Cours détaillé sur JAVASCRIPT

1 Introduction au langage JavaScript

Programmation JavaScript : généralités

Javascript est un langage de programmation très 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.
Le code du programme est interprété par le navigateur client (qui reçoit la page). Il ne peut pas être exécuté indépendamment, ce qui le limite comme langage de programmation, contrairement à JAVA (à ne pas confondre !).

C'est un langage basé sur des objets, très simple et conçu, en principe, pour des non spécialistes.
En résumé, voici ses principales caractéristiques :

- JS est un langage de programmation **structurée** qui concourt à **enrichir le HTML**, à le rendre plus "intelligent" et interactif.
- Le code JS est **intégré** complètement dans le **code HTML**, et interprété par le navigateur client
- 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...

Mais c'est un langage limité :

- ce n'est pas un langage de programmation à part entière, indépendant.
- c'est un langage de script, dépendant de HTML, c'est une **extension de HTML**. 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 véritablement un langage orienté objet (pas d'héritage de classe , ni de polymorphisme ..)

Ecriture et exécution du code JS

On peut placer du code JS dans une page HTML à 3 endroits et sous des formes bien différentes.

1. **Entre les balises <SCRIPT>....</SCRIPT>** dans la section d'en-tête, ou dans le corps de la page.

```
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
... code
//code : instructions, déclarations de fonctions,  etc..
</SCRIPT>
</HEAD>
```

- 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 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 la section <HEAD>..</HEAD>. En effet, les fonctions doivent être connues dès le chargement du document, mais ne doivent être exécutées que lors d'un appel explicite de l'utilisateur, le plus souvent en réponse à un événement (voir ci-dessous).

2. **Associé à une balise HTML qui gère un événement**.
Le code JS est généralement inséré sous forme d'un appel de fonction, affectée à un *gestionnaire d'événement* qui apparait comme un nouvel attribut d'un composant de formulaire
L'exécution du code est alors provoquée par appel d'une fonction JS (préalablement déclarée) 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.

Ecriture générale

```
<balise ... onEvenement="code JS" | "fonction JS">
```

où

- *balise* est le nom de certaines balises, souvent des composants de formulaire
- *onEvenement* est un nouvel attribut de la balise
- Bien entendu il faut connaitre les associations entre événements et balises "sensibles" à ces événements.

Exemple

Le code HTML suivant crée un bouton de nom "bouton1", sur lequel est écrit "Calculer". Quand l'utilisateur appuie sur ce bouton, l'événement *onClick* est déclenché et la fonction *calculer()* est appelée.
Son code, déclaré dans l'en-tête est alors exécuté.

```
<HTML>
<HEAD>
 <SCRIPT language="JavaScript">
  function calculer() {
   ....// code.....
  }
 </SCRIPT>
</HEAD>

<BODY>
<FORM>
<INPUT TYPE="button"  VALUE="Calculer" onClick="calculer()">
</FORM>
</BODY>
</HTML>
```

3. **Associé au pseudo-protocole *javascript:* dans une URL**.
Cette pseudo-URL 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:)

Ecriture générale

```
<A HREF="JavaScript:code JS" | "appel fonction JS">texte|image activable</A>
```

L'opérateur **void**
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>
```

Exemple

```
<HTML>
<HEAD>
 <SCRIPT language="JavaScript">
  function calculer() {
   ....// code.....
  }
 </SCRIPT>
</HEAD>
<BODY>
.........
<A HREF="JavaScript:calculer()">Pour calculer</A>
.........
</BODY>
```

Quelques remarques

- JS fait la distinction entre majuscules et minuscules, contrairement aux balises HTML. C'est une source fréquente d'erreur.
- Pour comprendre le code, inclure des commentaires abondants :
  // pour une simple ligne de commentaires
  /* .....*/ pour les encadrer sur plusieurs lignes.
- Quand on ne définit pas de fonctions, on peut inclure le code directement dans la section <BODY>.
- On peut (depuis Netscape 3) placer le code dans un fichier spécifique d'extension **.JS** :
  <SCRIPT LANGAGE=JavaScript SRC=*source.js*>
  </SCRIPT>
  où *source.js* doit être un fichier accessible au moment de l'exécution, dans le répertoire courant ou à une adresse URL précisée.
  Un tel fichier externe permet de réutiliser le code dans de multiples pages, sans avoir à l'inclure dans le source.

Exemple-résumé

