PHP/MySQL

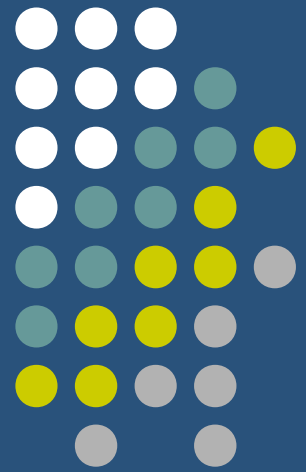
Mise à niveau / Rappels des notions essentielles de Langage Web

HTML , CSS, JavaScript

Chapitre 6

Éric CABRET

email = eric.cabret@gmail.com



CS2I, Concepteur de Système d'Information Informatisé

Groupe CCI Formation Nevers

Bachelor 1

6 – JavaScript

Généralités

- Javascript est un langage de programmation récent, créé par les sociétés Netscape et Sun Microsystems vers la fin de l'année 1995.
- **Son objectif principal :** introduire de l'interactivité avec les pages HTML et effectuer des traitements simples sur le poste de travail de l'utilisateur.
- **Le moyen :** introduire de petits programmes, appelés SCRIPTS, dans les pages HTML.
- Jusqu'ici la programmation ne pouvait être exécutée que sur le serveur. La possibilité d'inclure des programmes dans les pages HTML et de les exécuter directement sur le poste client est intéressante car elle permet de décharger le serveur de ce travail et ... d'éviter les attentes des réponses aux requêtes adressées via le réseau

6 – JavaScript

Caractéristiques

- C'est un langage basé sur des objets, très simple et conçu, en principe, pour des non spécialistes. Ses caractéristiques sont :
  - JS est un langage de programmation **structuré** qui concourt à **enrichir le HTML**, à le rendre plus "intelligent" et interactif.
  - JS contient des **gestionnaires d'événement** : il reconnaît et réagit aux demandes de l'utilisateur, comme un clic de la souris, une validation de formulaire, etc...
  - Le code JS est intégré complètement dans le code HTML et interprété par le navigateur client
  - Ce n'est pas un langage de programmation à part entière, indépendant
  - Sa syntaxe ressemble à Java car elle reprend celle du langage C mais il est en fait très différent. Java est un langage complet, compilé et complètement autonome du langage HTML. Ce n'est pas le cas de JS.
  - Ce n'est pas véritablement un langage orienté objet (pas d'héritage de classe, ni de polymorphisme)

6 – JavaScript

Déclaration 1/4 : interne grâce à <script> et </script> 1/4

- Le code JS sera signalé au navigateur en entourant celui-ci des balises <script> ... </script>

```
<script  type="text/javascript"  language="javascript1.2">
<!--
Code JavaScript
//-->
</script>
```

- On a pris l'habitude d'entourer le code JS des balises `<!--  ...  //-->` pour cacher le code pour les navigateurs qui ne reconnaissent pas le JS.
  - `<!--` et `-->` sont des commentaires HTML
  - `//` est un commentaire JS

6 – JavaScript

Déclaration 1/4 : interne grâce à <script> et </script> 2/4

- Où inclure les balises <script> ... </script> dans le document HTML ?
  - Le code inclus dans la séquence <script> est évalué au début du chargement de la page
    - S'il est inclus dans la section <head>, il n'est pas exécuté tout de suite
    - Par contre, s'il fait partie du corps (<body>) du document, il est immédiatement exécuté en même temps que le code HTML est interprété
  - Il est nécessaire d'inclure les déclarations de fonctions dans le section <head>
    - **IMPORTANT :** Le code JS doit être encodé avant ou au plus tard au moment où il doit être exécuté

6 – JavaScript

Déclaration 1/4 : interne grâce à <script> et </script> 3/4

- Voici un premier exemple de code interne JS

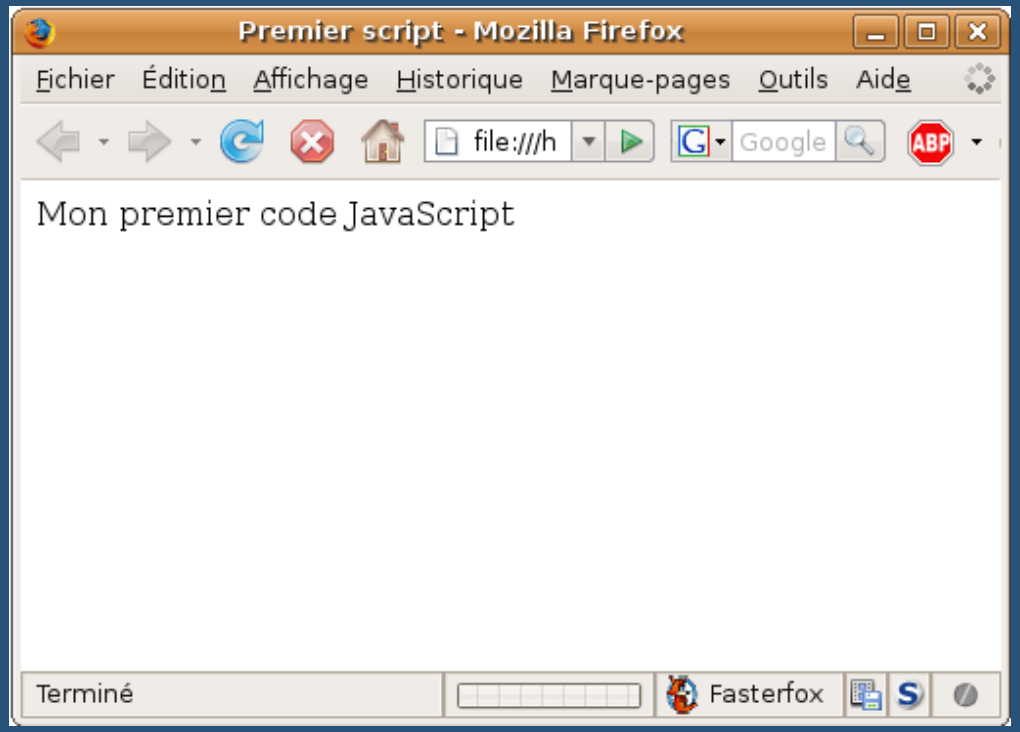
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
   <head>
      <title>Premier script</title>
      <meta http-equiv="Content-Script-Type" content="text/javascript">
   </head>
   <body>
       <script  type="text/javascript"  language="javascript1.2">
       <!--
       document.write("Mon premier code JavaScript");
       //-->
       </script>
    </body>
 </html>
```

  - L'instruction `document.write("Mon premier code JavaScript");` demande d'écrire (write) dans le document le texte saisi entre les parenthèses. Les instructions en JS se termine par un ;

6 – JavaScript

Déclaration 1/4 : interne grâce à <script> et </script> 4/4

- Voici le résultat du code JS de la page précédente :
  - Sur certains navigateurs, il sera nécessaire d'autoriser l'exécution de code JS qui est parfois bloqué par défaut.



6 – JavaScript

Déclaration 2/4 : externe grâce à <script> et </script> 1/2

- Il est possible de noter le code JS dans un ou plusieurs fichiers séparés de la page HTML.
  - Il est ainsi possible d'appeler le même code JS à partir de fichiers HTML sans avoir à le ré-écrire.
  - Le code JS sera placé dans un fichier séparé avec l'extension .js qui contient le code JS sans les balises <script> ... </script>.
  - Le fichier sera appelé dans la page HTML par :

```
<script  type="text/javascript"  language="javascript1.2"
         src="fichier_codeJS_externe.js">
</script>
```

    - Cette déclaration sera placée généralement entre les balises <head></head> ou éventuellement <body></body>

6 – JavaScript

Déclaration 2/4 : externe grâce à <script> et </script> 2/2

- Voici un premier exemple de code externe JS

```
<html>
   <head>
      <title>Script externe</title>
      <meta http-equiv="Content-Script-Type" content="text/javascript">
   </head>
   <body>
       <script  type="text/javascript"  language="javascript1.2"
                src="write.js">
       </script>
    </body>
 </html>
```

Contenu du fichier write.js :

```
document.write("Mon premier JavaScript externe");
```

- Le résultat est identique à l'exemple précédent

6 – JavaScript

Déclaration 3/4 : associé à une balise HTML qui gère un événement 1/3

- Le code JS est généralement inséré sous forme d'un appel de fonction affectée à un gestionnaire d'événement qui apparaît comme un nouvel attribut d'un composant de formulaire
  - **<balise ... onEvenement="code ou fonction JS">**
    - **balise** est le nom de certaines balises particulières (souvent des composants de formulaire. Exemple : INPUT)
    - **onEvenement** est un nouvel attribut de la balise (exemple : **OnClick**)
      - Il faut connaître les différents événements associés à une balise donnée (voir suite du cours)
    - Si « code ou fonction JS » contient une chaîne de caractères, il faut remplacer les " qui entoure cette chaîne par des '
- L'exécution du code est alors provoquée par l'appel d'une fonction JS (préalablement déclarée entre les balises <head></head>) dont l'exécution constitue une réponse à l'événement.
- Un événement survient à l'initiative de l'utilisateur, par exemple en cliquant sur un bouton ou après la saisie du texte dans un champ de formulaire.

6 – JavaScript

Déclaration 3/4 : associé à une balise HTML qui gère un événement 2/3

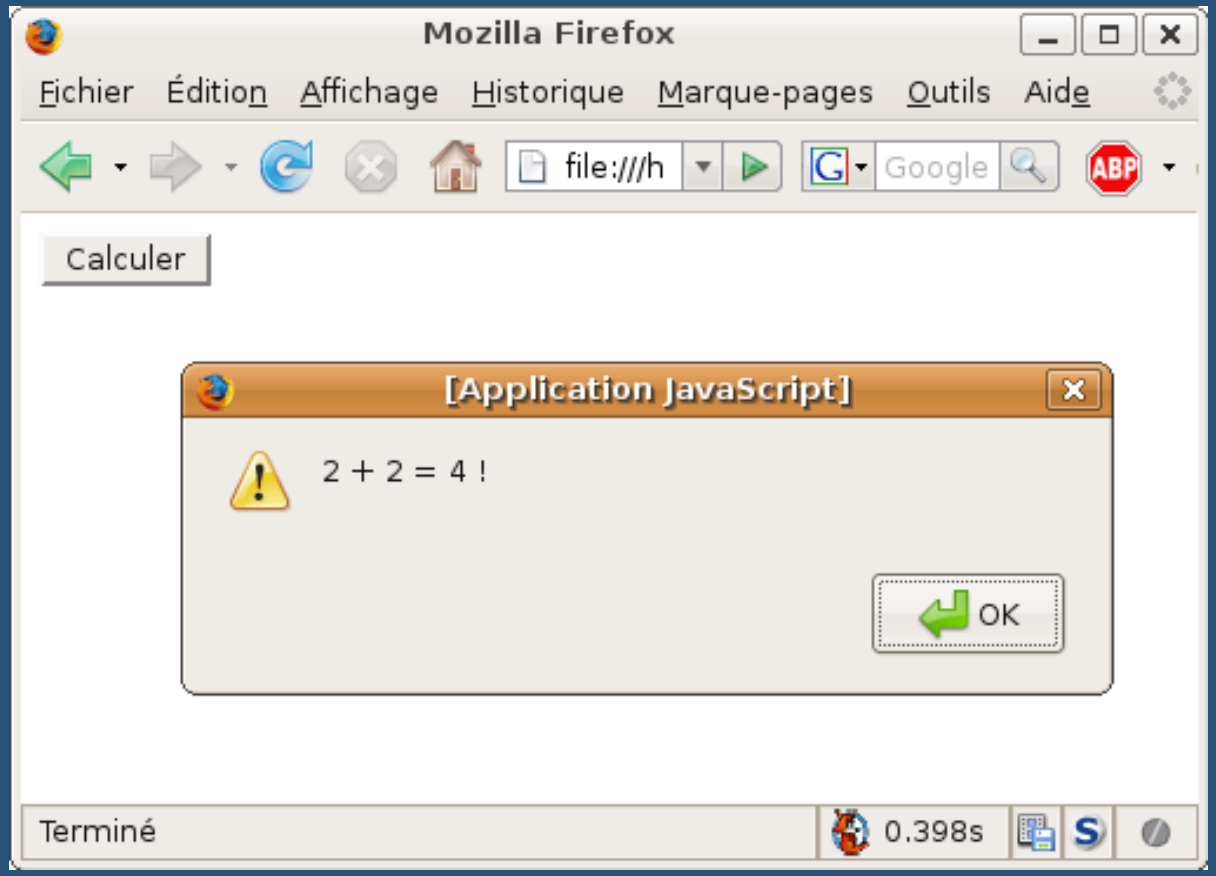
- Exemple :
  - Le code HTML suivant créé un bouton de nom "**bt1**" sur lequel est écrit "**Calculer**". Quand l'utilisateur appuie sur ce bouton, l'événement **onClick** est déclenché et la fonction **calculer()** est appelée.
  - La fonction **alert()** sert à afficher à l'utilisateur des informations simples de type texte. Une fois que ce dernier a lu le message, il doit cliquer sur OK pour faire disparaître la boite de dialogue.

```
<html>
   <head>
      <script language="JavaScript">
         function calculer() {
            alert("2 + 2 = 4 !");
         }
      </script>
   </head>
   <body>
      <form>
      <input type="submit" name="bt1" value="Calculer" onClick="calculer();">
      </form>
   </body>
</html>
```

6 – JavaScript

Déclaration 3/4 : associé à une balise HTML qui gère un événement 3/3

- Voici le résultat du code JS de la page précédente :



6 – JavaScript

Déclaration 4/4 : associé au pseudo-protocole *javascript:* dans une URL 1/3

- La pseudo-URL **"javascript:code ou fonction JS"** permet de lancer l'exécution d'un script écrit en JS, au lieu d'être une requête pour obtenir un nouveau document (comme avec les protocoles habituels http: ftp:)
  - **<A HREF="JavaScript:code ou fonction JS">texte|image activable</A>**
- Pour empêcher que le code ou la fonction appelée dans l'URL JavaScript ne remplace le document courant, on applique l'opérateur **void()** qui neutralise toute valeur ou tout effet de retour :
  - **<A HREF="javascript:void(appel-fct(..))"> ... </A>**

6 – JavaScript

Déclaration 4/4 : associé au pseudo-protocole *javascript:* dans une URL 2/3

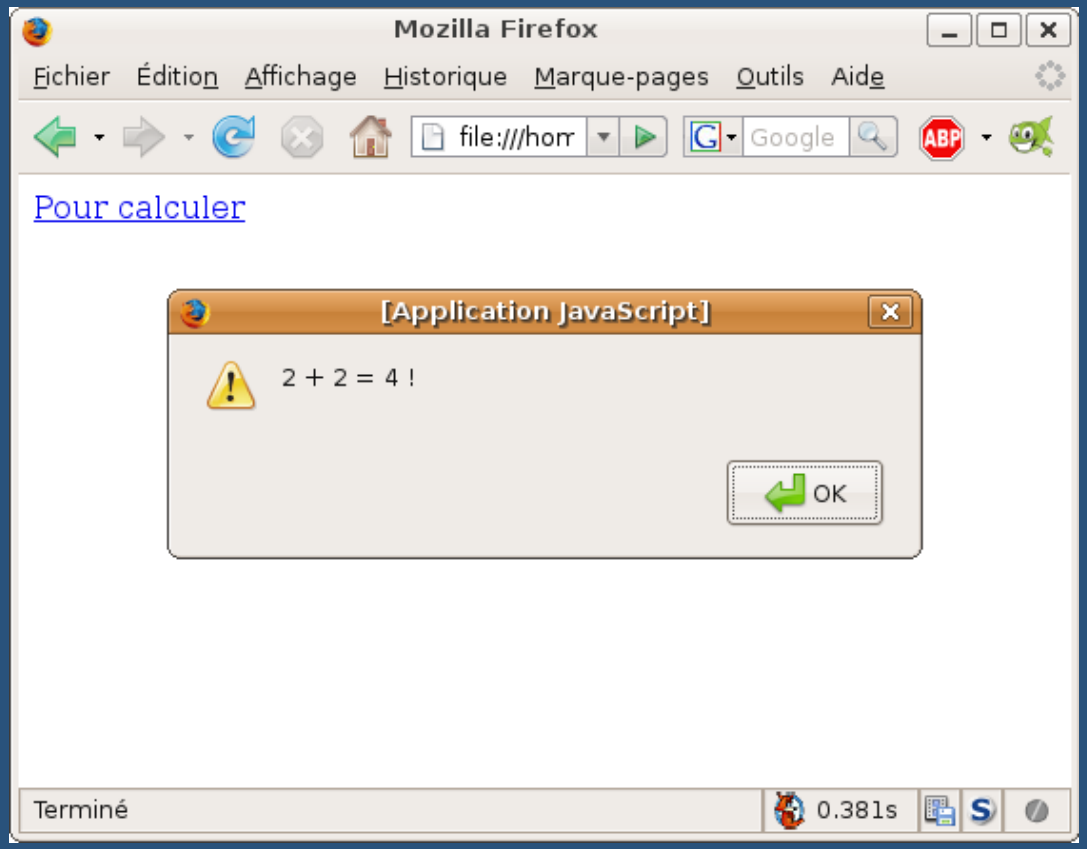
- Exemple :
  - Le code HTML suivant créé un lien dans lequel est écrit "**Calculer**".
  - Quand l'utilisateur appuie sur ce lien, la fonction **calculer()** est appelée.

