne le nombre d'objets dans l'historique
back	permet d'aller à l'URL précédent dans l'historique
forward	permet d'aller à l'URL suivant dans l'historique
go(nombre)	permet d'aller à une des URL de l'historique. "nombre" détermine le nombre de pages relatif auquel se trouve l'URL désiré.

Gérer le temps : l'objet Date

- Création

```
maDate = new Date();            / heure et date en cours
maDate = new Date(940000000000);          // millisecondes depuis le 1er jan 1970 00:00:00
maDate = new Date("Jan 1, 1999");
maDate = new Date(1999, 12, 31, 8, 0);          // 31 déc 1999    08:00:00
```

- Quelques méthodes

Soit l'exemple : date = 29 mai 2001 13 h 32 mn 33 s

Propriétés	Description	Exemple
getDate()	extrait le rang du jour dans le mois	maDate.getDate() // =29

Propriétés	Description	Exemple
getDay()	extrait le jour de la semaine	maDate.getDay() // = 2 (=mardi) car 0=dimanche
getMonth()	extrait le mois	maDate.getMonth() // = 4 (=mai) car 0=janvier
getYear()	extrait l'année	maDate.getYear() // = 2001
getHours()	extrait l'heure du jour	maDate.getHours() // = 13
getMinutes()	extrait le nombre de minutes	maDate.getMinutes() // = 32
getSeconds()	extrait le nombre de secondes	maDate.getSecondes() // = 33
getMilliseconds()	extrait le nombre de millisecondes	
getTime()	extrait le nombre de millisecondes écoulées depuis le 1er janv 1970 0h	maDate.getTime() // =30991135953404
getTimezoneOffset()	extrait la différence en minutes entre le fuseau horaire local et UTC (temps universel coordonné)	maDate.getTimezoneOffset() // =-120 (2 heures de décalage par rapport à UTC
toGMTString()	retourne la date et l'heure UTC	maDate.toGMTString () // Tue, 29 May 2001 11:32:33 UTC
toLocaleString()	retourne la date et l'heure en format local	maDate.toGMTString () //mardi 29 mai 2001 13:32:33
setDate(nbre)	définit le jour du mois de l'objet Date (nbre [1, 31])	maDate.setDate(31); document.write(maDate.getDate());//=31
setHours(nbre)	définit l'heure de l'objet Date (nbre [0, 23])	today.setHours(21); document.write(maDate.getHours());//=21
setMinutes(nbre)	définit les minutes de l'objet Date (nbre [0, 59])	
setYear(nbre)	définit l'année de l'objet Date	
setTime(nbre)	définit la date souhaitée (nbre = nombre de millisecondes écoulées depuis le 1 janvier 1970 - 0 h)	

Calculer : objet Math

L'objet **Math** permet de manipuler des nombres et contient des fonctions mathématiques courantes

- Méthodes

Propriétés	Description	Exemple

abs(nb)	Retourne la valeur absolue d'un nombre	x = Math.abs(3.26) // x = 3.26
		x = Math.abs(-3.26) // x = 3.26

ceil(nb)	Retourne l'entier supérieur ou égal à la valeur donnée en paramètre	x = Math.ceil(6.01) // x = 7
		x = Math.ceil(3.99) // x = 4

floor(nb)	Retourne l'entier inférieur ou égal à la valeur donnée en paramètre	x = Math.floor(6.01) // x = 6
		x = Math.floor(3.99) // x = 3

round(nb)	Arrondit à l'entier le plus proche la valeur donnée en paramètre	x = Math.round(6.01) // x = 6
		x = Math.round(3.8) // x = 4
		x = Math.round(3.5) // x = 4

Propriétés	Description	Exemple

max(nb1, nb2)	Retourne le plus grand des deux entiers donnés en paramètre	x = Math.max(6, 7.25) // x = 7.25
		x = Math.max(-8.21, -3.65) //x=-3.65
		x = Math.max(5, 5) // x = 5

min(nb1, nb2)	Retourne le plus petit des deux entiers donnés en paramètre	x = Math.min(6, 7.25) // x = 6
		x = Math.min(-8.21, -3.65) // x = -8.21
		x = Math.min(5,5) // x = 5

pow(val1, val2)	Retourne le nombre Val1 à la puissance Val2	x = Math.pow(3, 3) // x = 27
		x = Math.pow(9, 0.5) // (racine carrée) x = 3