```
<html>
<head>
 <script>
   function saluer() {
   alert("Bonjour tout le monde !");
  }
```

```
 </script>
</head>

<body>
<H4>Exécution immédiate</H4>
 <script>
  alert("Bonjour tout le monde !");
 </script>
<H4>Exécution sur événement onClick</H4>
 <form>
  <input type="button" name="Bouton1" value="Salut" onClick="saluer()">
 </form>
<H4>Exécution sur protocole javascript:</H4>
<A HREF = "javascript:saluer()">pour saluer</a>
</body>
</html>
```

Introduction à JavaScript par l'exemple

Voici quelques exemples introductifs et progressifs

- accompagnés de commentaires et
- de corrigés écrits eux-mêmes en JavaScript

I. Afficher du texte :

```
EXEMPLE 1
 <HTML>
  <HEAD> <TITLE> Exemple 1 </TITLE> </HEAD>
   <BODY>
    <SCRIPT LANGUAGE="JavaScript">
     var bonjour = "Bonjour !";
     var question = "Comment allez vous ";
     var phrase = bonjour + "<BR>" + question;
     document.write(phrase, "aujourd'hui ?");
    </SCRIPT>
   </BODY>
 </HTML>
```

A la lecture de ce code JS, comprendre :

- la *déclaration* et *l'initialisation* (c'est-à-dire : donner une valeur initiale) des variables bonjour, question, et phrase.
- l'*affectation* d'une valeur à une variable, avec l'**opérateur =** , sur le modèle : **variable = valeur (ou expression)**
- l'instruction **document.write("texte")** est l'instruction générale d'affichage de "texte" dans le même document (le document "courant").
- les constantes "texte" sont placées entre " "; à défaut JS croit qu'il s'agit d'une variable.
- la *concaténation* (mise "bout à bout") de variables de type texte et de constantes "texte" est réalisée avec l'**opérateur +**
- document.write(...) affiche aussi bien les valeurs des variables que les constantes texte.
  On peut séparer les éléments à afficher par l'**opérateur +** ou plus simplement par la **virgule ,**.

- toutes les balises HTML peuvent figurer comme constantes texte, et sont exécutées comme du code HTML.
  Ainsi, "
" inséré dans la valeur de la variable **phrase**, fera passer à la ligne suivante la suite du texte à afficher.

Comprendre le script et prévoir le résultat de son exécution.
Pour vérifier, appuyer sur ce bouton :

II. Afficher une image

```
EXEMPLE 2
<HTML>
<BODY> <CENTER>
<SCRIPT language = "JavaScript" >
document.write("<IMG SRC='../../images/lycee2.jpg'> <BR>  <H1>
                                        Une photo de mon lycée</H1><P>");

document.write("<IMG SRC='../../images/monchat.gif'> <BR>  <H1>
                                        et de mon chat .... </H1>");
</SCRIPT>
</CENTER>
</BODY>
</HTML>
```

Comprendre le script, en particulier le rôle du code HTML inséré et prévoir le résultat de son exécution.

III. Afficher la date et l'heure

```
EXEMPLE 3
<HTML>
<BODY>
 <SCRIPT language = "JavaScript" >
var date = new Date();
          // déclaration d'une variable de type Date
          // la variable date contient alors la date courante
var mois = date.getMonth() + 1;
          // la variable mois contient le N° du mois à partir de 0 (à 11)
var jour = date.getDate() ;
var annee = date.getYear();
if (navigator.appName =='Netscape')
  annee = annee + 1900 ;
          // getYear() donne le numéro de l'année
       // pour netscape et Mozilla à partir de 1900
document.write("<Date> " , jour, " /  ", mois, " / "+annee, "<BR>");
document.write("<Heure>  ", date.getHours(), " :  ",date.getMinutes(),
                    " : ",date.getSeconds() );
</SCRIPT>
</BODY>
</HTML>
```

A la lecture de ce code JS, comprendre :

- La déclaration *var date = new Date();* déclare un objet appelé *date*, de **type Date**, initialisé à la date du système.

IV. Calculer la somme des 100 premiers naturels non nuls :