```
<html>
   <head>
      <script language="JavaScript">
         function calculer() {
            alert("2 + 2 = 4 !");
         }
      </script>
   </head>
   <body>
      ...
      <a href="javascript:void(calculer())">Pour calculer</a>
      ...
   </body>
</html>
```

6 – JavaScript

Déclaration 4/4 : associé au pseudo-protocole *javascript:* dans une URL 3/3

- Voici le résultat du code JS de la page précédente :



6 – JavaScript

Comment déboguer du code JS ?

- Il existe sous Firefox une barre d'outils bourrée de fonctionnalités : le « Web Developer »
  - Disponible à l'adresse suivante http://chrispederick.com/work/web-developer/
  - Il suffit de choisir le menu « Outils – Web Developer – Outils – Console d'erreurs » pour afficher une « console d'erreurs » traçant les erreurs d'exécution de scripts JS (et leurs localisations)
- Le même type d'outil existe aussi sous Internet Explorer.

6 – JavaScript

La structure d'un script en JavaScript

- La syntaxe du langage JS s'appuie sur le modèle de Java et C
- Règles générales :
  - L'insertion des espaces peut s'effectuer n'importe où dans le script
  - Chaque commande doit être terminée par un point-virgule **;**
  - Un nombre à virgule est séparé par un point **.** et non par une virgule
  - Le langage JS est **sensible à la casse**
  - Il existe deux méthodes permettant d'intégrer des commentaires à vos scripts
    - Utiliser // pour une simple ligne de commentaires
    - Utiliser /* ... */ pour les encadrer sur plusieurs lignes

6 – JavaScript
La grammaire de JavaScript

- La grammaire de JavaScript comme d'autres langages de programmation est composée des éléments suivants :
  - Variables (var total)
  - Opérateurs, expressions (total < 200, stotal = total + 10)
  - Instructions (if (total > 200) { … ; })
  - Fonctions (ensemble d'instructions regroupées)
  - Objets (entités caractérisées par des propriétés (**attributs**) et des comportements (**méthodes**))

6 – JavaScript

Les structures de données : les constantes : valeurs possibles 1/2

- Les constantes sont fixées par la valeur indiquée dans le code. Les valeurs reconnues par JS sont :
  - **Les nombres entiers, décimaux ou à virgule flottante**
    - Exemples : 2005 16.6666 2718E-3 (pas d'espace avant E)
  - Les 2 constantes booléennes : **true** ou **false**
  - La constante spéciale **null**
    - C'est la « valeur » d'une variable lorsqu'il n'y a pas de donnée valide dedans
  - La constante spéciale **undefined**
    - C'est la « valeur » d'une variable qui n'a pas été initialisée
  - **Les chaînes de caractères**
    - Elles sont entourées indifféremment de guillemets " " ou d'apostrophes ' ' (à privilégier)

6 – JavaScript

Les structures de données : les constantes : valeurs possibles 2/2

- **Les chaînes de caractères** (suite)
  - Une chaîne commencée par des " doit se terminer par des ". La même règle s'applique pour les '
  - Une chaîne peut être vide (définie par "" ou '')
  - Le signe + permet de concaténer deux chaînes de caractères : exemple : "1 et 1" + "= 2" donne "1 et 1 = 2"
  - Si la chaîne de caractères contient un **"** ou **'** ou **** ou **< >** elle peut être interprétée comme du code HTML ou JS. Pour « protéger » ces caractères spéciaux contre cette interprétation non voulue, il suffit de les faire précéder d'un \
    - Exemple : \"Va à l\'école\"
  - Les caractères spéciaux suivants peuvent aussi être insérés dans les chaînes : **\b** (retour arrière), **\n** (nouvelle ligne), **\r** (touche entrée), **\t** (tabulation), **\f** (formulaire plein)

6 – JavaScript

Les structures de données : les variables : les différents types

- Contrairement à la plupart des autres langages de programmation, JS n'est que faiblement typé : il n'est pas nécessaire de déclarer le type des variables et une variable peut à tout moment changer de type (transtypage).
- On distingue 5 types de variables :
  - Les nombres : **number**
  - Les chaînes : **string**
  - Les booléens : **boolean**
  - Les objets : **object**
  - Les fonctions : **function**
- La fonction **typeof()** appliquée à une variable retourne son type
  - Exemples :
    - typeof(1.0) retourne **number**
    - typeof('Hi') retourne **string**

6 – JavaScript

Les structures de données : les variables : déclaration

- Les variables en JS se déclarent ainsi :
  - **var nom = valeur ;**
    - Le mot **var** est facultatif mais recommandé
    - **= valeur** est facultatif mais fortement recommandé
    - Le type de la variable **nom** est du type de sa **valeur**
- Le **nom** d'une variable soit respecter la syntaxe suivante :
  - **nom** doit débuter par une lettre ou un souligné _
  - **nom** peut comporter un nombre indéterminé de lettres, chiffres ainsi que des caractères _ et $
  - les espaces ne sont pas autorisés
  - **nom** ne doit pas être un mot réservé de JS. Par exemple : on ne peut pas nommer une variable **var**, **true**, **false**, **else** ...
  - pour rappel, JS est sensible aux majuscules et minuscules

6 – JavaScript

Les structures de données : les variables : exemples

- Exemples de déclaration de variables :
  - var pi = 3.1415926535 ;
  - var code_postal = 58000 ;
  - var formulaire1 = "Ville" ;
  - var result$ = true ;
  - Le code suivant affiche : "Mon chiffre préféré est le 7"

```
<html>
   <head>
      <title>Variable</title>
      <meta http-equiv="Content-Script-Type" content="text/javascript">
   </head>
   <body>
      <script  type="text/javascript"  language="javascript1.2">
      var texte = "Mon chiffre préféré est le ";
      var variable = 7;
      document.write(texte + variable);
      </script>
    </body>
</html>
```

6 – JavaScript

Les structures de données : les variables : affectation

- L'affectation d'une variable en JS s'écrit ainsi :
  - **variable = valeur (de l'expression, de même type) ;**
  - Comme en Langage C, **=** est réservé à l'affectation. Le symbole de comparaison d'égalité se note **==** (2 fois **=**)
- Exemple :

```
var  a= 10 ; b= 15;
document.write(" a= "+a + " ;  b= "+b+"<br>");
a = 2 * b - 5 ;   // valeur de a = 2 fois la valeur de b - 5
document.write(" a= "+a + " ;  b= "+b+"<br>");
b = a + b ;       // b = précédente valeur de b + valeur de a
document.write(" a= "+a + " ;  b= "+b);
```

  - donne comme résultat après exécution :

```
a= 10 ; b= 15
a= 25 ; b= 15
a= 25 ; b= 40
```

6 – JavaScript

Les structures de données : les variables : portée

- La portée d'une variable est le domaine où cette variable est connue et utilisable.
- De façon générale les variables définies directement dans une séquence de script (entre <script></script>) ont une portée globale sur toutes les parties de script du fichier HTML
- Exemple :

```
<head>
   <script>var ville="Nevers";
   document.write( "J\'habite "+ville+"<br>");</script>
</head>
<body><script>
   document.write( "J\'habite "+ville+" x2");
</script></body>
```

  - donne comme résultat après exécution :

```
J'habite Nevers
J'habite Nevers x2
```

6 – JavaScript

Les structures de données : construction des expressions 1/3

- On peut distinguer plusieurs types d'expressions :
  - **Expressions arithmétiques** construites par opérations sur les entiers et les réels. Principales opérations :
    - les 4 opérations usuelles : + , - , * , /
    - ++ (incrément) , -- (décrément)
    - % (modulo ou reste par une division entière)
  - **Expressions d'affectation (ou attribution)**
    - l'opérateur d'affectation : variable = expression
    - autres opérateurs d'attribution : += , -= , *= , /= , %= , ++ , --
    - x += 4 équivaut à x = x + 4
    - x ++ équivaut à x = x + 1

6 – JavaScript

Les structures de données : construction des expressions 2/3

- Plusieurs types d'expressions (suite) :
  - **Expressions chaînes de caractères**
    - + opérateur de concaténation (mise bout à bout) de 2 chaînes

```
var aujourdhui = " Lundi " + 3 + " novembre" + 1997;
document.write(aujourdhui + "<br>" );
```

    - += ajoute la chaîne de droite à la chaîne de gauche

```
var message = "Bonjour ";
message += "tout le monde !" ;
document.write(message );
```

      - donne comme résultat d'exécution :

```
Bonjour tout le monde !
```

6 – JavaScript

Les structures de données : construction des expressions 3/3

- Plusieurs types d'expressions (suite) :
  - **Expressions booléennes ou logiques**
    - Les opérateurs de comparaison :
      - **==** (égal, même valeur), **!=** (différent), **>**, **>=**, **<**, **<=**
    - Les opérateurs logiques :
      - **&&** (ET), **||** (OU), **!** (NON) utilisés surtout dans les instructions conditionnelles.
    - Les expressions conditionnelles :
      - **(condition) ? val1 : val2 ;**
        - Évalue la condition et exécute val1 si vrai ou val2 si faux
      - Exemple : message = (fin == false) ? "bonjour" : "au revoir";

6 – JavaScript

Les structures de contrôle : introduction

- Un programme informatique est assemblé à partir de 3 catégories principales d'instructions quelque soit le langage utilisé pour le coder :
  - Les instructions séquentielles
  - Les instructions conditionnelles (ou alternatives)
  - Les instructions itératives (ou répétitives)
- Nous allons voir en parallèle les notations algorithmiques et leur traduction en JS

6 – JavaScript

Les structures de contrôle : la séquence d'instructions

- Il s'agit d'une suite d'actions qui sont exécutées dans l'ordre, les unes après les autres sans choix possibles, ni répétitions.

Séquence ou bloc d'intruction	
algorithme	code JS

début
Instruction 1
Instruction 2
...
fin
	{
	Instruction 1;
	Instruction 2;
	...
	}

6 – JavaScript

Les structures de contrôle : l'instruction conditionnelle if [then] else 1/2

Structure conditionnelle	
algorithme	code JS

SI Condition
ALORS Séquence 1
SINON Séquence 2
FINSI
	if (Condition) {
	  Séquence 1;
	}
	else
	{
	  Séquence 2;
	}

- Remarque :
  - La condition doit toujours être entourée de ( )
  - Le mot alors (then) est toujours sous-entendue.
  - La séquence alors est exécutée si la condition est vraie.
  - La séquence else (optionnelle) est exécutée si la condition est fausse.
  - Les { } ne sont pas obligatoires qu'en cas d'instructions multiples.

6 – JavaScript

Les structures de contrôle : l'instruction conditionnelle if [then] else 2/2

- Exemple :

```
<script type="text/javascript" language="javascript1.2">
var prixHT=150 ; prixTTC=0;
if (prixHT == 0)
   alert("donner un prix HT !");
else
   prixTTC = prixHT * 1.20;
document.write("Prix HT = " + prixHT + "<br>");
document.write("Prix TTC = ", prixTTC, "<br>");
</script>
```

  - donne comme résultat après exécution :

```
Prix HT = 150
Prix TTC = 180
```

6 – JavaScript

Les structures de contrôle : les instructions conditionnelles imbriquées

- Les instructions conditionnelles peuvent être imbriquées :
  - Cela signifie que dans la structure conditionnelle, séquence 1 et/ou séquence 2 peuvent elles-mêmes contenir une structure conditionnelle.
- Exemple :

```
<script type="text/javascript" language="javascript1.2">
var age=0;
age=prompt("Donnez votre âge : ","");
if ( age <= 0  )
   alert("Cela n'a pas de sens !");
else
   if (age <=13)
      alert("Vous êtes encore bien trop jeune ...");
   else
      if (age <18)
         alert("Désolé, vous êtes encore mineur(e)");
      else
         if (age <25)
            alert("Vous êtes déjà majeur(e) !");
         else alert("Ne vous vieillissez donc pas !");
</script>
```

6 – JavaScript

Les structures de contrôle : l'itération contrôlée for (pour) 1/4

- L'instruction « **for** » permet de répéter une séquence d'instructions tant qu'une condition est vraie

Structure itérative FOR	
algorithme	code JS

POUR I de valeur initiale
à valeur finale
REPETER
 	
Séquence d'instructions
 	
FINPOUR
	for (valeur initiale;
	condition;
	poursuite) {
	
	Séquence d'instructions;
	
	}

- La condition qui suit « **for** » est composée de 3 éléments :
  - Une **valeur ou expression initiale** portant sur une variable entière appelée compteur.
  - Une **condition** : tant qu'elle est vraie, la répétition est poursuivie.
  - Une expression de **poursuite** qui consiste en la mise à jour du compteur.

6 – JavaScript

Les structures de contrôle : l'itération contrôlée for (pour) 2/4

- Voici quelques remarques sur l'instruction « **for** » :
  - Les accolades sont facultatives s'il y a qu'une et une seule instruction
  - Chaque instruction se termine par un ;
  - La séquence sera répétée tant que la condition est vraie, compte tenue de la mise à jour du compteur
  - Normalement la mise à jour du compteur doit « rapprocher » de l'arrêt de l'itération sinon les conditions mal conçues peuvent entraîner des « boucles infinies » comme par exemple
    for (i=11; i>10; i++) {...}
  - L'instruction **break** permet de sortir complètement de l'itération en cours
  - L'instruction **continue** sort de la séquence en cours puis reprend à l'itération suivante

6 – JavaScript

Les structures de contrôle : l'itération contrôlée for (pour) 3/4

- Exemple 1 :

```
<script type="text/javascript" language="javascript1.2">
document.write("Table des carrés<br>");
for (var i = 1; i <=15; i++) {
   document.write("i = "+i+"  i<sup>2</sup> = "+i*i+"<br>");
}
</script>
```

  - donne comme résultat après exécution :

```
Table des carrés jusqu'à 15
i = 1 i² = 1
i = 2 i² = 4
i = 3 i² = 9
i = 4 i² = 16
...
i = 14 i² = 196
i = 15 i² = 225
```

6 – JavaScript

Les structures de contrôle : l'itération contrôlée for (pour) 4/4

- Exemple 2 :

```
<script type="text/javascript" language="javascript1.2">
function hasard(N) {
   // renvoie une valeur entière au hasard entre 1 et N inclus
   return Math.floor(Math.random()*N)+1;
}
document.write("Tableau de 10 nombres au hasard<br>");
max = prompt("Nombres au hasard de 1 à ", "10");
tab = new Array(10);
for (var i = 0; i <10; i++)
   tab[i]= hasard(max);
for (var i = 0; i <10; i++)
   document.write("tab [" + i + "] = " + tab[i] + "<br>");
</script>
```

- donne comme résultat après exécution :

```
Tableau de 10 nombres au hasard
tab [0] = 9
tab [1] = 7
tab [2] = 8
...
tab [8] = 10
tab [9] = 8
```

6 – JavaScript

Les structures de contrôle : l'itération while (tant que) 1/2

- L'instruction répétitive « **while** » permet de répéter une séquence d'instructions tant qu'une expression est vraie

algorithme	code JS

TANT QUE (condition vraie)
REPETER
 	
Séquence d'instructions
 	
FINTANTQUE
	while (condition) {
	
	Séquence d'instructions;
	
	}

6 – JavaScript

Les structures de contrôle : l'itération while (tant que) 2/3

- Exemple :

```
<script type="text/javascript" language="javascript1.2">
function hasard(N) { return Math.floor(Math.random()*N)+1; }
max= prompt("Nombres au hasard de 1 à ","6");
document.write("<h2>Tableau de nombres entre 1 et " + max + " tirés au
   hasard, jusqu'à obtenir " + max + "</h2>");
tab = new Array(10*max);
a = hasard(max);  tab[0] = a;
i = -1 ;
while ( a != max ) {
   i ++;
   a = hasard(max);  tab[i]= a;
}
i ++;
document.write(max + ' a été obtenu au ' + i + 'ème tirage <br>') ;
i=0;
while ( tab[i] != null ) {
   document.write("tab[", i,"] = ", tab[i],"---");
   if ((i+1) % 5 == 0) document.write("<br>");
   i ++;
}
</script>
```

6 – JavaScript

Les structures de contrôle : l'itération while (tant que) 3/3

- L'exemple précédent donne comme résultat :

```
Tableau de nombres entre 1 et 6 tirés au hasard, jusqu'à obtenir 6

6 a été obtenu au 7ème tirage

tab[0] = 1---tab[1] = 2---tab[2] = 5---tab[3] = 2---tab[4] = 1---
tab[5] = 4---tab[6] = 6---
```

6 – JavaScript

Les procédures et fonctions : introduction 1/2

- On distingue traditionnellement les **procédures** et les **fonctions**. JavaScript ne différencie pas leur syntaxe. Il est recommandé de les inclure dans la section d'en-tête du document à l'intérieur du couple de balises <head></head>
  - Une **procédure** est une suite d'instructions qui forment un tout et qui sont regroupées sous un même nom.
  - Une **fonction** est une suite d'instructions qui calcule un résultat : celui-ci est transmis à l'expression qui a appelé la fonction, après le mot **return**. A noter que l'instruction **return** peut être utilisée plusieurs fois en cas de valeur retournée conditionnellement.

6 – JavaScript

Les procédures et fonctions : introduction 2/2

- Procédures et fonctions peuvent admettre des **paramètres** :
  - Ce sont des variables dont les valeurs sont fixées par le programme appelant au moment de l'exécution et qui apparaissent « formellement » sans valeur affectée au niveau de la déclaration.
  - S'il n'y a pas besoin de paramètres, le nom de la fonction est suivi d'un couple de parenthèses vides.

6 – JavaScript

Les procédures et fonctions : déclaration

- La déclaration générale d'une procédure et d'une fonction suit la syntaxe suivante :

```
<head>
<script type="text/javascript" language="javascript1.2">

function nomProcédure (param1, param2, ...)  {
   Séquence d'instructions;
}

function nomFonction (param1, param2, ...)   {
   Séquence d'instructions;
   return nom_variable;
}

</script>
</head>
```

6 – JavaScript

Les procédures et fonctions : appel

- JS lit les fonctions présentes dans une page lors de son ouverture **mais ne les exécutent pas**.
- **Une fonction n'est exécutée qu'au moment de son appel**.
- La syntaxe d'appel d'une procédure et d'une fonction est :