random()	Retourne un nombre aléatoire compris entre 0 et 1	x = Math.random() // x = 0.6489534931546957
sqrt(Valeur)	Retourne la racine carrée du nombre passé en paramètre	x = Math.sqrt(9) // x = 3
exp(nb)	Retourne l'exponentielle de la valeur nb	
log(nb)	Retourne le logarithme de la valeur nb	
sin(nb)	Retourne le sinus de la valeur nb (donnée en radians)	
asin(nb)	Retourne l'arcsinus de la valeur nb (donnée en radians)	
cos(nb)	Retourne le cosinus de la valeur nb (donnée en radians)	
acos(nb)	Retourne l'arcosinus de la valeur nb (donnée en radians)	
tan(nb)	Retourne la tangente de la valeur nb (donnée en radians)	
atan(nb)	Retourne l'arctangente de la valeur nb (donnée en radians)	

- Propriétés

Propriétés	Description
Math.PI	Retourne la valeur du nombre PI, soit environ 3.1415927
Math.SQRT1_2	Propriété qui retourne la valeur de 1 sur racine de 2 (0.707)
Math.SQRT2	Racine de 2 (1.414)
Math.E	Propriété qui retourne le nombre d'Euler (environ 2.718)
Math.LN2	Retourne le logarithme népérien de 2
Math.LN10	Retourne le logarithme népérien de 10
Math.LOG2E	Propriété qui retourne la valeur du logarithme du nombre d'Euler en base 2

Conserver une information chez le client : objet Cookie

- Qu'est-ce-que c'est ?

Un **cookie** permet de stocker de manière permanente des informations sur le poste de l'internaute. Elles pourront être récupérées lors de visites ultérieures. Les cookies sont très utilisés notamment par les sites commerciaux.

Exemples d'informations pouvant être stockées dans les cookies :

- nombre de visites, date de la dernière visite, ...
- identifiant et un mot de passe pour une reconnaissance du visiteur
- informations à transférer d'une page à l'autre du site.,
- ...

Un cookie est un fichier texte qui contient une chaîne de caractères contenant les informations les unes à la suite des autres.
Le cookie étant une propriété de l'objet **document**, on y accède par :

```
document.cookie
```

- ## Propriétés

Propriétés	Description
name	le nom donné au cookie
value	sa valeur. C'est une chaîne de caractères standard qu'il faudra analyser par les fonctions **escape()** et **unescape()**
expires	date d'expiration : si elle est absente, le cookie est détruit à la fermeture du navigateur. Donnée sous la forme "Wdy, DD-Mon-YY HH:MM:SS GMT"». Pour détruire un cookie, on lui donne une date passée.
path	indique d'où vient le fichier cookie. Limite encore plus la possibilité d'être lu par d'autres pages Javascript
domain	le nom du domaine qui a été écrit le cookie (afin qu'un autre site ne puisse lire le cookie)
secure	accès au cookie sécurisé ou non (valeur false ou true)

Les spécifications des cookies sont données sur le site
http://www.cis.ohio-state.edu/htbin/rfc/rfc2109.html

- ## Enregistrer un cookie

Pour créer un cookie, on écrit :

```
document.cookie ="monCookie=qqChose;".
```

Exemple : pour écrire le nom du navigateur utilisé, on écrira :

```
document.cookie="navig=" + escape(navigator.appName);
```

Il est nécessaire d'utiliser **escape()** afin de transformer la chaîne en suite de caractères Ascii interprétable dans une adresse URL (gestion des caractères espaces par exemple)

Sur la toile, il existe des fonctions toutes faites. En voici une qui crée un cookie :

```
function SetCookie ( name, value, expires, path, domain, secure) {
   szCookie = name + "=" + escape (value) +
      ((expires) ? "; expires=" + expires.toGMTString() : "") +
      ((path) ? "; path=" + path : "") +
      ((domain) ? "; domain=" + domain : "") +
      ((secure) ? "; secure" : "");
   document.cookie = szCookie;
}
```

Donc pour stocker la valeur "enitab" dans la variable "ecole" il suffit d'appeler la fonction comme ceci :

```
    var pathname=location.pathname;
    var myDomain=pathname.substring(0,pathname.lastIndexOf('/'))
+'/';
    var date_exp = new Date();
    date_exp.setTime(date_exp.getTime()+(365*24*3600*1000));
    // Ici on définit une durée de vie de 365 jours
    SetCookie("ecole","enitab",date_exp,myDomain);
```

- Lire les informations d'un cookie

Voyons un cookie qui contiendrait les informations suivantes :