```
EXEMPLE 4-1
<HTML>
 <HEAD> <TITLE> Exemple 4</TITLE> </HEAD>
 <BODY>
  <SCRIPT LANGUAGE="JavaScript">
  var somme=0;
  var nombre= 100;
  for (i=1; i <=nombre ; i++)
     somme=somme + i ;
  document.write("La somme des premiers naturels jusqu'à " + nombre +
              " est égale à " + somme+  "<BR>");
  document.write( "Pour vérifier : ("+nombre+ " * "+(nombre+1)+" )/2 = "+somme);
  </SCRIPT>
 </BODY>
</HTML>
```

A la lecture de ce code JS, comprendre :

- l'écriture générale d'une structure répétitive de type FOR
- i++ signifie i = i +1

Comprendre le script et prévoir le résultat de son exécution.

Pour vérifier :

```
EXEMPLE 4-2 (modification de l'exercice 4-1)
<SCRIPT LANGUAGE="JavaScript">
   var somme=0;
   var nombre= 100;
   for (i=1; i <=nombre ; i++) {
     somme=somme + i ;
     document.write("Pour i = ", i,  "  ---> somme = " , somme , "<BR>");
}
</SCRIPT>
```

Ici remarquer les accolades { ...} indispensables pour englober plusieurs instructions (séparées par des ;) dans l' instruction répétitive FOR.
Qu' obtient-on pour l'exercice 4-2, où (contrairement à l'ex 4-1) l'instruction d'affichage est *incluse dans l'instruction FOR* ?

V. L'événement OnClick sur un bouton de formulaire

Considérons un composant de type **bouton**
Un clic sur le bouton provoque un événement **OnClick** auquel le programmeur peut rattacher du code JS (le plus souvent une fonction).
Dans l'exemple qui suit, la fonction **ALERT()** est appelée par le clic. Son rôle est d'afficher une boîte avec un message.

```
EXEMPLE 5
<HTML>
 <HEAD> <TITLE> Exemple 5 du cours JS1</TITLE>
 < /HEAD>
```

```
<body>
<form>
<input type="button" value="Cliquez" onClick="alert('Coucou, c\'est moi !')">
</form>
</body>
</HTML>
```

VI. Boîte de dialogue et instruction conditionnelle

alert() affiche une boite de message simple et attend la validation ou Echap.

***reponse* = prompt("texte","chaine par défaut")** affiche "texte" et une ligne de saisie dans une boîte.
La poursuite du programme est bloquée tant que l'utilisateur n'a pas saisi une chaîne de caractères validée.
Prompt(..) a pour résultat la chaine saisie ou sinon, la "chaine par défaut" (si elle doit être vide, mettre ""). Cette chaine est donc affectée ici à la variable nommée *reponse*

Exemple :

```
EXEMPLE 6
<HTML>
 <HEAD> <TITLE> Exemple 6</TITLE> </HEAD>
 <BODY>
  <SCRIPT LANGUAGE="JavaScript">
  var prenom="" ,  bahut="" ;
  prenom=prompt( "Votre prénom","");
  bahut=prompt( "Le nom de votre établissement","");
  if ( prenom !=""  && bahut !="")
   document.write("Bonjour ", prenom," ! <BR> Vous plaisez-vous à ", bahut, " ?")
  else
   alert("saisie incomplète !");
  </SCRIPT>
</BODY>
</HTML>
```

Lire attentivement ce code JS. Voici quelques informations :

- Dans l'instruction conditionnelle, la condition qui suit **if** doit obligatoirement être mise entre parenthèses.
- principaux opérateurs de comparaison :
  - **==** (égalité, à ne pas confondre avec l'affectation =)
  - **!=** (différent)
  - Autres : > , < , >= , <=
- principaux opérateurs logiques
  - **&&** : et (utilisé ici pour composer 2 conditions simples).
  - **||** : ou (le caractère | est obtenu au clavier avec *Alt Gr* -)
  - **!** : non.

S'efforcer de décrire pas à pas la signification et le résultat des lignes de code.

VII. Fonctions et procédures

Les fonctions doivent être déclarées dans la section <HEAD>... </HEAD>, entre les balises <SCRIPT> </SCRIPT>, pour être comprises avant toute utilisation.
Elles sont appelées habituellement dans le corps de la page entre <BODY>... </BODY>

- soit à partir d'une section de code <SCRIPT>... </SCRIPT> (voir exemple VII et exemple VIII)
- soit affectées à une procédure d'événement liée à un composant d'un formulaire (voir exemple IX).

Leur nom est introduit par le mot **function**. L'identificateur de la "function" est obligatoirement suivi de parenthèses.
Entre ces parenthèses on écrit éventuellement la liste des paramètres (cf exemple VIII).

JS ne fait apparemment pas de distinction entre fonctions et procédures.

- Les **procédures** sans paramètre s'écrivent :