```
nomProcédure(valeur1, valeur2, ...) ;
variable = nomFonction(valeur1, valeur2, ...) ;
```

- Dans l'écriture de l'appel de la procédure ou de la fonction, il faut fournir une liste de valeurs correspondant exactement à la liste des paramètres présents dans la déclaration.
- Les procédures forment des instructions à part entière tandis que les fonctions doivent être affectées à une variable du type de retour ou incluses dans des expressions comme document.write(...).

6 – JavaScript

Les procédures et fonctions : exemple

- Exemple :

```
<head>
   <script type="text/javascript" language="javascript1.2">
      function bonjour(prenom) {     // Déclaration de procédure
         document.write("Bonjour, comment vas-tu ", prenom," ?<br>");
      }
      function volumeSphere(rayon) { // Déclaration de fonctions
         return 4/3*Math.PI*Math.pow(rayon,3);
      }
      function calculerPrix(PrixUnitaire, NbArticles)  {
         return  PrixUnitaire* NbArticles;
      }
   </script>
</head><body>
   <script type="text/javascript" language="javascript1.2">
      // appel de la procédure
      bonjour("Toto") ;
      //appels des fonctions
      var montant=calculerPrix( 150 , 4) ;
      document.write( "Tu dois payer  "+ calculerPrix(150, 4)+ " euros.<br>");
      document.write( "Le volume de la sphère de rayon unité est "+
                      volumeSphere(1) + "<br>" );
   </script>
</body>
```

6 – JavaScript

Les procédures et fonctions : visibilité des paramètres 1/2

- De façon générale, les paramètres formels d'une fonction ne sont connus qu'à l'intérieur de la fonction.
- Il en est de même **des variables locales**, variables qui sont explicitement déclarées à l'intérieur de la fonction.

- **Conséquences :**
- Si la valeur d'un paramètre ou d'une variable locale est modifiée **dans** la fonction, cette modification ne sera pas connue à l'extérieur de la fonction. On dit que cette variable n'est pas visible au niveau du programme général.

6 – JavaScript

Les procédures et fonctions : visibilité des paramètres 2/2

- Exemple : qu'obtient-on exactement à l'exécution ? :

```
<head>
   <script type="text/javascript" language="javascript1.2">
      function bonjour(nom) {    // nom est un paramètre local
         var ch ="Salut !";      // ch est une variable locale
         document.write("Au début de la fonction :  Bonjour "+nom +"<br>");
         nom = "Alain";              // on modifie le paramètre local
         document.write("A la fin de la fonction : Bonjour "+nom +"<br>");
      }
      var prenom = "Jacques";
      bonjour(prenom) ;
      document.write("Après appel de la fonction : Bonjour  "+prenom
   +"<br>");
   </script>
</head>
```

- Le résultat d'exécution vaut :

```
Au début de la fonction : Bonjour Jacques
A la fin de la fonction : Bonjour Alain
Après appel de la fonction : Bonjour Jacques
```

6 – JavaScript

Les procédures et fonctions : méthodes JS : introduction

- Les méthodes JS sont des fonctions déjà implantées dans le langage JS et dédiées à un objet particulier.
- Nous avons jusqu'ici utilisé la méthode write() spécifique à l'objet document.
- Passons en revue quelques autres méthodes :
  - alert()
  - confirm()
  - prompt()
  - setTimeout() / clearTimeout()

6 – JavaScript

Les procédures et fonctions : méthodes JS : alert() : définition

- La méthode **alert()** de l'objet fenêtre affiche une boîte de dialogue d'avertissement :
  - Qui comporte un message qui reproduit la valeur (variable et/ou chaîne de caractères) de l'argument qui lui est fourni
  - Qui bloque le programme en cours tant que l'utilisateur n'a pas cliqué sur OK pour fermer celle-ci
- Sa syntaxe est :
  - **alert(variable);**
  - **alert("chaîne de caractères");** (utiliser \n pour afficher plusieurs lignes)
  - **alert(variable + "chaîne de caractères");**
- **alert()** est peu utilisée sur un site Web. Par contre, elle est très utile pour vous aider à tester un script JS ou pour trouver d'éventuelles erreurs de programmation.

6 – JavaScript

Les procédures et fonctions : méthodes JS : alert() : exemple

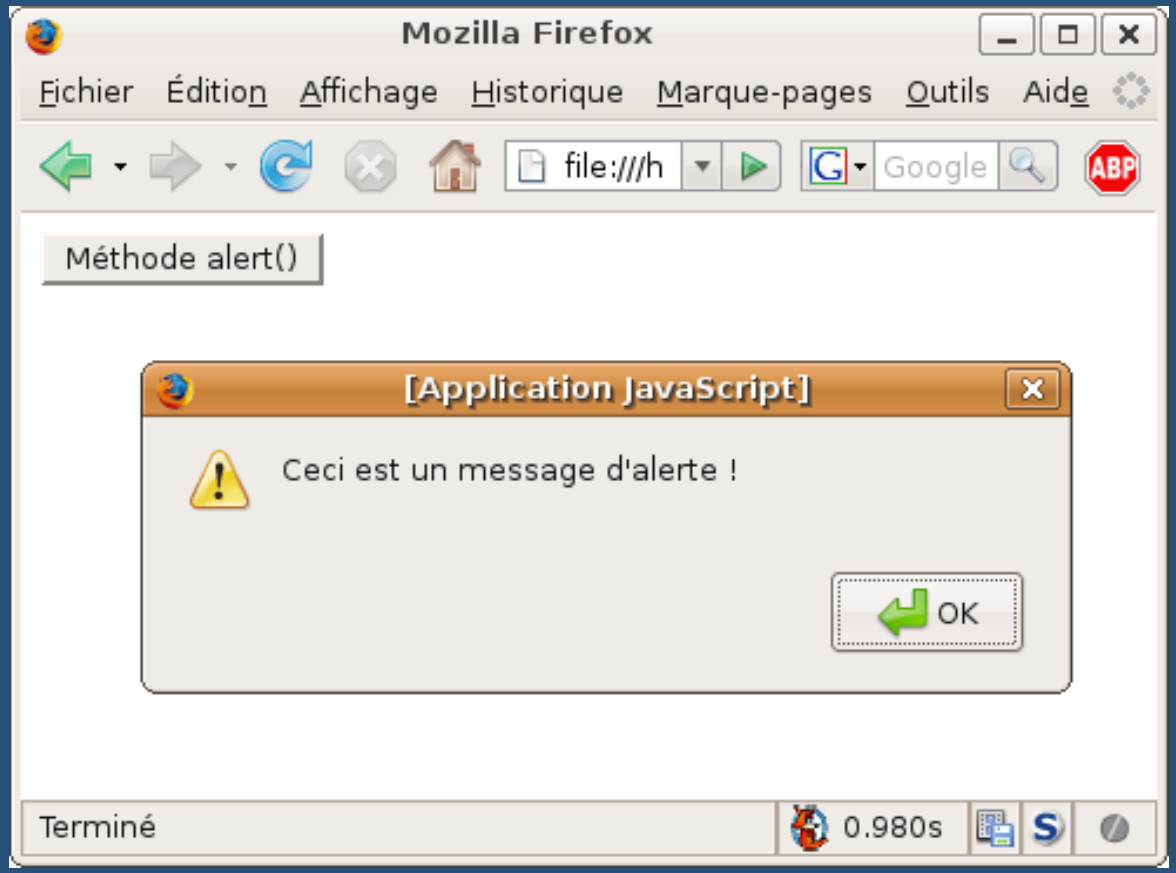
- Exemple :
- Une boîte d'alerte va se déclencher lorsque le bouton est cliqué :

```
<html>
<head>
   <script type="text/javascript" language="javascript1.2">
      function alerte() {
         alert("Ceci est un message d'alerte !");
      }
   </script>
</head>
<body>
   <form>
      <input  type=button  value="Méthode alert()"  onclick="alerte()">
   </form>
</body>
</html>
```

6 – JavaScript

Les procédures et fonctions : méthodes JS : alert() : résultat

- Le résultat d'exécution affichera :



6 – JavaScript

Les procédures et fonctions : méthodes JS : confirm() : définition

- La méthode **confirm()** de l'objet fenêtre affiche une boîte de dialogue comportant deux boutons : **OK** et **Annuler**
  - En cliquant sur **OK**, la méthode retourne la valeur **true** et bien entendu **false** si on a cliqué sur **Annuler**.
  - Ceci peut permettre, par exemple, de choisir une option dans un programme.

- Sa syntaxe est :
  - **var reponse = confirm("chaîne de caractères");**

6 – JavaScript

Les procédures et fonctions : méthodes JS : confirm() : exemple

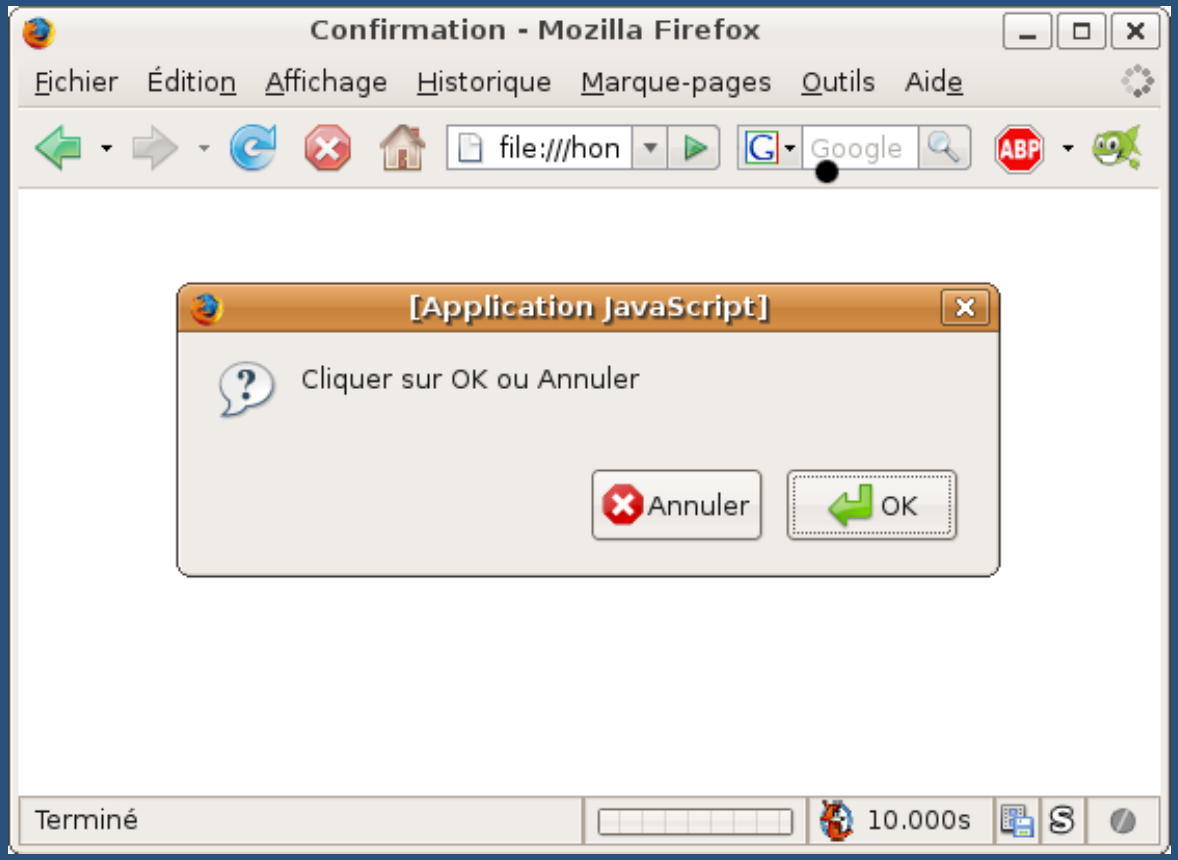
- Exemple :
- Le script lance une boîte de confirmation. La valeur renvoyée conditionne l'affichage d'une alerte correspondante :

```
<html>
<head> <title>Confirmation</title> </head>
<body>
   <script type="text/javascript" language="javascript1.2">
      var a = confirm("Cliquer sur OK ou Annuler");
      if (a == true) {
         alert("Vous avez cliqué sur OK");
      } else {
         alert("Vous avez cliqué sur Annuler");
      }
   </script>
</body>
</html>
```

6 – JavaScript

Les procédures et fonctions : méthodes JS : confirm() : résultat

- Le résultat d'exécution affichera :



6 – JavaScript

Les procédures et fonctions : méthodes JS : prompt() : définition

- La méthode **prompt()** de l'objet fenêtre affiche une boîte de dialogue appelé « boîte d'invite » :
  - Composée d'un champ comportant une entrée à compléter par l'utilisateur
  - Cette entrée peut aussi posséder une valeur par défaut
  - En cliquant sur **OK**, la méthode renvoie la valeur saisie par l'utilisateur ou la valeur par défaut.
  - Si l'utilisateur clique sur **Annuler**, la valeur **null** est renvoyée.
- Sa syntaxe est :
  - **var reponse = prompt("texte de la boîte d'invite",**
    "valeur par défaut");

6 – JavaScript

Les procédures et fonctions : méthodes JS : prompt() : exemple

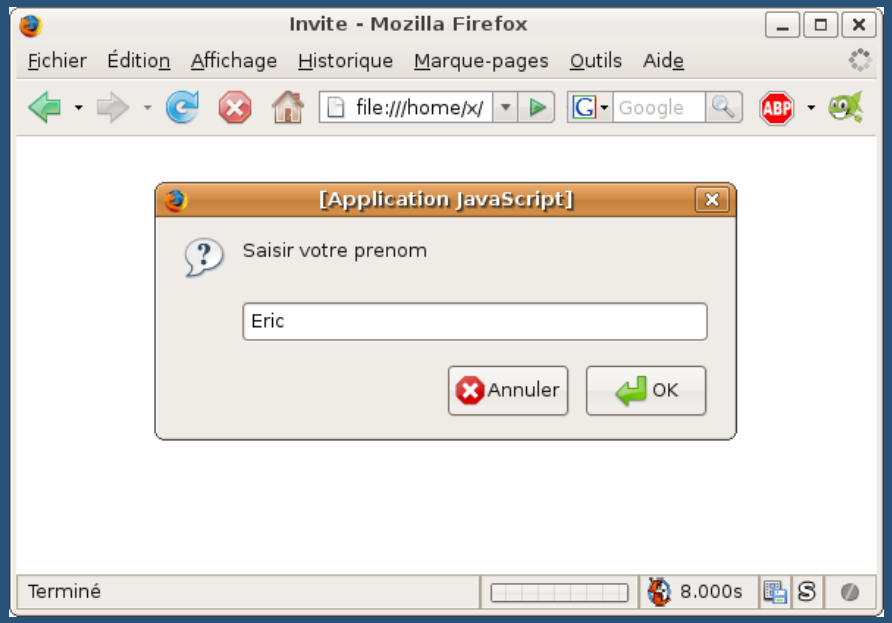
- Exemple :
- Le script demande le prénom du visiteur par une boîte d'invite et l'affiche sur la page Web :

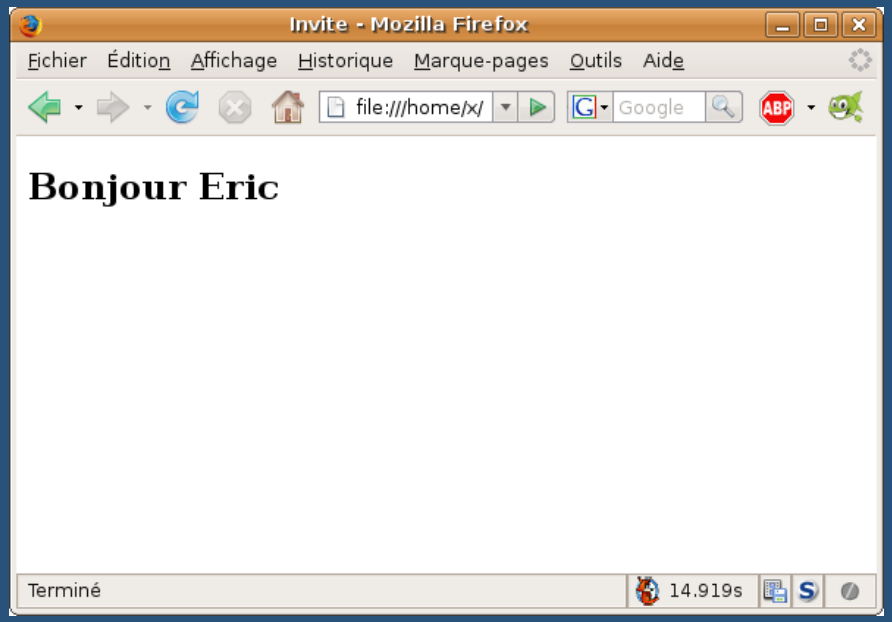
```
<html>
<head> <title>Invite</title> </head>
<body>
   <script type="text/javascript" language="javascript1.2">
      var prenom = prompt("Saisir votre prenom", "Votre prenom ici");
      document.write("<h2>Bonjour " + prenom + "</h2>");
   </script>
</body>
</html>
```

6 – JavaScript

Les procédures et fonctions : méthodes JS : prompt() : résultat

- Le résultat d'exécution affichera :





6 – JavaScript

Les procédures et fonctions : alert(), confirm(), prompt() et les accents

- Les accents dans les boîtes de dialogue **alert()**, **confirm()** et **prompt()** ne s'affichent pas correctement. La solution consiste à remplacer les caractères accentués par leur équivalent en octal précédé de \ (anti-slash).
- Par exemple :
  - à : \340
  - è : \350
  - é : \351
  - ê : \352

6 – JavaScript

Les procédures et fonctions : méthodes JS : setTimeout()/clearTimeout() 1/2

- La méthode **setTimeout()** (de l'objet Window) déclenche une minuterie, un compte à rebours au bout duquel une expression JS passée en 1er paramètre sera exécutée. Le temps d'attente est passé en second paramètre.
  - **Syntaxe : TimeoutID=setTimeout(fonction, ms)**
    - **fonction** est le nom de la fonction appelée à l'issue du délai. Cette action n'est exécutée qu'une seule fois.
    - **ms** est une valeur numérique entière, exprimée en millisecondes.
    - **TimeoutID** est un identificateur qui est utilisé seulement pour annuler l'évaluation avec la méthode **clearTimeout()**.
- La méthode **clearTimeout(TimeoutID)** annule la minuterie avant sa fin. Elle est surtout utilisée pour arrêter des appels récursifs d'une fonction passée dans setTimeout()

6 – JavaScript

Les procédures et fonctions : méthodes JS : setTimeout()/clearTimeout() 2/2

- L'exemple suivant ajoute indéfiniment 1 à la variable x et affiche une alerte (alert()) toutes les 3 secondes (3000 ms). Quand le bouton stop est cliqué, clearTimeout() est appelé pour arrêter l'affichage régulier de l'alerte :