- prenom = Arthur
- nb_visite = 3

Pour récupérer l'information prenom, il faut extraire, de la chaîne de caractères composant le cookie, le nom de la variable soit prenom. La valeur de prenom est la chaine de caractères située immédiatement après et séparée par ';' et par ' '.
Voici le code des fonctions nécessaires à la récupération d'une information :

```
function getCookieVal(offset) {
    var endstr=document.cookie.indexOf (";", offset);
    if (endstr==-1)
                endstr=document.cookie.length;
    return unescape(document.cookie.substring(offset, endstr));
}
function GetCookie (name) {
    var arg=name+"=";
    var alen=arg.length;
    var clen=document.cookie.length;
    var i=0;
    while (i<clen) {
            var j=i+alen;
            if (document.cookie.substring(i, j)==arg)
                return getCookieVal (j);
            i=document.cookie.indexOf(" ",i)+1;
                if (i==0) break;}
    return null;
}
```


Si la variable demandée n'est pas contenue dans le cookie, elle est considérée comme valant null.
Pour récupérer la variable prenom, il suffit d'appeler la fonction :

```
le_prenom=GetCookie("prenom");
```

Animer une page : Calques et styles

Le DHTML introduit la notion de **calque** (ou **couche** ou **layer**). Javascript permet la manipulation de leur position, dimension, visibilité et style d'affichage (police, attributs de police, couleurs).

- Qu'est-ce que c'est ?

Un **calque** est défini par le couple de balises **<DIV>** et **</DIV>** et par :

- un identifiant, pour le repérer
- un style d'affichage, pour indiquer sa position (absolute, top:, left:) sa taille (width:, height:), sa couleur (background-color:, color:), son rang d'affichage (z-index:), sa visibilité (visibility:) etc...
- un contenu, qui sera affiché

Exemple : Le code suivant place le tableau sur fond jaune contenant le texte ***Enita Bordeaux*** dans un calque dont le coin supérieur gauche est à 200 pixels du haut de la page et à 100 pixels du bord gauche.

```
<DIV id="moncalque"
style="position:absolute;top:200px;left:100px;visibility:hidden;
background-color:yellow">
Enita Bordeaux
</DIV>
```

- **L'identifiant** est enregistré grâce à l'attribut **id**. Sur une page, l'identifiant de tous les calques (et de tous les autres objets) doit être unique, pour éviter les ambiguïtés. Une erreur ne surviendra pas forcément en cas de doublons, mais le fonctionnement normal risque d'être perturbé.
- Le **style** est enregistré grâce à l'attribut style. C'est une chaîne de caractères au format **css** (voir les feuilles de style)
- Le **contenu** est du code HTML (texte, tableaux, images...) placé entre les balises <DIV et </DIV>

- Manipuler les calques avec Javascript

Le calque peut être accédé par Javascript grâce à la fonction **document.getElementById** qui retourne l'objet calque.

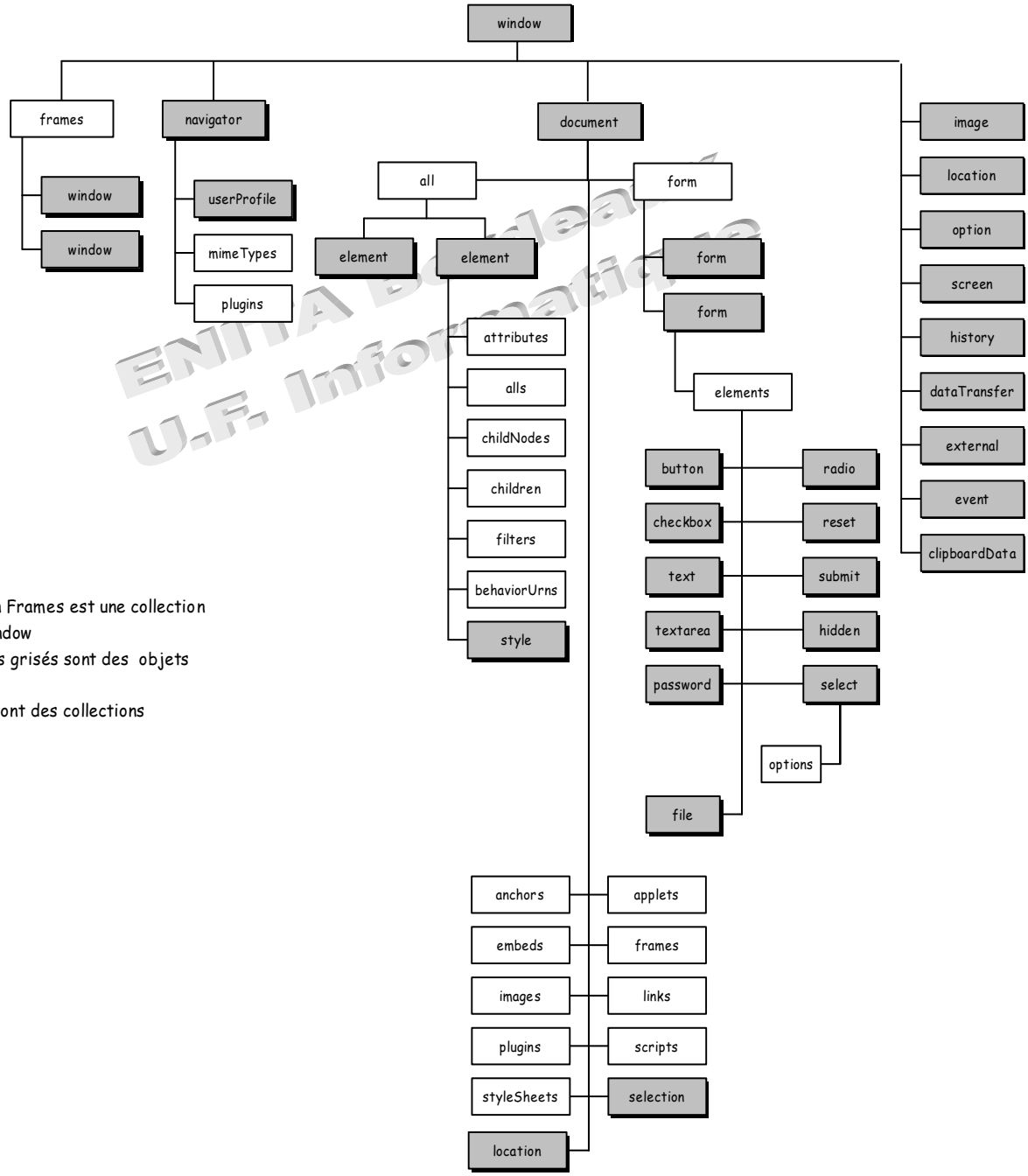
Propriétés	Description
document.getElementById("moncalque").style.visibility="visible";	le calque est visible
document.getElementById("moncalque").style.visibility="hidden";	le calque est masqué
document.getElementById("moncalque").style.left=50;	le calque est à 50 pixels du bord gauche de l'écran
document.getElementById("moncalque").style.top=100;	le calque est à 10 pixels du bord haut de l'écran
document.getElementById("moncalque").style.top= parseInt(document.getElementById("moncalque").style.top)+10;	le calque est à 10 pixels + bas que sa valeur initiale
document.getElementById("moncalque").style.width=100;	largeur du calque = 100 pixels
document.getElementById("moncalque").style.height=30;	hauteur du calque = 30 pixels
document.getElementById("moncalque").style.backgroundColor= "red";	couleur du fond = rouge
document.getElementById("moncalque").style.color="blue";	couleur du texte = bleu

Attention : en Javascript, il n'est possible de modifier un attribut seulement si il a été défini dans la création du calque dans la zone de style de la balise DIV.