```
<script type="text/javascript" language="javascript1.2">
   var x = 0; var timer = 0;

   function testertimeout(){
      x = x+1;
      alert(" value of x is - "+x);
      timer = setTimeout(testertimeout,3000);
   }
   function stoppertimeout() {
      clearTimeout(timer);
   }
</script>

<form name="timeout">
   <input type="button" onClick="testertimeout()"  value="start">
   <input type="button" onClick="stoppertimeout()" value="stop">
</form>
```

6 – JavaScript

Les classes et objets en JS : généralités 1/3

- JS n'est pas un vrai langage orienté-objet. Des concepts primordiaux ne sont pas implémentés, notamment l'héritage ou le polymorphisme. Mais il peut constituer une introduction à la syntaxe objet.

- Dans le présent chapitre, nous prendrons connaissance de quelques classes de données prédéfinies.

- Ensuite, nous étudierons les objets du navigateur

6 – JavaScript

Les classes et objets en JS : généralités 2/3

- Une **classe** d'objets est un ensemble d'informations regroupés sous un même nom qui décrivent la structure et le comportement commun de ces objets.

- Pour définir une **classe**, il faut préciser :
  - Ses **propriétés**. Ce sont des variables attachées à un type d'objets et qui contiennent une de ses caractéristiques.
  - Ses **méthodes**. Ce sont des procédures ou fonctions qui décrivent ses comportements et ses actions.

6 – JavaScript

Les classes et objets en JS : généralités 3/3

- **Construction des objets :**
  - Pour obtenir un **objet**, appelé aussi instance de la classe, on utilise une fonction spéciale appelée **constructeur** et qui porte le nom de la classe.
  - Un objet est alors créé par l'instruction **new** :
    - **var objet = new Classe();**
- **Utilisation des objets :**
  - Propriétés et méthodes constitutives d'une classe ne sont alors applicables qu'à un objet de cette classe.
    - **objet.propriété**
    - **objet.méthode()**

6 – JavaScript

Les objets prédéfinis : introduction

- Elles sont définies dans JS, accompagnées de données (**propriétés**) et de fonctions (**méthodes**) utilisables directement par le programmeur.
- Nous allons parcourir 4 classes d'objets principales :
  - Math, String, Array, Date
  - afin de savoir construire des objets de ces classes et utiliser leurs propriétés et leurs méthodes.

6 – JavaScript

Les objets prédéfinis : l'objet Math : introduction

- Les constantes et fonctions mathématiques usuelles doivent être préfixées par le nom de l'objet Math desquelles elles dépendent.
  - Ce sont les « propriétés » et « méthodes » de calcul de l'objet Math.
- Par exemple :
  - **Math.PI** désigne une propriété de l'objet Math : le nombre PI
  - **Math.sin(1.50)** appelle une méthode de l'objet Math et calcule sin(1.50), l'angle étant exprimé en radians.

6 – JavaScript

Les objets prédéfinis : l'objet Math : principales méthodes

- Liste des principales méthodes :
  - **Math.sqrt()** : racine carrée
  - **Math.log() , Math.exp() , Math.abs() ,Math.cos (), Math.sin() , Math.tan()**
  - **Math.floor(), Math.ceil()** : entier immédiatement inférieur / supérieur.
  - **Math.pow(base, exposant) :** fonction puissance où base et exposant sont des expressions numériques quelconques évaluables
  - **Math.max() , Math.min()**
  - **Math.random() :** nombre "réel" choisi au hasard dans [0 , 1[
  - **Math.round()** : arrondit à l'entier le plus proche

6 – JavaScript

Les objets prédéfinis : l'objet String : déclaration

- **Syntaxe :**
  - **var nom = "chaîne de caractères" ;**
    - Créé une variable nommée **nom** et lui attribue :
      - le type **String**
      - la valeur **"chaîne de caractères"**
- **Exemple :**

```
var chaine = "<B>Bonjour !</B>";
document.write ("La longueur de la chaine ",chaine, " est : ",
chaine.length,". Pourquoi ?<br>");
```

  - **donne comme résultat :**

```
La longueur de la chaine Bonjour ! est : 16. Pourquoi ?
```

6 – JavaScript

Les objets prédéfinis : l'objet String : propriétés

- La valeur peut donc être composée d'une suite de caractères quelconques, y compris des balises HTML
- Des caractères spéciaux peuvent aussi être insérés dans les chaînes : **\b** (retour arrière), **\f** (saut de page), **\n** (nouvelle ligne), **\r** (Entrée), **\t** (tabulation), **\'** pour une apostrophe
- L'unique propriété **length** permet d'obtenir la longueur de la chaîne
- L'opération + concatène (mise à la suite) 2 chaînes pour en former une nouvelle unique

6 – JavaScript

Les objets prédéfinis : l'objet String : méthodes 1/4

- **eval()** : évalue numériquement une expression arithmétique fournie sous forme de chaîne de caractères.
- **parseInt()** : donne un nombre entier résultant de la conversion (si possible) d'une chaîne de caractères.
  - Si la conversion n'est pas possible, la valeur renvoyée est 0
- **parseFloat()** : donne un nombre décimal de la même façon.
- **toString(base)** : convertit l'objet (un nombre généralement) en une chaîne représentant le nombre écrit dans la base indiquée.

6 – JavaScript

Les objets prédéfinis : l'objet String : méthodes 1/4 : exemple

- **Exemple :**

```
var a = 5 + "007";
var b = 5 + parseInt("007");
var c = 2 + parseFloat("1.1416");
var d=255;
document.write(" a = ", a, "<br> b = ", b, "<br> c = ", c,
   "<br>");
for (var i=0; i<255 ; i++)
{
  document.write("En hexadécimal d = "+i+ " s'écrit "+
   i.toString(16)+"<br>");
}
```

- Résultat obtenu :

```
a = 5007
b = 12
C = 3.1416
...
En hexadécimal d = 128 s'écrit 80
...
```

6 – JavaScript

Les objets prédéfinis : l'objet String : méthodes 2/4

- **chaine.toLowerCase()** : met chaîne en minuscule
- **chaine.toUpperCase()** : met chaîne en majuscule
  - Exemple :

```
var chaine = "Bonjour !";
chaine = chaine.toUpperCase();
```

- **chaine.substring(d, l)** : extrait une partie de chaîne à partir du caractère de position d jusqu'à l-1 (d débute à 0)
  - Exemple :

```
var chaine = "aujourd\'hui";
document.write("chaine.substring(2,6) = ", chaine.substring(2,6));
```

  - résultat : `chaine.substring(2,6) = jour`

6 – JavaScript

Les objets prédéfinis : l'objet String : méthodes 3/4

- **chaine.charAt(n)** : donne le caractère placé en nième position (n de 0 à chaine.length-1)

  - Exemple :

```
var chaine = "informatique";
document.write("J'épelle : ");
for (i=0; i<chaine.length; i++) document.write (chaine.charAt(i), "-");
```

  - résultat : `J'épelle : i-n-f-o-r-m-a-t-i-q-u-e-`

6 – JavaScript

Les objets prédéfinis : l'objet String : méthodes 4/4

- **chaine.indexOf(s_ch)** : donne la 1ère position de la chaîne de caractères égale à s_ch (débute en 0).
  - Retourne -1 si la recherche est infructueuse
  - Il est possible de transmettre comme second paramètre la position à partir de laquelle la recherche doit commencer
  - Exemple :

```
var chaine = "informatique";
var s_ch = "ma";
var car = "i";
var position = 2;
document.write ("1ère position de ", s_ch, " dans ", chaine," est : ",
chaine.indexOf( s_ch),"<br>");
document.write ("position de ", car ," dans ", chaine," à partir de la
position ", position," est : ", chaine.indexOf(car, position), "<br>");
```

  - résultat :

```
1ère position de ma dans informatique est : 5
position de i dans informatique à partir de la position 2 est : 8
```

6 – JavaScript

Les objets prédéfinis : l'objet Array : introduction

- Un tableau (Array en anglais) dans un langage de programmation n'a rien à voir avec un tableau HTML ou Word !
- C'est un ensemble de données, en général de même type (chaîne de caractères, nombres) rangées sous un même nom et distingués par un numéro.
- Ce numéro, l'indice, est placé entre [ ] et caractérise un élément du tableau.

6 – JavaScript

Les objets prédéfinis : l'objet Array : déclaration

- La syntaxe de déclaration d'un tableau est :
  - **var nom_tableau = new Array(dimension) ;**
    - Le mot **new** commande la construction d'un objet de type **Array**, c'est-à-dire tableau.
    - Le paramètre dimension, s'il est présent, est le nombre d'éléments dans le tableau
- Exemples :
  - var MonTableau = new Array(8);
    - construit un tableau nommé MonTableau, de taille 8 éléments.
  - var Les4saisons = new Array("printemps", "été", "automne", "hiver");
    - construit un tableau en initialisant 4 éléments, c'est-à-dire en leur affectant une valeur initiale (qui pourra ensuite être modifiée).
  - var mois= new Array(12);

6 – JavaScript

Les objets prédéfinis : l'objet Array : utilisation

- **Les éléments d'un tableau de taille dim sont indicés à partir de 0 jusqu'à dim - 1.**
- Exemples :
- ```
  var mois= new Array(12);
  mois[0]="Janvier"; ...mois[11]="Décembre";
  ```

- ```
  var Les4saisons = new Array("printemps","été","automne","hiver");
  document.write("Voici les 4 saisons : <ul>")
  for (i=0; i<4; i++) document.write("<li>"+Les4saisons[i]+"</li>");
  document.write("</ul>");
  ```

  - donne comme résultat :

```
Voici les 4 saisons :
```

- printemps
- été
- automne
- hiver

6 – JavaScript

Les objets prédéfinis : l'objet Array : les tableaux associatifs

- Les tableaux associatifs sont des tableaux dont l'indice est une chaîne de caractères (et non plus un nombre). La chaîne est considérée comme la clé pour l'accès aux éléments du tableau.
- Exemple :

```
var moteur = new Array("Google", "Yahoo", "Voila");
moteur["Google"] = "http://www.google.fr";
moteur["Yahoo"]  = "http://www.yahoo.fr";
moteur["Voila"]  = "http://www.voila.fr";
```

- Noter la présence des guillemets entre les crochets, ce qui est la règle pour les chaînes de caractères.

6 – JavaScript

Les objets prédéfinis : l'objet Array : les tableaux à plusieurs dimensions

- Les tableaux à plusieurs dimensions doivent être gérés comme des tableaux de tableaux à une dimension
- Exemple et exécution :

```
tab=new Array(3);
tab[0]= new Array(1,2,3);
tab[1]= new Array(4,5,6);
tab[2]= new Array(7,8,9);
for (i=0;i<3;i++)  {
  for (j=0;j<3;j++)
     document.write(tab[i][j],"  ");
  document.write("<br>");
}
```

- donnera comme résultat :

```
1  2  3
4  5  6
7  8  9
```

6 – JavaScript

Les objets prédéfinis : l'objet Array : propriété et méthodes

- **Propriété :**
  - **length** : donne le nombre d'éléments d'un tableau
    - Exemple :

```
var mois= new Array(12);
document.write("Il y a " + mois.length + "mois dans l'année");
var NbTrim = mois.length / 3;
document.write(" partagés en " + NbTrim + " trimestres");
```

- **Méthodes :**
  - **reverse()** : change l'ordre des éléments (mais ne les trie pas)
  - **sort()** : trie les éléments par ordre alphabétique (à condition qu'ils soient de même nature)

6 – JavaScript

Les objets prédéfinis : l'objet Array : exemple de tri 1/2

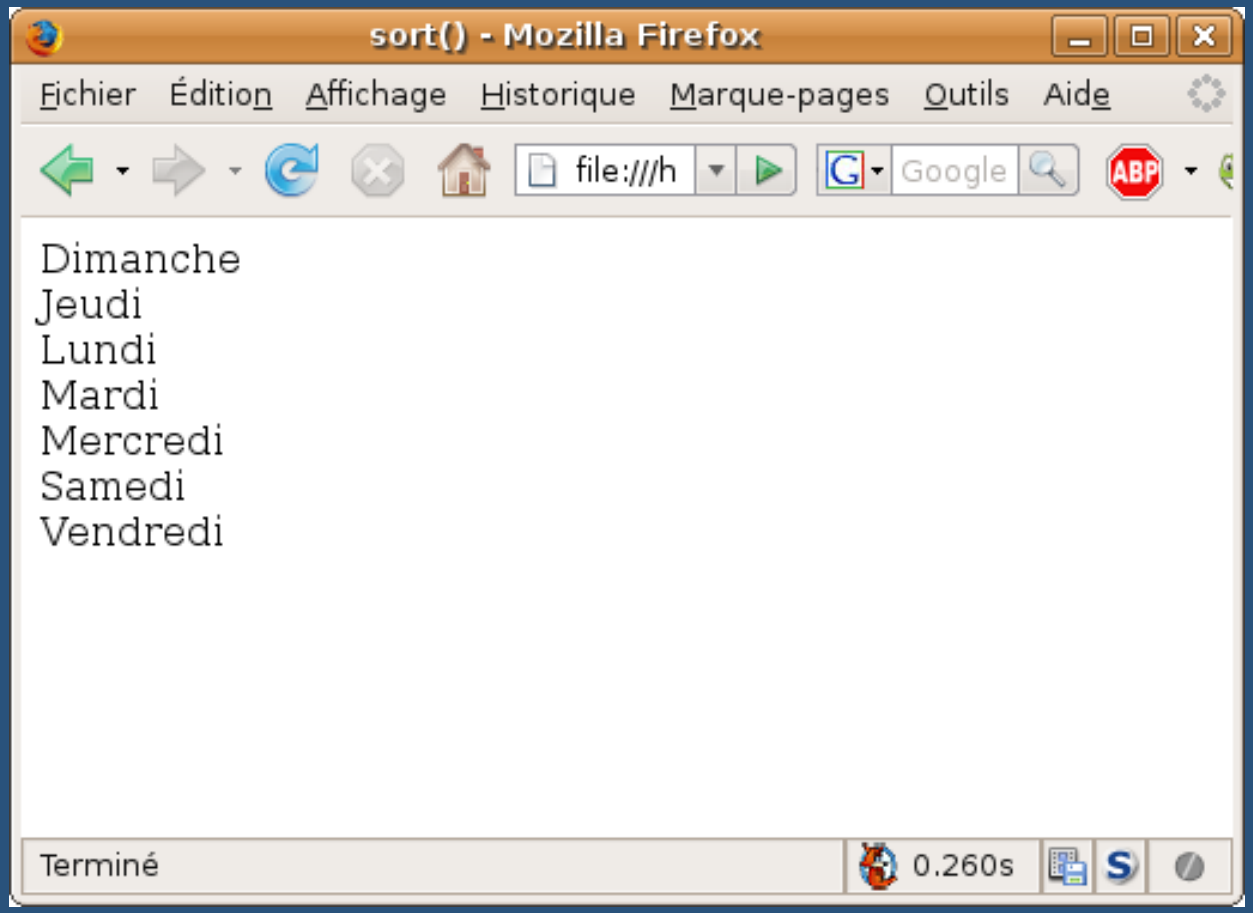
- Exemple utilisant la méthode sort() :

```
<html>
<head>
   <title>sort()</title>
</head>
<body>
   <script type="text/javascript" language="javascript1.2">
      var semaine = new Array(7);
      semaine[0] = "Lundi" ;
      semaine[1] = "Mardi" ;
      semaine[2] = "Mercredi" ;
      semaine[3] = "Jeudi" ;
      semaine[4] = "Vendredi" ;
      semaine[5] = "Samedi" ;
      semaine[6] = "Dimanche" ;
      semaine.sort();
      for (i=0; i<7; i++)  document.write(semaine[i] + "<br>");
   </script>
</body>
</html>
```

6 – JavaScript

Les objets prédéfinis : l'objet Array : exemple de tri 2/2

- Le résultat d'exécution affichera :



6 – JavaScript

Les objets prédéfinis : l'objet Date : construction 1/2

- L'objet Date permet de définir et gérer les dates et les heures. L'origine des dates a été choisie le 1er janvier 1900 et est exprimée en milli-secondes.
- **Construction d'un objet de type Date :**
  - Pour construire un objet de type Date, il faut utiliser un constructeur Date() avec le mot-clé new
  - **variable = new Date(liste de paramètres)**
- **Attention :** les secondes et les minutes sont notées de 0 à 59, les jours de la semaine de 0 (dimanche) à 6, les jours du mois de 1 à 31, les mois de 0 (janvier) à 11 et les années sont décomptées depuis 1900 .

6 – JavaScript

Les objets prédéfinis : l'objet Date : construction 2/2

- On peut passer différents paramètres pour construire divers objets date :
  - **Date()** : pour obtenir la date et l'heure courante (connue du système)
  - **Date("month day, year hour:min:sec")**
    - Par exemple : Date("December 25, 1995 13:30:00")
  - **Date(année, mois, jour)** : une suite convenable de 3 entiers.
    - Par exemple : Date(2000, 0, 1)
  - **Date(année, mois, jour, heures, minutes, secondes)** : une suite de 6 entiers.
    - Par exemple : Date(1995, 11, 25, 13, 30, 00)

6 – JavaScript

Les objets prédéfinis : l'objet Date : méthodes

- Elles permettent d'extraire diverses informations d'un objet date :
  - **set....()** : pour transformer des entiers en Date
  - **get....()** : pour transformer en date et heure des objets Date
    - Après les préfixes set et get, on peut mettre **Year**, **Month**, **Date**, **Hours**, **Minutes**, **Seconds**
    - Pour obtenir respectivement : nombre d'années depuis 1900, le n° du mois (0 pour janvier), le n° du jour dans le mois et les heures, minutes et secondes.
    - **getDay()** donne le n° du jour de la semaine (0 pour dimanche)
    - **getTime()** donne le nombre de milli-secondes écoulées depuis le 1/1/1970. Ceci est très pratique pour calculer des intervalles entre 2 dates.

6 – JavaScript

Les objets prédéfinis : l'objet Date : méthodes : exemple

- Exemple :