Annexe

Le DOM

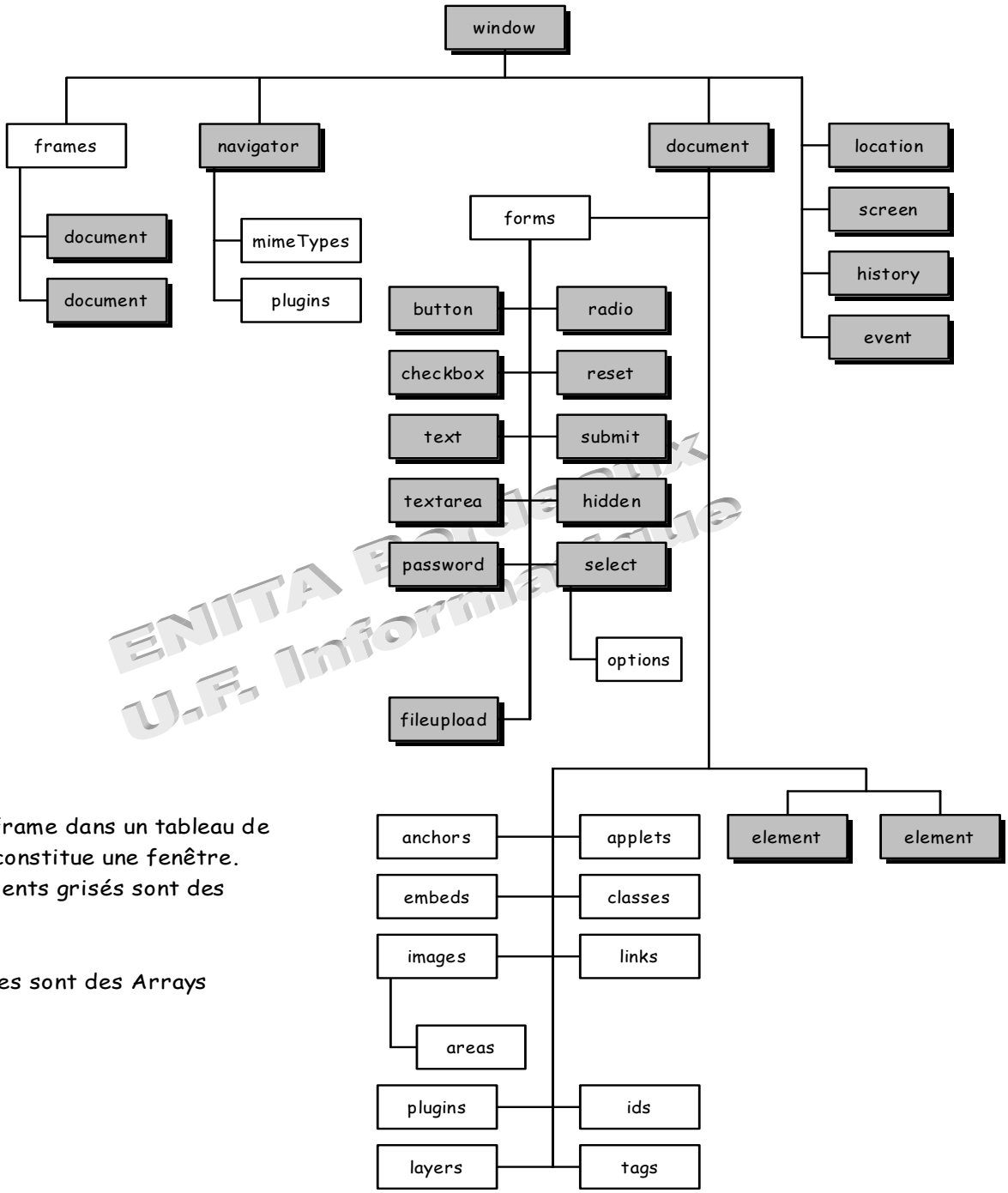
- Le DOM de Internet Explorer



La collection Frames est une collection d'objets window
Les éléments grisés sont des objets

Les autres sont des collections

- Le DOM de Netscape



Chaque frame dans un tableau de frames constitue une fenêtre.
Les éléments grisés sont des objets

Les autres sont des Arrays

Caractères spéciaux

Caractère	Code Javascript
Espace arrière	\b
Saut de page	\f
Alinéa (retour ligne)	\n
Retour chariot	\r
Tabulation	\t

Apostrophe	\'
Guillemet	\"
Antislash	\\

Les opérateurs

Les opérateurs vont permettre de manipuler les variables.

- Les opérateurs de calcul

Dans les exemples ci-dessous, on va supposer que la valeur initiale de x est 11

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

- Les opérateurs de comparaison

Opérateur	Description	Exemples	
==	Egal	if (w == 1) { … }	Si w égal 1 alors …
!=	Différent	if (w != 1) { … }	Si w différent de 1 alors …
<	Inférieur	if (w < 1) { … }	Si w plus petit que 1 alors …
<=	Inférieur ou égal	if (w <= 1) { … }	Si w plus petit ou égal à 1 alors …
>	Supérieur	if (w > 1) { … }	Si w plus grand que 1 alors …
>=	Supérieur ou égal	if (w >= 1) { … }	Si w plus grand ou égal à 1 alors …
? :	Opérateur ternaire	x = (w>0) ? 3 : 5	Si w>0 alors x=3, sinon x=5

- Les opérateurs associatifs

Ils sont à la fois opérateur de calcul et d'attribution de valeur. (Dans les exemples suivants **x** vaut 11 et **y** vaut 5)

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

- Les opérateurs logiques (ou booléens)

Opérateur	Description	Exemples	
!	Opposé ("Not" = opérateur booléen)	if (! w) { … }	Si w non vide (= non nul) alors …

&&	Et	if (x>=1) && (x<=10) { ... }	Si x>=1 ET x<=10 alors ... (si x compris entre 1 et 10 inclus)
||	Ou	if (w>=1) || (x==2) { ... }	Si w>=1 OU x=2 alors ...