```
var aujourdhui= new Date();
var maDate = new Date ("November 24, 1981");
var jour = maDate.getDate ()          // jour vaut  24.
document.write("Nous étions le  ", jour, "/",
maDate.getMonth()+1, "/", maDate.getYear()+1900 , "<br>");
document.write("Nous sommes le ", aujourdhui.getDate(), "/",
aujourdhui.getMonth()+1, "/", aujourdhui.getYear()+1900 );
```

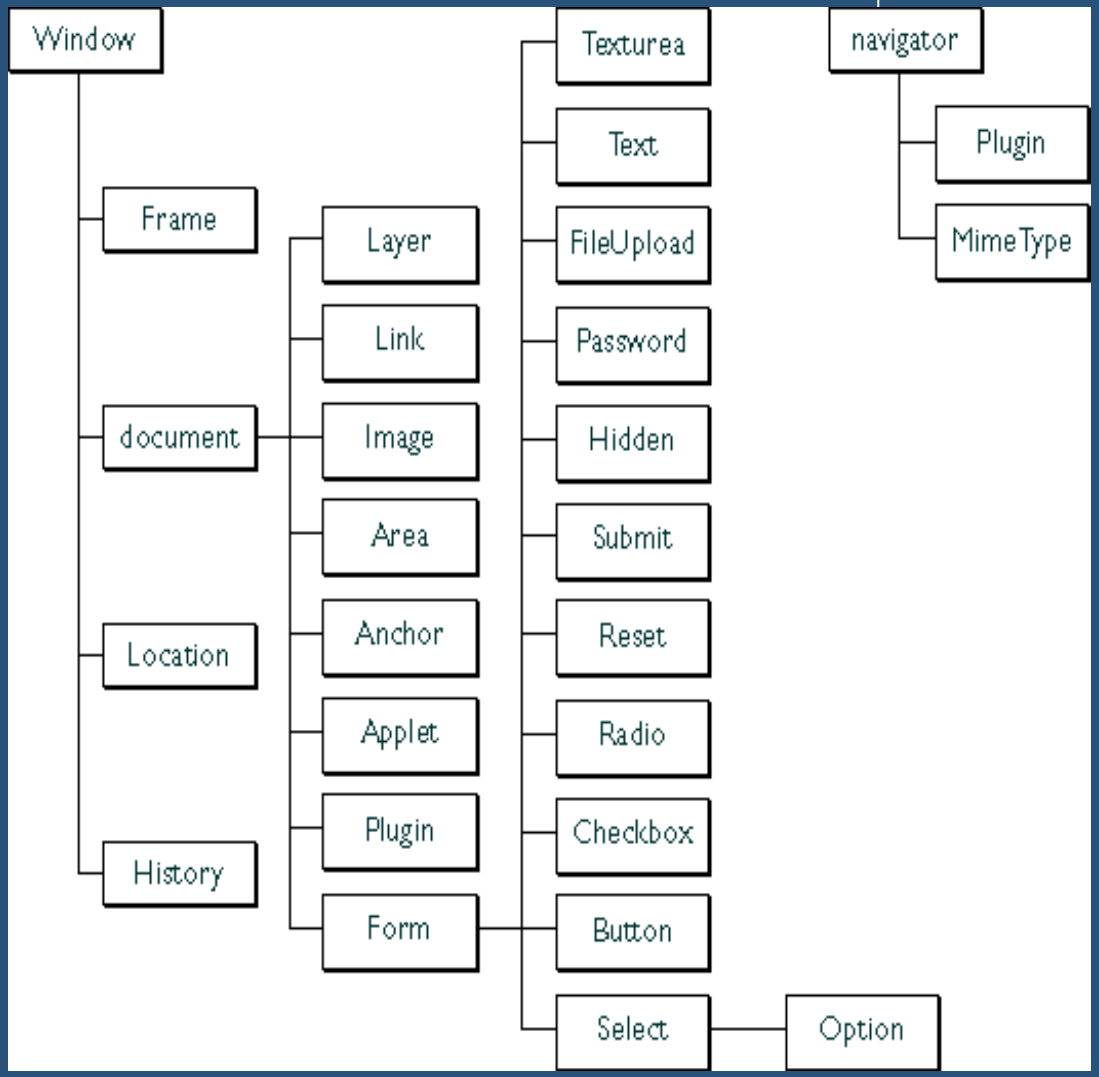
- Résultat obtenu :

```
Nous étions le 24/11/1981
Nous sommes le 30/11/2009
```

6 – JavaScript

Les objets du navigateur : introduction 1/3

- Les objets du navigateur sont organisés en hiérarchie :
  - L'objet **window** (fenêtre) occupe le sommet de cette hiérarchie.
  - Les « descendants » de **window** sont :
    - l'objet **frame**
    - l'objet **document**
    - l'objet **location**
    - l'objet **history**
  - L'objet **navigator** fournit des informations sur l'environnement du visiteur



6 – JavaScript

Les objets du navigateur : introduction 2/3

- Les objets du navigateur sont automatiquement instanciés à chaque étape du fonctionnement du navigateur
  - par exemple lors de l'ouverture d'une fenêtre ou de frames, la création des documents contenus dans chaque fenêtre ou frame et les divers éléments (formulaires, images, liens...) contenus dans chaque élément.
- Les applications JS peuvent alors dialoguer avec ces objets visuels et les manipuler. Le programmeur peut ainsi agir sur l'état du navigateur, de ses fenêtres et des documents et des composants qui y sont inclus.
- Mais attention, cette hiérarchie d'objets n'a rien à voir avec la notion d'héritage : les objets « descendants » ne sont considérés que comme des propriétés particulière de l'objet « ancêtre »
- Ainsi un objet **document** n'est pas un objet **window** particulier mais une propriété de **window** qui est elle-même un objet doté de propriétés et de méthodes...

6 – JavaScript

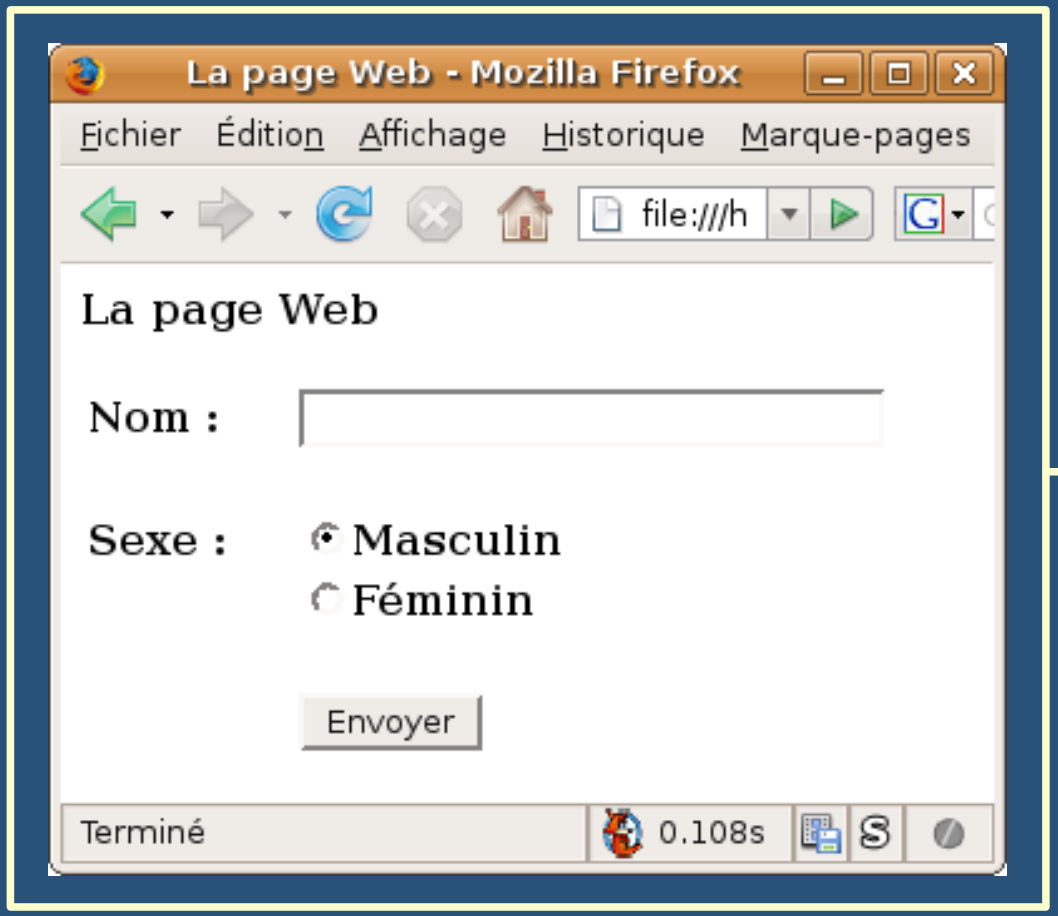
Les objets du navigateur : introduction 3/3

- L'objet **window**, le plus haut dans la hiérarchie correspond à la fenêtre même du navigateur.
- L'objet **document** fait référence au contenu de la fenêtre.
  - **document** regroupe au sein de ses propriétés l'ensemble des éléments HTML présents sur la page. Pour atteindre ces différents éléments, nous utiliserons :
    - Soit des méthodes propres à l'objet **document**, comme la méthode **getElementById()**, qui permet de trouver l'élément en fonction de son identifiant (id)
    - Soit des collections d'objets qui regroupent sous forme de tableaux JS tous les éléments d'un type déterminé

6 – JavaScript

Les objets du navigateur : exemple : un objet window

- La page suivante s'affiche dans une fenêtre du navigateur. C'est l'objet **window** (fenêtre) :



Objet **window**

6 – JavaScript

Les objets du navigateur : exemple : un objet document

- Dans cette fenêtre, il y a un document html. C'est l'objet **document**. Autrement dit, l'objet **window** contient l'objet **document** (notion de hiérarchie des objets JavaScript).

6 – JavaScript

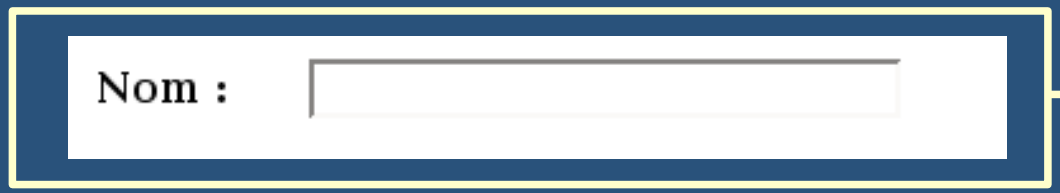
Les objets du navigateur : exemple : un objet form

- Dans ce document, on trouve un formulaire au sens html. C'est l'objet **form** (formulaire). Autrement dit, l'objet **window** contient un objet **document** qui lui, contient un objet **form**.

6 – JavaScript

Les objets du navigateur : exemple : des objets text, radio, button

- Dans ce formulaire, on trouve trois éléments : une zone de texte, des boutons radio et un bouton d'envoi. Ce sont respectivement l'objet **text**, l'objet **radio** (boutons radio) et l'objet **button** (bouton). L'objet **window** contient donc l'objet **document** qui contient l'objet **form** qui contient à son tour l'objet **text**, l'objet **radio** et l'objet **button**.



Objet **text**

Objet **radio**

Objet **button**

6 – JavaScript

Les objets du navigateur : exemple : synthèse

- La hiérarchie des objets de cet exemple est donc :

			->text
window	-> document	->form	->radio
			->button

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

- Soit par exemple pour le bouton **radio** "Masculin" : (**window).document.form.radio[0]**
  - L'objet **window** a été placé entre parenthèses car **window** occupe toujours la première place dans la hiérarchie des objets. Il est repris par défaut sous JavaScript et devient donc facultatif.

6 – JavaScript

Les événements : introduction

- L'utilisateur déclenche un « événement » (clic sur un bouton, déplacement souris, choix d'une option de liste déroulante ...) relativement à un objet (lien, composant de formulaire ...)
- L'événement est décelé (capté) par l'objet cible si celui-ci possède une "sensibilité" à l'événement. Il faut donc connaître la correspondance objet-événement.
- Si le programmeur prévoit un intérêt à "répondre" à cet événement, il doit à l'avance associer du code ou une fonction JS à un tel couple objet-événement. L'appel et l'exécution de ce code ou fonction seront automatiquement déclenchés par l'événement et constituent ainsi la "réponse" à celui-ci.
- Les fonctions sont déclarées dans la partie <head> et les appels en général associés à la balise de l'objet html qui va capter l'événement. Il faut veiller à bien gérer le passage de paramètres souvent un formulaire entier.

6 – JavaScript

Les événements : gestionnaires d'événements : introduction

- Le navigateur reconnaît un ensemble d'événements associés à des balises, des liens et des composants de formulaires. Par programmation, on peut leur associer des fonctions particulières appelées **gestionnaires d'événements** exécutés lorsque ces événements sont provoqués.
- Un **gestionnaire d'événement** est une procédure particulière attachée à une balise HTML
  - prédéfinie par le langage JS (par exemple onClick)
  - déclenchée par l'événement correspondant (clic sur un composant de type button)
  - qui apparaît dans la balise du composant qui reçoit l'événement
    - <input type="button" onClick= ...>
  - à laquelle on affecte une fonction écrite en JS déclarée au préalable
    - <input type="button" value="Calcul" onClick="calculer()">

6 – JavaScript

Les événements : gestionnaires d'événements : définition

- **<Balise onEvent = "code JS" >**
  - **Balise** est un nom de balise qui sait gérer un tel événement.
  - **onEvent** est le nom du gestionnaire d'événements associé à l'événement **Event**, comme **onClick**
  - **"code JS"** est généralement une fonction déclarée auparavant dans une section <head><script> ... </script></head>
    - Mais ce peut être aussi une suite d'instructions JS séparées par des point-virgules.

6 – JavaScript

Les événements : gestionnaires d'événements : exemple

- **<input type="button" value="Calcul" onClick="calculer(this.formul)">**

- Supposons déjà déclarée une fonction nommée **calculer()**. On peut appeler le navigateur à l'exécuter au moment où l'utilisateur clique sur un bouton.
- Pour cela il suffit d'affecter cette fonction **calculer()** avec ses paramètres au gestionnaire **onClick** qui réagit à l'événement click
- Le paramètre **this.formul** fait référence au formulaire de nom **formul** qui contient le bouton lui-même (**this** fait référence à l'objet **document courant**).
- Remarque : On peut utiliser plusieurs gestionnaires d'événements sur un même composant. Par exemple :
  - lien

6 – JavaScript

Les événements : liste non exhaustive 1/2

- Voici une liste non exhaustive des événements JS possibles :
  - **onClick** : se produit quand un composant Button, Checkbox, Radio, Link, Submit ou Reset reçoit un clic souris
  - **onFocus** : se produit quand un composant Textarea, Text ou Select est activé.
  - **onBlur** : se produit quand un champ Textarea, Text ou Select n'est plus activé (ils ont perdu le "focus").
  - **onChange** : se produit quand un champ Textarea, Text ou Select est modifié par l'utilisateur.
  - **onSelect :** se produit quand un composant Textarea ou Text est sélectionné.
  - **onSubmit :** se produit quand un formulaire est globalement validé par l'appui du bouton Submit.
  - **onReset :** se produit quand on clique sur le bouton reset d'un formulaire

6 – JavaScript

Les événements : liste non exhaustive 2/2

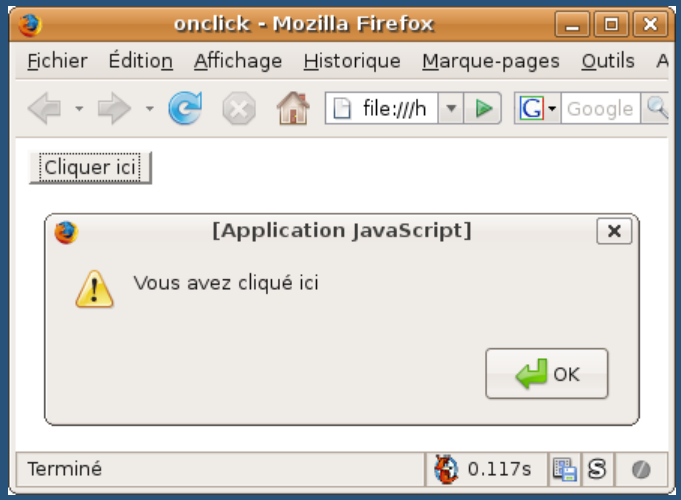
- **onLoad :** se produit quand le navigateur a fini de charger une fenêtre ou toutes les frames d'un frameset. L'événement onLoad se positionne dans la balise <body> ou dans la balise <frameset>
- **onUnload :** se produit quand on quitte un document. onUnload se positionne dans la balise <body> ou dans la balise <frameset>
- **onError** : se produit quand le chargement d'une page ou d'une image produit une erreur.
- **onMouseOver** : se produit quand la souris passe sur un hyperlien ou une zone activable d'une image.
- **onMouseOut** : se produit quand la souris quitte une zone Area d'une image ou un hyperlien.
- **onMouseDown** : se produit à la pression du bouton de la souris
- **onMouseUp** : se produit au relâchement du bouton de la souris (suite de **onMouseDown**).

6 – JavaScript

Les événements : l'événement onClick

- La clic de la souris est un événement fréquemment utilisé (**onClick**). Il survient lorsque l'utilisateur clique sur un lien ou un élément de formulaire. Il est possible de l'associer à pratiquement toutes les balises html. Il en est de même pour les événements : **onMouseDown**, **onMouseUp**, **onMouseOver**, **onMouseOut**.

- Exemple :
  - Au clic du bouton, une boîte d'alerte surgit.



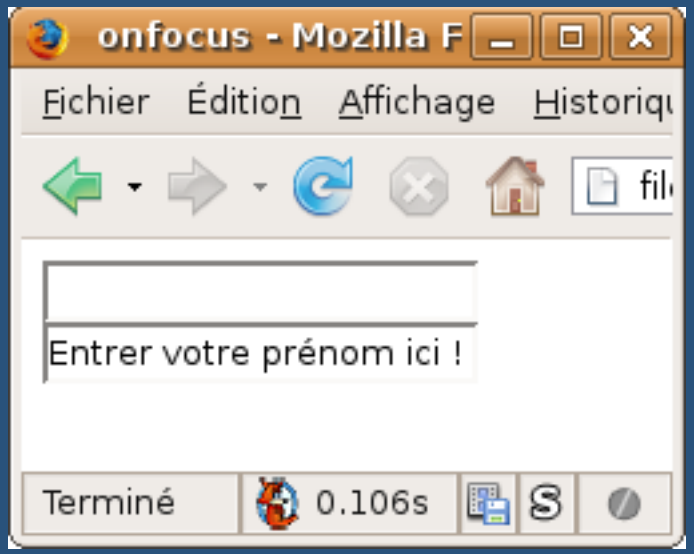

```
<html>
<head>
   <title>onclick</title>
</head>
<body>
<form>
   <input type="button" value="Cliquer ici"
   onclick="alert('Vous avez cliqu\351 ici')">
</form>
</body>
</html>
```

6 – JavaScript

Les événements : l'événement onFocus

- Un événement **onFocus** est généré lorsqu'un élément est activé, par exemple une ligne de texte d'un formulaire.

- Exemple :
  - Lorsque l'utilisateur clique dans une zone de texte, une inscription apparaît dans celle-ci.




```
<html>
<head>
   <title>onfocus</title>
</head>
<body>
<form name="form">
   <input name="t1" type="text"
   onfocus="document.form.t1.value='Entrer votre nom ici !'">
   <br>
   <input name="t2" type="text"
   onfocus="document.form.t2.value='Entrer votre pr&eacute;nom ici !'">
</form>
</body>
</html>
```

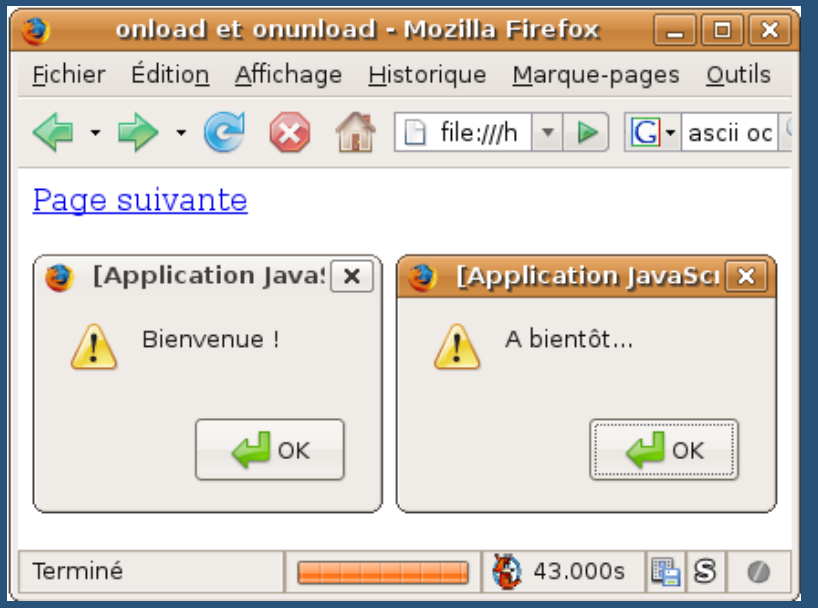
6 – JavaScript

Les événements : les événements onLoad & onUnload

- Au chargement (**onLoad**) de la page ou à la sortie (**onUnload**) de la page, des événements sont générés.

- Exemple :
  - Affichage d'un message au chargement de la page et à la sortie de celle-ci.

```
<html>
<head>
   <title>onload et onunload</title>
</head>
<body onload="alert('Bienvenue !')"
  onunload="alert('A bient\364t...')">
<a href="onfocus.htm">Page suivante</a>
</body>
</html>
```



6 – JavaScript

Les événements : les événements onMouseOver & onMouseOut 1/3

- L'événement **onMouseOver** se produit lorsque le pointeur de la souris passe au dessus (sans cliquer) d'un lien ou d'une image.
- L'événement **onMouseOut**, généralement associé à **onMouseOver**, se produit lorsque le pointeur quitte la zone sensible (lien ou image).
  - Ces 2 événements sont souvent utilisés pour réaliser un effet d'affichage d'une autre image au survol de l'image originale par le pointeur de la souris. Cet effet porte le nom de **rollover** en anglais.
    - Il importe que la dimension des images concernées par le script soit rigoureusement identique en hauteur et en largeur.

6 – JavaScript

Les événements : les événements onMouseOver & onMouseOut 2/3

- Exemple d'utilisation de onMouseOver et onMouseOut :
  - Au chargement de l'image, l'image initiale "**world.jpg**" est affiché. Au survol de la souris (**onMouseOver**), l'image "**world2.jpg**" apparaît. Lorsque le pointeur de la souris quitte l'image (**onMouseOut**), l'image originale "**world.jpg**" est ré-affichée.

```
<html>
<head>
   <title>rollover</title>
</head>
<body>
   <img name="image" src="world.jpg"
    onmouseover="document.image.src='world2.jpg';"
    onmouseout="document.image.src='world.jpg';">
</body>
</html>
```

6 – JavaScript

Les événements : les événements onMouseOver & onMouseOut 3/3

- Il est possible de reproduire le même effet que dans l'exemple précédent avec uniquement des feuilles de style CSS.

```
<html>
<head>
   <title>rollover CSS</title>
   <style type="text/css" media="all">
   a.image {display: block;
            width: 110px;
            height: 110px;
            background-image: url(world.jpg);
            background-repeat: no-repeat;
   }
   a.image:hover {background-image: url(world2.jpg);}
   </style>
</head>
<body>
   <a class="image" href="#"></a>
</body>
</html>
```

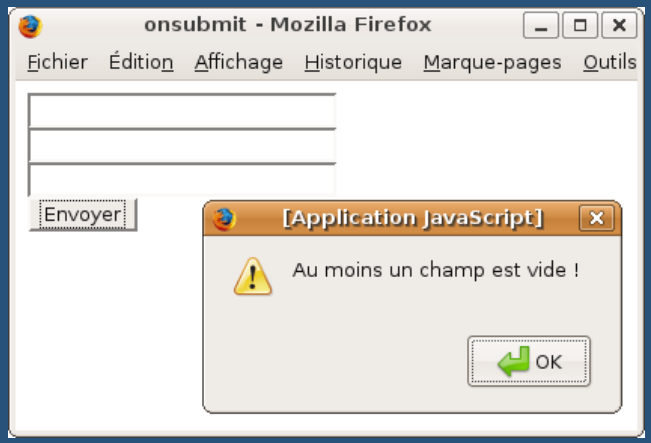
- Le résultat est le même que celui sur le transparent précédent.

6 – JavaScript

Les événements : l'événement onSubmit

- L'événement créé par le clic sur le bouton d'envoi d'un formulaire pourra judicieusement être mis à profit pour, par exemple, effectuer des vérifications concernant l'encodage des champs de celui-ci.

```
<html><head><title>onsubmit</title>
   <script type="text/javascript" language="javascript1.2">
   function verif() {
      a = document.formulaire.texte1.value;
      b = document.formulaire.texte2.value;
      c = document.formulaire.texte3.value;
      if (a==""|| b==""|| c=="") {
         alert("Au moins un champ est vide !");
         return false;
      }
      return true;
   }
   </script>
</head><body>
   <form name="formulaire" onsubmit="return verif();">
      <input name="texte1" size="26"><br>
      <input name="texte2" size="26"><br>
      <input name="texte3" size="26"><br>
      <input type="submit">
   </form>
</body></html>
```



6 – JavaScript
Les formulaires : introduction

- Avec JS, les formulaires HTML prennent une toute autre dimension.
- En JS, il est possible d'accéder à chaque élément d'un formulaire pour, par exemple, y lire et/ou écrire une valeur.
  - Les pages Web deviennent ainsi plus interactives

6 – JavaScript

Les formulaires : la ligne de texte : introduction

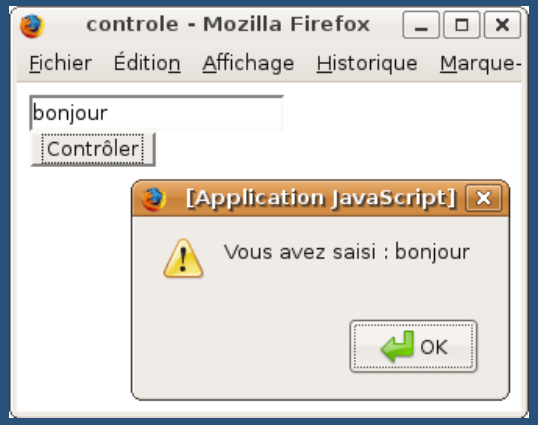
- La zone de texte est l'élément d'entrée/sortie par excellence de JavaScript.
- L'objet **text** possède 3 propriétés et une méthode :
  - **name :** indique le nom du contrôle par lequel on pourra accéder à la zone de texte.
  - **defaultvalue :** indique la valeur par défaut qui sera affiché dans la zone de texte à sa création.
  - **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.
  - **focus() :** donne le focus, c'est à dire que le curseur est dans le champ de saisie.

6 – JavaScript

Les formulaires : la ligne de texte : exemple : lire une valeur

- Il s'agit d'attendre la saisie d'une valeur dans la zone de texte et d'appuyer sur un bouton pour contrôler celle-ci dans une boîte d'alerte. Le script complet est :

```
<html>
<head>
   <title>controle</title>
   <script type="text/javascript" language="javascript1.2">
   function controler(formulaire) {
      var test = document.formulaire.input.value;
      alert("Vous avez saisi : " + test);
   }
   </script>
</head>
<body>
   <form name="formulaire">
   <input type="text" name="input" value=""><br>
   <input type="button" name="bouton" value="Contr&ocirc;ler"
   onclick="controler(formulaire)">
   </form>
</body>
</html>
```

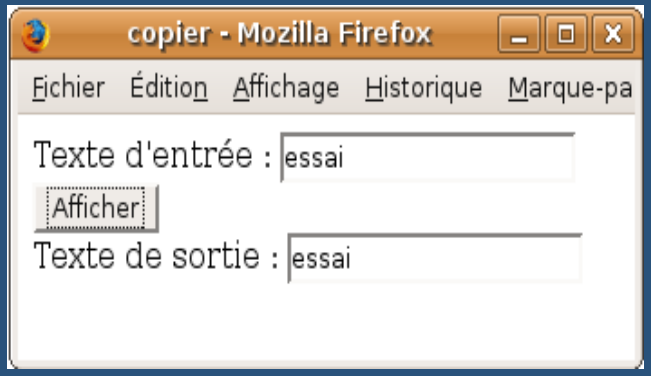


6 – JavaScript

Les formulaires : la ligne de texte : exemple : copier une valeur

- Après avoir entré une valeur quelconque dans la zone de texte d'entrée, appuyer sur un bouton pour afficher cette même valeur dans la zone de texte en sortie. Le code est :

```
<html>
<head>
<title>copier</title>
<script type="text/javascript" language="javascript1.2">
function copier() {
   var testin=document.formulaire.input.value;
   document.formulaire.output.value=testin;
}
</script>
</head>
<body>
   <form name="formulaire">
   Texte d'entr&eacute; : <input type="text" name="input" value=""><br>
   <input type="button" name="bouton" value="Afficher" onclick="copier()"><br>
   Texte de sortie : <input type="text" name="output" value=""><br>
   </form>
</body>
</html>
```

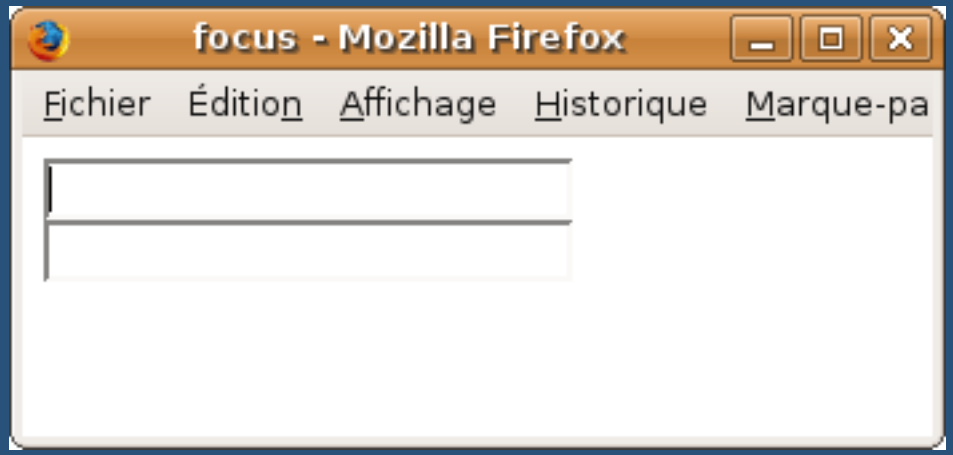


6 – JavaScript

Les formulaires : la ligne de texte : exemple : donner le focus

- Il est souvent utile de placer directement le curseur dans la première zone de texte d'un formulaire. Le code est :

```
<html>
<head>
   <title>focus</title>
</head>
<body onload="document.formulaire.entree.focus();">
   <form name="formulaire">
   <input type="text" name="entree" size="25" value="">
   <input type="text" name="entree2" size="25" value="">
   </form>
</body>
</html>
```

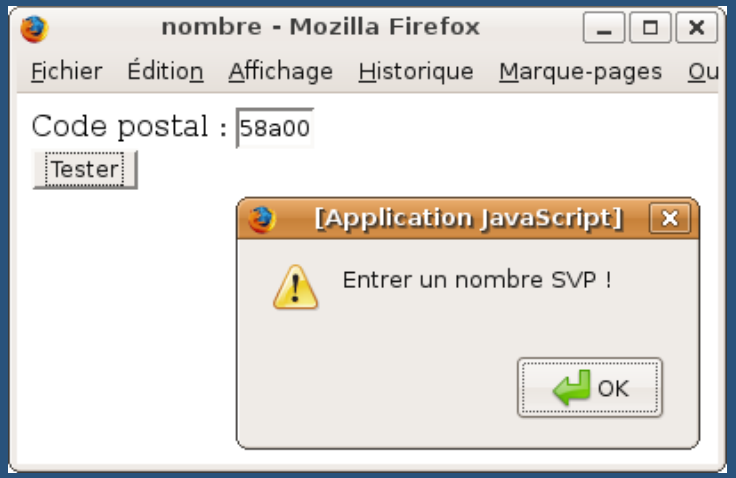


6 – JavaScript

Les formulaires : la ligne de texte : exemple : saisie d'un nombre

- Certaines valeurs saisies ne peuvent être que des nombres. Le script suivant permet de le vérifier (grâce à **isNaN(n)** pour déterminer s'il ne s'agit pas d'un nombre NaN=Not a Number) :

```
<html><head>
<title>nombre</title>
<script type="text/javascript" language="javascript1.2">
function verifn() {
   var n=document.formulaire.nombre.value;
   if (isNaN(n) || (n==0))
      alert("Entrer un nombre SVP !");
   else
      alert("Entr\351e valide...");
}
</script></head>
<body>
   <form name="formulaire">
   Code postal : <input type="text" name="nombre" size="5" maxlength="5"><br>
   <input type="button" name="bouton" value="Tester" onclick="verifn()"><br>
   </form>
</body>
</html>
```



6 – JavaScript

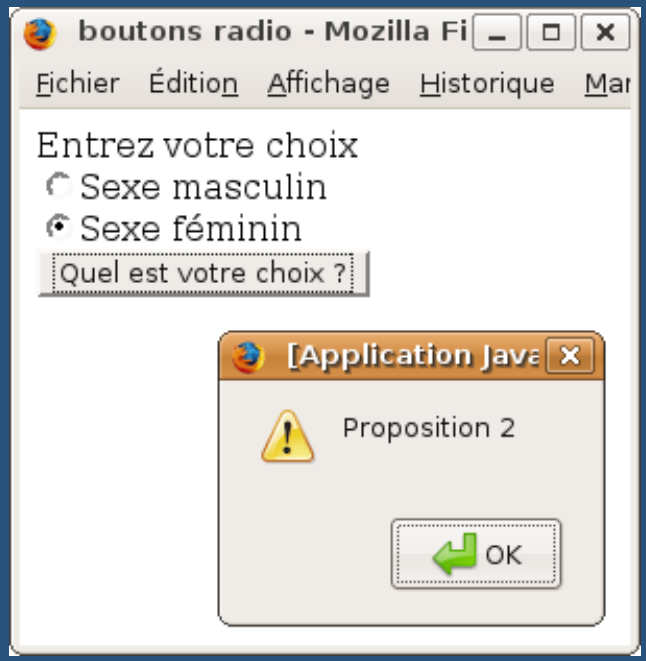
Les formulaires : les boutons radio : introduction

- Les boutons radio sont utilisés pour noter un choix **et seulement un** parmi un ensemble de propositions. Voici 5 propriétés utiles liées à ces boutons radio :
  - **name :** indique le nom du contrôle. Tous les boutons portent le même nom.
  - **index :** indique 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.

6 – JavaScript

Les formulaires : les boutons radio : exemple

- Exemple :
- Un formulaire demande le sexe du visiteur.