- ## Les opérateurs de données binaires

Ce type d'opérateur traite ses opérandes comme des données binaires. Les opérateurs suivants effectuent des opérations bit-à-bit, c'est-à-dire avec des bits de même poids.

Opérateur	Description	Exemples avec x = 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 suivants effectuent des rotation 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.

Opérateur	Description	Exemples avec x = 7	
<<	Rotation à gauche	Décale les bits vers la gauche (multiplie par 2 à chaque décalage). 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 "0" 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 à droite avec remplissage de zéros	Décale les bits vers la droite (divise par 2 à chaque décalage). Les "0" qui sortent à droite sont perdus, tandis que des "0" sont insérés à gauche	6 >>> 1 (0110 >>> 1) // 3 (0011)

- ## Les opérateurs d'incrémentation

Ces opérateurs vont augmenter ou diminuer la valeur de la variable d'une unité. Ce qui sera fort utile, par exemple, pour mettre en place des boucles.
Dans les exemples x vaut 3.

Opérateur	Description	Exemple	Signification	Résultat
x++	Incrémentation (instruction équivalente à x=x+1)	y = x++	3 puis plus 1	4
x--	Décrémentation (instruction équivalente à x=x-1)	y= x--	3 puis moins 1	2

Remarque :
On trouve aussi l'écriture suivante : ++x ou --x. L'incrémentation (ou la décrémentation) a lieu avant exécution de l'instruction. Ainsi :
y = x++ + 2 si x=3 alors y = 5 puis x = 4
alors que y = ++x + 2 si x=3 alors x= 4 puis y = 6
Donc, il convient d'être très vigilent lorsqu'on utilise ces opérateurs combinés à d'autres.

Priorités des opérateurs Javascript
Les opérateurs s'effectuent dans l'ordre suivant de priorité (du degré de priorité le plus faible ou degré de priorité le plus élevé).
Dans le cas d'opérateurs de priorité égale, de gauche à droite.

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

Bibliographie

✓ Tutoriels

http://www.ccim.be/ccim328/js
http://www.allhtml.com/langages/javascript.php3
http://www.toutjavascript.com/savoir/savoir17.php3

✓ Autres références

ECMAScript : Dernier standard (ISO) : ECMA 262 (2édition - Août 1998) accessible sur http://www.ecma.ch/stand/ecma-262.htm

Info sur les différentes versions de Javascript : télécharger "JavaScript Object Road Map and Compatibility Guide" à l'adresse www.dannyg.com

Références du DHTML
http://msdn.microsoft.com/workshop/author/dhtml/reference/dhtmlrefs.asp

Glossaire

▪ **Client**	Utilisateur du navigateur
▪ **Code**	Lignes d'instructions
▪ **HTML**	HyperText Markup Language
▪ **Balise**	Marqueurs de commandes HTML
▪ **Plug-In**	Fonctionnalité supplémentaire ajoutée à un navigateur (sous la forme d'un module téléchargeable)
▪ **Acrobat**	Logiciel permettant de consulter (Acrobat Reader) ou de créer et modifier (Acrobat) des document au format PDF (Portable Document Format)
▪ **RealAudio**	Plug-In permettant d'écouter un flux de données audio (pour écouter une radio sur le Net, par exemple)
▪ **Evénement**	Action réalisée par l'utilisateur et qui déclenche l'exécution d'une fonction Javascript. Par exemple, sur un clic de souris (événement), ouvrir une fenêtre
▪ **Paramètre**	Information en entrée nécessaire à une fonction pour en assurer le bon fonctionnement
▪ **Expression**	L'association de mots-clé avec des fonctions, des variables et des opérateurs permet de créer des expressions qui seront passées à l'interpréteur Javascript. Tout script contient une ou plusieurs expressions
▪ **Constante**	Valeur immuable généralement associée à un nom
▪ **Variable**	Ensemble de cellules mémoires associées à un nom et permettant de conserver, consulter, modifier une information
▪ **IE**	Internet Explorer
▪ **NN**	Netscape Navigator