```
<html><head><title>boutons radio</title>
<script type="text/javascript"
        language="javascript1.2">
function choixsexe(form) {
if (form.choix[0].checked)
   alert("Proposition "+form.choix[0].value);
if (form.choix[1].checked)
   alert("Proposition "+form.choix[1].value);
}
</script>
</head><body>
   Entrez votre choix <form name="formulaire">
   <input type="radio" name="choix" value="1">Sexe masculin<br>
   <input type="radio" name="choix" value="2">Sexe f&eacute;minin<br>
   <input type="button" name="but" value="Quel est votre choix ?"
   onclick="choixsexe(document.formulaire)">
   </form>
</body>
</html>
```

6 – JavaScript

Les formulaires : les cases à cocher : introduction

- Les cases à cocher (checkbox) sont utilisées pour noter un ou plusieurs choix parmi un ensemble de propositions. Voici 4 propriétés utiles liées à ces cases à cocher :
  - **name :** indique le nom du contrôle. Toutes les cases à cocher portent un nom différent.
  - **checked :** indique l'état en cours de l'élément case à cocher.
  - **defaultchecked :** indique l'état du bouton sélectionné par défaut.
  - **value :** indique la valeur de l'élément case à cocher.

6 – JavaScript

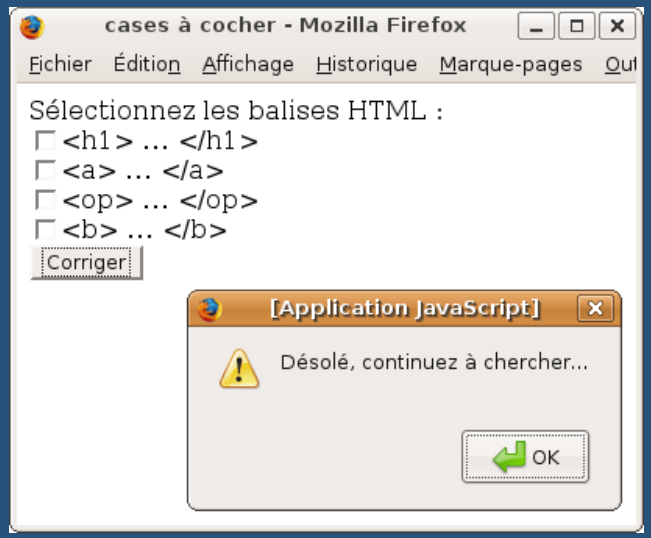
Les formulaires : les cases à cocher : exemple 1/2

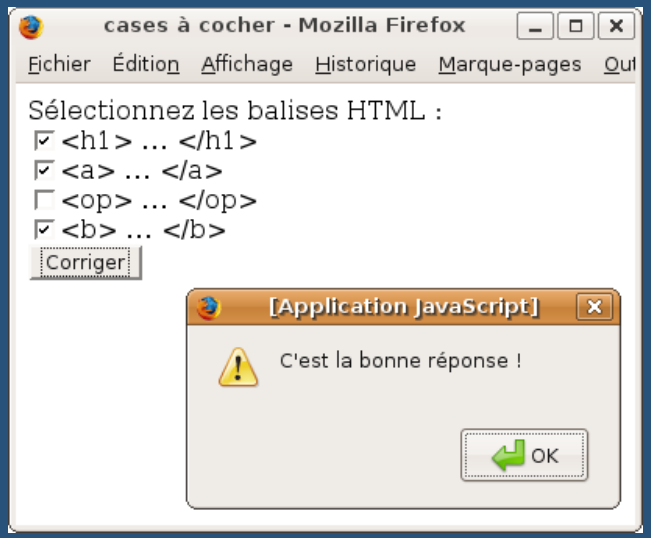
- Exemple : Soit le choix multiple suivant :

  Sélectionner les balises HTML :

  - ☐ **<h1> ... </h1>**
  - ☐ **<a> ... **
  - ☐ **<op> ... </op>**
  - ☐ ** ... **

  - La bonne réponse est la première, seconde et quatrième proposition.





6 – JavaScript

Les formulaires : les cases à cocher : exemple 2/2

- Voici le code de l'exemple du transparent précédent :

```
<html>
<head><title>cases &agrave; cocher</title>
<script type="text/javascript" language="javascript1.2">
function reponse(form) {
if ( (form.check1.checked == true) && (form.check2.checked == true) &&
    (form.check3.checked == false) && (form.check4.checked == true) )
   alert("C'est la bonne r\351ponse !");
else
   alert("D\351sol\351, continuez \340 chercher...");
}
</script>
</head><body>
   S&eacute;lectionnez les balises HTML : <form name="formulaire">
   <input type="checkbox" name="check1" value="1">&lt;h1&gt; ... &lt;/h1&gt;<br>
   <input type="checkbox" name="check2" value="2">&lt;a&gt; ... &lt;/a&gt;<br>
   <input type="checkbox" name="check3" value="3">&lt;op&gt; ... &lt;/op&gt;<br>
   <input type="checkbox" name="check4" value="4">&lt;b&gt; ... &lt;/b&gt;<br>
   <input type="button" name="but" value="Corriger"
   onclick="reponse(document.formulaire)">
   </form>
</body>
</html>
```

6 – JavaScript

Les formulaires : la liste déroulante : introduction

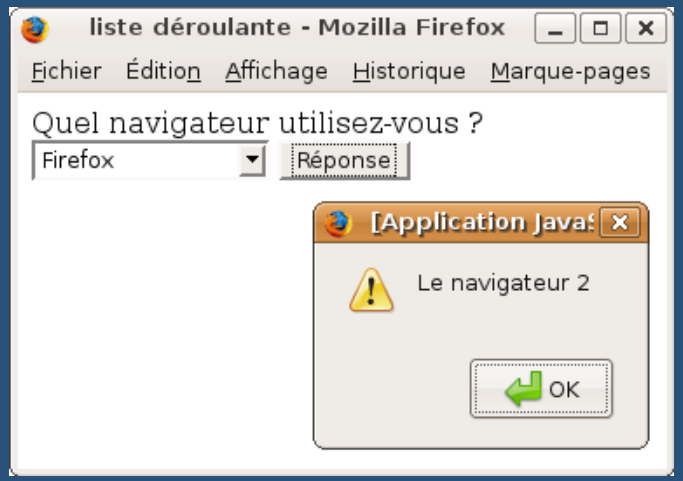
- La liste déroulante permet de proposer diverses options sous la forme d'une liste dans laquelle l'utilisateur peut cliquer pour faire son choix. Ce choix reste alors affiché. La liste déroulante est créée par la balise **<select>...</select>** et les éléments de la liste par une ou plusieurs balise(s) **<option>**. Voici 4 propriétés utiles liées la liste déroulante :
  - **name :** indique le nom de la liste déroulante.
  - **length :** indique le nombre d'éléments de la liste.
  - **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 zone.

6 – JavaScript

Les formulaires : la liste déroulante : exemple

- Exemple : On demande le type de navigateur utilisé au moyen d'une liste déroulante. Les options proposées sont : Internet Explorer, Firefox ou autre.

```
<html>
<head><title>liste d&eacute;roulante</title>
<script type="text/javascript" language="javascript1.2">
function liste(form) {
   alert("Le navigateur "+(form.list.selectedIndex+1));
}
</script>
</head>
<body>
   Quel navigateur utilisez-vous ?
   <form name="formulaire">
   <select name="list">
      <option value="1">Internet Explorer
      <option value="2">Firefox
      <option value="3">Autre
   </select>
   <input type="button" name="b" value="R&eacute;ponse"
   onclick="liste(document.formulaire)">
   </form>
</body>
</html>
```

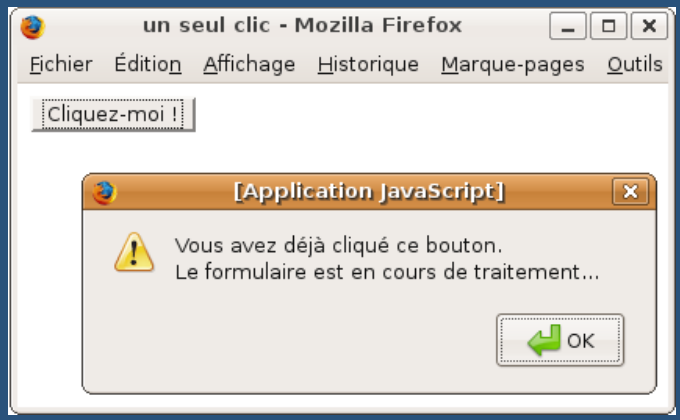


6 – JavaScript

Les formulaires : le bouton de validation : exemple

- Certains utilisateurs (impatients) ont la fâcheuse manie de cliquer plusieurs fois le bouton de validation, ce script calmera leur ardeur.

```
<html>
<head><title>un seul clic</title>
<script type="text/javascript" language="javascript1.2">
var nombreclic=0;
function compteclic(form) {
nombreclic++;
if (nombreclic > 1)
   alert("Vous avez d\351j\340 cliqu\351 ce bouton.\nLe formulaire
   est en cours de traitement...");
}
</script>
</head><body>
   <form name="formulaire">
   <input type="button" value="Cliquez-moi !"
   onclick="compteclic(document.formulaire)">
   </form>
</body>
</html>
```



6 – JavaScript

Les formulaires : la variable this

- La variable **this** est une sorte de raccourci qui permet de référencer l'objet **document courant**.
- **this** est souvent utilisée lorsque l'on insère du code JavaScript au sein d'une balise HTML.
  - Elle permet alors de faire référence à l'objet **document** défini par cette balise.
- Exemple :

```
<form name="formulaire">
<input type="button" value="Cliquez-moi !" onclick="compteclic(this)">
</form>
```

6 – JavaScript

L'objet window : caractéristiques

- De façon générale, comme tout se déroule dans une fenêtre du navigateur, le nom de la fenêtre est implicite :
  - le préfixe **window** peut être omis pour désigner un objet ou une méthode de la fenêtre courante (sauf dans un gestionnaire d'événement dont l'objet courant étant un document, il faut préciser la fenêtre de ce document).

- Bien entendu si la propriété ou la méthode s'adresse à une fenêtre définie par la programmeur à l'aide de la méthode **open()** (méthode utilisée pour ouvrir une nouvelle fenêtre dans le navigateur), il faut préfixer par son nom.

6 – JavaScript

L'objet window : propriétés defaultStatus et status : définitions

- La barre d'état (**status bar**) se situe en bas du navigateur et permet au surfeur de visualiser l'adresse du lien que survole le pointeur de sa souris.
- La propriété **defaultStatus** : représente le message qui sera affiché par défaut dans la barre de statut à l'ouverture de la page

```
<body onLoad="window.defaultStatus='Bonjour à tous'">
```

- La propriété **status** : contient le texte affiché dans la barre de statut de la fenêtre.

```
window.status="Pensez à fermer vos fenêtres !"
```

- **Attention :** Les deux propriétés ci-dessus ne fonctionne pas avec Firefox !

6 – JavaScript

L'objet window : propriétés defaultStatus et status : exemple

- Exemple :
- On affiche le message « Bonjour à tous » comme texte par défaut de la barre d'état par l'événement **onload**.
- Au survol d'un lien par la souris, le message « Pensez à fermer vos fenêtres ! » s'affichera dans la barre d'état.
  - « **return true;** » force le navigateur à effectuer l'action **onmouseover** plutôt que sa façon habituelle de procéder.

```
<html>
<head> <title>status</title> </head>
<body onload="window.defaultStatus='Bonjour à tous';">
   <h1>JavaScript</h1>
   <a href="page2.html" onmouseover="window.status='Pensez à fermer
   vos fenêtres !'; return true;">Lien</a>
</body>
</html>
```

6 – JavaScript

L'objet window : autres propriétés

- Voici les autres propriétés de l'objet **window**
  - **length** : représente le nombre de cadres dans la fenêtre parente (0 sinon).
  - **name** : représente le nom de la fenêtre
  - **opener** : spécifie le nom de la fenêtre parent qui l'a créée dynamiquement avec **open()**
  - **parent** : est le nom de la frame où se trouve la fenêtre
  - **self** : est un synonyme pour le nom de la fenêtre et fait référence à la fenêtre courante
  - **top** : fait référence à la fenêtre principale du navigateur.
  - **window** : est un synonyme pour la fenêtre courante
  - **closed** : est un booléen qui indique si la fenêtre a été fermée.
    - Utilisation : `if (! fen.closed) fen.close();`

6 – JavaScript

L'objet window : méthodes déjà abordées

- Certaines méthodes de l'objet **window** ont déjà été abordées précédemment.

- On peut citer ainsi :
  - la méthode **alert()**
  - la méthode **confirm()**
  - la méthode **prompt()**
  - les méthodes **setTimeout()** et **clearTimeout()**

6 – JavaScript

L'objet window : méthodes open() et close()

- Avec les méthodes **open()** et **close()**, le programmeur dispose de moyens très souples pour ouvrir et fermer des fenêtres auxiliaires avec les gestionnaires d'événements.
- La syntaxe de **open()** est :
  - **nomFenetre =window.open("URL","nom_fenetre","options");**
    - La mention « **window.** » est facultative.
    - **"URL"** est l'adresse du document à charger dans la nouvelle fenêtre.
    - **"nom_fenetre"** est le nom auquel on pourra faire référence à la fenêtre dans le code.
    - **"options"** est une liste d'éléments optionnels qui précisent l'aspect de la fenêtre (voir page suivante).
    - **nomFenetre** sera utilisé par la suite pour faire référence à un objet ou une propriété de la nouvelle fenêtre.
- La syntaxe de **close()** est :
  - **nomFenetre.close();** ou **window.close("nom_fenetre");** ou **self.close()**

6 – JavaScript

L'objet window : méthode open() : valeurs de la partie « options »

- On peut contrôler l'apparence de la nouvelle fenêtre à l'aide du 3ème paramètre de la méthode **open()** qui accepte une liste d'options sur une seule ligne séparées par des virgules :
  - `height= x` Hauteur de la nouvelle fenêtre en pixels
  - `width= y` Largeur de la nouvelle fenêtre en pixels
  - `toolbar[=yes|no]` Affichage de la barre d'outils (défaut : yes)
  - `location[=yes|no]` Affichage de la barre d'adresse url
  - `directories[=yes|no]` Affichage des boutons d'accès rapide (liens)
  - `status[=yes|no]` Affichage de la barre d'état en bas
  - `menubar[=yes|no]` Affichage de la barre de menu
  - `scrollbars[=yes|no]` Affichage des barres de défilement
  - `resizable[=yes|no]` Nouvelle fenêtre redimensionnable
  - `alwaysRaised[=yes|no]` Nouvelle fenêtre avec visibilité permanente
  - `dependent[=yes|no]` Nouvelle fenêtre fermée avec la fen. parente
  - `top = x` Position en pixels relatif au coin haut de l'écran
  - `left = y` Position en pixels relatif au coin gauche de l'écran

6 – JavaScript

L'objet window : méthode open() : exemple 1 : ouvrir seul

- Exemple :
- La fonction **OuvrirFenetre()** déclenchée par l'événement **onClick** sur le bouton « Nouvelle fenêtre 1 » utilise la fonction **open()** pour ouvrir une nouvelle fenêtre nommée **fen1**

```
<script type="text/javascript" language="javascript1.2">
   var fen1;
   var hauteur=200;   // hauteur de la fenêtre à créer
   var largeur=500;
   var options="width="+largeur+",height="+hauteur+
"toolbar=yes, directories=no, menubar=no, scrollbars=yes, status=yes";

   function OuvrirFen1() {
      fen1 = open("","Nouvelle_fenetre1",options);
   }
</script>
<form>
Pour cr&eacute;er une fen&ecirc;tre "enfant", cliquer sur ce bouton :
<input type=Button value="Nouvelle fen&ecirc;tre 1" onClick="OuvrirFen1();">
</form>
```

6 – JavaScript

L'objet window : méthode open() : exemple 2 : ouvrir / fermer

- Exemple :
- La méthode **close()** adressée à une variable fenêtre permet de la fermer. Il est préférable avant de tester si elle a été ouverte (elle peut être masquée) avec une condition du genre if (fen != null)

```
<script type="text/javascript" language="javascript1.2">
   var fen1;
   var hauteur=200;   // hauteur de la fenêtre à créer
   var largeur=500;
   var options="width="+largeur+",height="+hauteur+
"toolbar=yes, directories=no, menubar=no, scrollbars=yes, status=yes";

   function OuvrirFen1() {
      fen1 = open("","Nouvelle_fenetre1",options);
   }
</script>
<form>
Pour cr&eacute;er une fen&ecirc;tre "enfant", cliquer sur ce bouton :
<input type=Button value="Nouvelle fen&ecirc;tre 1" onClick="OuvrirFen1();">
<input type=Button value="Fermer"  onClick="fen1.close()">
</form>
```

6 – JavaScript

L'objet location de window : introduction

- Avec l'objet **location**, il est possible d'avoir accès à l'adresse **url** de la page affichée par le navigateur.
  - **href** : cette propriété sauvegarde l'adresse de la fenêtre actuelle. Si cette adresse est modifiée par l'intermédiaire de JS, le navigateur exécute un saut à la nouvelle adresse comme s'il s'agissait d'un lien.
    - Exemple :

```
<script type="text/javascript" language="javascript1.2">
   if (screen.width >= 1024)
      window.location.href = "fichier_highres.htm";
   else
      window.location.href = "fichier_lowres.htm";
</script>
```

    - On peut aussi rencontrer l'écriture `window.location="fichier.htm";`
  - **reload()** : cette méthode agit exactement comme si l'utilisateur cliquait sur le bouton **Actualiser** du navigateur, ce qui sera parfois nécessaire pour certains scripts

6 – JavaScript

L'objet window : méthode open() : exemple 3 : afficher un document

- Exemple : chargement d'un document à la création d'une nouvelle fenêtre :
- Cet exemple est semblable au précédent, à la différence que le fichier **nouvfen.html**, passée en 1er paramètre de **open()** va être affiché dans le document de la nouvelle fenêtre.

```
function OuvrirFenetre2() {
   fen2 =open("nouvfen.html","Nouvelle_fenetre",options);

   /* les 2 lignes suivantes sont équivalentes
   fen2 =open("","Nouvelle_fenetre",options);
   fen2.location.url="nouvfen.html"; */
}
```

6 – JavaScript

L'objet window : méthode print()

- La méthode **print()** permet d'imprimer la page courante.
  - Elle ne comporte pas de paramètres
  - Elle correspond à la commande **Fichier - Imprimer** du navigateur
  - Exemple :

```
<form>
<input type= "button" value="Imprimer" onclick="print();">
</form>
```

6 – JavaScript
L'objet screen : introduction

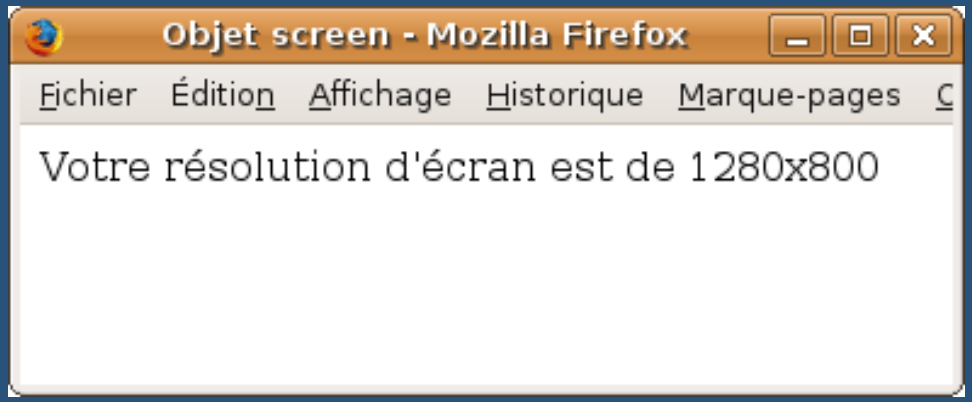
- L'objet **screen** renvoie des informations relatives à l'écran de l'utilisateur.
  - Les propriétés de cet objet sont peu nombreuses mais souvent très utiles. Voici quelques propriétés de **screen :**
    - **width** : indique (en pixels) la largeur de l'écran définie par la résolution d'écran (exemple = 1280).
    - **height** : indique (en pixels) la hauteur de l'écran définie par la résolution d'écran (exemple = 800).
    - **colorDepth** : donne la profondeur (en bits) de la palette de couleurs du navigateur (exemple = 24).
    - **availWidth :** indique (en pixels) la largeur de l'écran disponible moins la barre des tâches (exemple = 1280).
    - **availHeight** : indique (en pixels) la hauteur de l'écran disponible moins la barre des tâches (exemple = 750).

6 – JavaScript

L'objet screen : exemple

- Exemple :
- Voici un script permettant de connaître la résolution d'écran du visiteur

```
<html>
<head>
   <title>Objet screen</title>
</head>
<body>
   <script type="text/javascript" language="javascript1.2">
      document.write("Votre r&eacute;solution d'&eacute;cran est de "
                     + screen.width + "x" + screen.height);
   </script>
</body>
</html>
```



6 – JavaScript

L'objet history de window : introduction

- Avec l'objet **history**, il est possible d'avoir accès aux pages Web qui ont été visitées par l'utilisateur.
  - **back()** : cette méthode charge la page Web visitée précédente. Elle n'attend aucun paramètre. Exemple :

```
<html><head><title>history</title></head><body>
   <a href="javascript:history.back();">Retour &agrave; la page
  pr&eacute;c&eacute;dente</a>
</body></html>
```

  - **forward()** : cette méthode charge la page Web suivante si elle existe. Elle n'attend aucun paramètre.
  - **go(n)** : cette méthode avance ou recule du nombre de pages désirées **n** dans l'historique sauvegardé. Elle attend comme paramètre le nombre de pages à sauter (un nombre positif avance, un nombre négatif recule). Exemple :

```
<a href="javascript:history.go(-3);">Retour au sommaire</a>
```

6 – JavaScript
L'objet navigator : introduction

- L'objet **navigator** de JS fournit un ensemble d'informations sur l'environnement du visiteur et tout particulièrement sur le navigateur qu'il utilise.
  - Cet objet permet de déterminer le système d'exploitation de la machine du visiteur, le type de navigateur, sa version exacte, la langue du navigateur, les plugins installés afin d'adapter les scripts de vos pages HTML.

6 – JavaScript

L'objet navigator : propriétés : généralités

- Voici quelques propriétés de l'objet **navigator**
  - **appCodeName** : retourne le « surnom » du navigateur (toujours "Mozilla")
  - **appName** : retourne le nom du navigateur
  - **appVersion** : indique la version du navigateur. Le format est : « numéro de version (plateforme, nationalité) »
  - **language** : spécifie la langue utilisée par le navigateur. Cette propriété n'existe pas sous « Internet Explorer »
  - **userLanguage** : spécifie la langue utilisée par le navigateur. Cette propriété ne fonctionne que sous « Internet Explorer »
  - **mimeTypes** : renvoie un tableau des types MIME supportés par le navigateur, c'est-à-dire les types de fichiers enregistrés.
  - **platform** : indique le système d'exploitation sur lequel tourne le navigateur
  - **plugins** : renvoie un tableau contenant la liste des plugins installés
  - **userAgent** : spécifie des informations relatives au navigateur

6 – JavaScript

L'objet navigator : propriétés : valeurs sur un navigateur donné

- Voici les valeurs retournées par mon navigateur (machine sous Ubuntu version 7.10 avec Mozilla Firefox 2.0.0.11) :
  - **appCodeName** : Mozilla (pour Firefox et Internet Explorer)
  - **appName** : Netscape (ou « Microsoft Internet Explorer »)
  - **appVersion** : 5.0 (X11; fr) (ou « 5.0 (Windows; fr-FR) »)
  - **language** : fr (ou undefined sous IE6)
  - **userLanguage :** undefined (ou fr sous IE6)
  - **mimeTypes** : application/x-shockwave-flash, audio/mpeg, audio/x-mpeg, video/mpeg, video/mpeg4, audio/mpeg4 ...
  - **platform** : Linux i686 (ou Win32 sous Windows)
  - **plugins** : libflashplayer.so, libvlcplugin.so, libjavaplugin_oji.so ...
  - **userAgent** : Mozilla/5.0 (X11; U; Linux i686; fr; rv:1.8.1.11) Gecko/20071204 Ubuntu/7.10 (gutsy) Firefox/2.0.0.11

6 – JavaScript

L'objet navigator : propriétés : déterminer le système d'exploitation

- Bien que la propriété **plateform** pourrait convenir, c'est plutôt la propriété **appVersion** de **navigator** qui est utilisée dans les scripts.
- Par la méthode **indexOf()** de l'objet **String**, on peut voir si la chaîne "**X11**", "**Win**" ou "**Mac**" est présente dans la propriété **appVersion**

```
<html><head><title>Systeme d'exploitation</title></head><body>
   <script type="text/javascript" language="javascript1.2">
      if (navigator.appVersion.indexOf("X11") != -1) {
         system = "Linux";
      } else {
         if (navigator.appVersion.indexOf("Win") != -1) {
            system = "Windows";
         } else {
            if (navigator.appVersion.indexOf("Mac") != -1)
               system = "Macintosh";
            else
               system = "Inconnu";
         }
      }
      document.write("Mon syst&egrave;me d'exploitation est " + system);
   </script>
</body></html>
```

6 – JavaScript

L'objet navigator : propriétés : déterminer le type de navigateur

- Par la méthode **indexOf()** de l'objet **String**, on peut voir si la chaîne "**Firefox**" ou "**MSIE**" est présente dans la propriété **userAgent** de l'objet **navigator** :

```
<html>
<head><title>type de navigateur</title></head>
<body>
   <script type="text/javascript" language="javascript1.2">
      if (navigator.userAgent.indexOf("Firefox") != -1) {
         navigateur = "Firefox";
      } else {
         if (navigator.userAgent.indexOf("MSIE") != -1) {
            navigateur = "Internet Explorer";
         } else
            navigateur = "Autre";
      }
      document.write("Votre navigateur est " + navigateur);
   </script>
</body>
</html>
```

6 – JavaScript

L'objet navigator : propriétés : identifier les versions d'Internet Explorer

- Il suffit d'utiliser les informations transmises par **navigator.userAgent**
  - Pour **Internet Explorer 6**, **navigator.userAgent** contiendra :
    - « Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1) »
  - Pour **Internet Explorer 5.5**, **navigator.userAgent** contiendra :
    - « Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.1; SV1) »
  - Pour **Internet Explorer 5**, **navigator.userAgent** contiendra :
    - « Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.1; SV1) »
  - Pour **Internet Explorer 4**, **navigator.userAgent** contiendra :
    - « Mozilla/4.0 (compatible; MSIE 4.01; Windows NT 5.1; SV1) »
- Il suffira de rechercher la présence des chaînes partielles « **MSIE 6** », « **MSIE 5.5** », « **MSIE 5.0** » ou « **MSIE 4.0** » pour identifier la version d'Internet Explorer.

6 – JavaScript

L'objet navigator : propriétés : déterminer la langue du navigateur

- La langue du navigateur (et en toute probabilité, la langue du visiteur) est fournie par la propriété **userLanguage** pour **MSIE** ou la propriété **language** pour **Firefox**, de l'objet **navigator**. On en extraira les 2 premières lettres par la fonction **substring(0, 2)** :

```
<html>
<head><title>langue</title></head>
<body>
   <script type="text/javascript" language="javascript1.2">
      if (navigator.userAgent.indexOf("MSIE") != -1) {
         langue=navigator.userLanguage.substring(0, 2);
      } else
         langue=navigator.language.substring(0, 2);
      if (langue="fr")
         langue="fran&ccedil;ais";
      document.write("Vous parlez le " + langue + ".");
   </script>
</body>
</html>
```

6 – JavaScript

L'objet navigator : propriétés : test des plugins

- Tout aussi important est la nécessité de savoir si le navigateur possède un plugin, module externe, pour interpréter certains types de fichiers comme les fichiers sons, vidéos, pdf ...
- Exemple :
- Il faut tester si le navigateur peut interpréter du code Shockwave avant de lui envoyer un fichier. Pour cela on interroge le tableau **plugins[ ]**, propriété de **navigator** : possède-t'il un élément indexé par 'Shockwave' ?

```
if (navigator.plugins['Shockwave'])
  document.write('<embed name="acte_1" src="acte_1.swf"
  width="100%" height="100%" align="LEFT" quality="high" salign="lT">');
else
 document.write('Désolé, ce navigateur ne sait pas afficher Shockwave');
```

6 – JavaScript

L'objet document de window : introduction

- L'objet **document** possède beaucoup de propriétés et de méthodes importantes comme **getElementById()** qui permet l'accès à un élément html par son attribut **id** et **getElementsByName()** pour l'accès à un élément HTML par son attribut **name**.
  - Ces méthodes utilisées dans le DHTML (« Dynamic HTML » : capacité à modifier les éléments d'une page Web après son chargement initial) sont hors de portée de ce cours.
- La propriété **lastModified** sauvegarde la date et l'heure de la dernière modification du fichier dans le format GMT (heure de Greenwich). Pour tout autre affichage, il suffit d'utiliser **Date**.

```
<script type="text/javascript" language="javascript1.2">
   document.write("Derni&egrave;re maj : " + document.lastModified);
</script>
Le script précédent affichera « Dernière maj : 12/20/2007 09:39:43 »
```

- La méthode **write()** a été utilisée tout au long de ce cours.

6 – JavaScript

Les cookies : définition

- Un « **cookie** » est une chaîne de caractères qu'une page HTML (contenant du code JavaScript) peut écrire à un emplacement UNIQUE et bien défini sur le disque dur du client.
  - Cette chaîne de caractères ne peut être lue que par le seul serveur qui l'a générée.

- Que faire avec un « **cookie** » ?
  - Transmettre des valeurs (contenu de variables) d'une page HTML à une autre.
    - Par exemple, créer un site marchand et constituer un "caddie" pour le client. Ce caddie restera sur son poste et vous permettra d'évaluer la facture au moment de la commande finale sans faire appel à quelque serveur que ce soit.
  - Personnaliser les pages présentées à l'utilisateur en reprenant par exemple son nom en haut de chaque page Web.

6 – JavaScript
Les cookies : limitations

- On ne peut pas écrire autant de cookies que l'on veut sur le poste de l'utilisateur (client d'une page). Il y a des limites :
  - Limites en nombre : Un seul serveur (ou domaine) ne peut pas être autorisé à écrire plus de 20 cookies.
  - Limites en taille : un cookie ne peut excéder 4 Ko.
  - Limites du poste client : Un poste client ne peut stocker plus de 300 cookies en tout.
- Où sont stockés les cookies ?
  - Par exemple, Microsoft Internet Explorer sous Windows stocke les cookies dans des répertoires tels que "C:\WINDOWS\Cookies" ou encore "C:\WINDOWS\TEMP\Cookies"

6 – JavaScript

Les cookies : structure d'un cookie 1/3

- Structure du contenu d'un cookie :
  - **nomCookie=contenuCookie; expires=expdate; path=Chemin; domain=NomDeDomaine; secure**
  - **où :**
    - **nomCookie=contenuCookie;**
      - est la déclaration du nom et du contenu d'un cookie.
      - le caractère de séparation est le ";"
      - il est possible d'ajouter d'autres combinaisons nom=contenu;
      - exemple : nom=Paul; prenom=Emile;

6 – JavaScript

Les cookies : structure d'un cookie 2/3

- Structure du contenu d'un cookie :
  - **nomCookie=contenuCookie; expires=expdate; path=Chemin; domain=NomDeDomaine; secure**
  - **où :**
    - **expires=expdate;**
      - Le mot réservé **expires** suivi du signe "=" (égal). Derrière ce signe, vous mettrez une date d'expiration représentant la date à laquelle le cookie sera supprimé du disque dur du client.
      - La date d'expiration doit être au format GMT (sortie de la fonction **toGMTString())** : Wdy, DD-Mon-YYYY HH:MM:SS GMT
      - Utiliser les fonctions de l'objet **Date**
      - Règle générale : indiquer un délai en nombre de jours (ou d'années) avant disparition du cookie.

6 – JavaScript

Les cookies : structure d'un cookie 3/3

- Structure du contenu d'un cookie :
  - **nomCookie=contenuCookie; expires=expdate; path=Chemin; domain=NomDeDomaine; secure**
  - **où :**
    - **path=Chemin;**
      - **path** représente le chemin de la page qui a créé le cookie
    - **domain=NomDeDomaine;**
      - domain représente le nom du domaine de cette même page
    - **secure**
      - valeur par défaut "**false**" (protocole HTTP) ou "**true**" (HTTPS)
    - Les arguments **path**, **domain** et **secure** sont facultatifs et prennent une valeur par défaut si ils sont omis.

6 – JavaScript

Les cookies : écrire un cookie

- Un cookie est une propriété de l'objet **document** (la page HTML chargée dans le navigateur). L'instruction d'écriture de cookie est :
  - **document.cookie = Nom + "=" + Contenu + "; expires=" + expdate.toGMTString() ;**

- Exemple :

```
var Nom = "MonCookie" ; // nom du cookie
var Contenu = "Vous avez un cookie sur votre disque !"; // contenu du cookie
var expdate = new Date(); // créé un objet date indispensable
// rajoutons lui 10 jours d'existence
expdate.setTime(expdate.getTime() + (10 * 24 * 60 * 60 * 1000));
document.cookie = Nom + "=" + Contenu + "; expires=" + expdate.toGMTString();
```

6 – JavaScript

Les cookies : lecture ou modification d'un cookie

- Pour lire un cookie, il suffit d'accéder à la propriété **cookie** de l'objet **document** (document.cookie).
- Exemple :

```
var LesCookies; // pour voir les cookies
LesCookies=document.cookie; // on met les cookies dans la variable LesCookies
```

- Pour modifier un cookie, il suffit de modifier le contenu de la variable **Contenu** puis de ré-écrire le cookie sur le disque dur du client.
- Exemple :

```
Contenu = "nouveau contenu du cookie..." ; // nouveau contenu
// écriture sur le disque
document.cookie = Nom + "=" + Contenu + "; expires=" + expdate.toGMTString();
```

6 – JavaScript

Les cookies : suppression d'un cookie

- Pour supprimer un cookie, il suffit de positionner la date de péremption du cookie à une valeur inférieure à celle du moment où on l'écrit sur le disque.

- Exemple :

```
// on enlève une seconde (ça suffit mais c'est nécessaire)
expdate.setTime (expdate.getTime() - (1000));

// écriture sur le disque
document.cookie = Nom + "=" + Contenu + "; expires=" + expdate.toGMTString();
```