```
function nom-fonction () {
 séquence d'instructions
}
```

- Les **fonctions** fournissent un résultat :

```
function nom-fonction () {
 séquence d'instructions
 return paramètre-résultat
}
```

Voici un exemple simple de procédure sans paramètre :

```
EXEMPLE 7
<HTML>
<HEAD>
<TITLE> Exemple de fonction</TITLE>
   <SCRIPT language="JavaScript">
   function bonjour() {
   document.write("Bonjour",<BR>);
   }
   </SCRIPT>
</HEAD>
<BODY>
  <SCRIPT
  bonjour();
  </SCRIPT>
<FORM>
<input type="button"  value="Salut" onClick="alert('Et bon courage à tous ' )">
</FORM>
</BODY>
</HTML>
```

A la lecture de ce code JS, noter et retenir :

- la place de l'appel de la fonction **bonjour()**

- elle effectue une **action**, afficher *Bonjour !* , **sans calculer une valeur résultat.** (l'absence d'instruction *return* caractérise une *procédure*).

VIII. Procédures et fonctions avec paramètre(s)

Les *procédures* et *fonctions* avec paramètre(s) sont déclarées conformément au paragraphe précédent.
Ils en différent par la présence d'une liste de paramètres placés entre parenthèses.
Bien sûr, ces paramètres doivent recevoir des valeurs correctes qui leur sont affectées lors de l'appel de la fonction.

```
Function nom-fonction (liste de paramètres formels) {
 séquence d'instructions
 [return paramètre-résultat ]
}
```

Voici l'exemple simple d'une fonction à un paramètre :

```
EXEMPLE 8
 <HTML>
  <HEAD> <TITLE> Exemple 8</TITLE>
    <SCRIPT LANGUAGE="JavaScript">
    function somme_N_entiers (N) {
    var somme=0;
      for (i=1; i <=N ; i++) {
      somme=somme + i ;
      document.write("Pour i = ", i,  "---> somme = " , somme , "<BR>");
      }
    return somme
    }
  </SCRIPT>
  </HEAD>

<BODY>
<SCRIPT>
var nombre=  prompt("Somme jusqu'à N = ", 10);
document.write("Somme des 100 premiers entiers non nuls  = ",
somme_N_entiers(nombre));
</SCRIPT>
</BODY>
</HTML>
```

A la lecture de ce code JS, comprendre :

- l'appel de la fonction n'est pas ici provoqué par un événement du genre **onClick** (cf exemple V), mais par *l'affichage de son résultat demandé* par l'instruction **document.write**
- ce résultat est l'*image de la valeur* de **nombre** par la fonction **somme_N_entiers**, c'est-à-dire la valeur du paramètre **somme** qui est calculée par le code de la fonction.
- la fonction est définie avec un paramètre **formel** dont la valeur n'est connue qu'au moment de l'appel. Il y a alors transmission de la valeur du paramètre d'appel et affectation au paramètre formel.

- bien comprendre que dans ces conditions, la définition d'une fonction informatique est la description formelle d'un calcul ou d'une suite d'instructions.
- l'appel, c'est-à -dire l'exécution de la fonction avec la valeur du paramètre transmis, s'apparente au calcul de f(xo), *l'image de la valeur xo* attribuée à la variable x, par une fonction mathématique f.

IX. Fonction appelée par un événement Utilisation de formulaire

```
EXEMPLE 9
<HTML>
<HEAD> <TITLE> Exemple 9</TITLE>
   <SCRIPT LANGUAGE="JavaScript">
   function somme_N_entiers (nb) {
   var somme=0;
   for (i=1; i <=nb ; i++) {
    somme=somme + i ;
    document.write("Pour i = ", i,  "---> somme = " , somme , "<BR>");
   }
   return somme;
   }
 </SCRIPT>
</HEAD>

<BODY>
<H3><CENTER>Somme des N premiers entiers non nuls</CENTER></H3>
 <FORM name=formulaire>
 <!--  attention,  ce qui suit n'est pas une affectation ! -->
  Nombre = <INPUT type=text name=nombre value=10>
  <INPUT type=button  name=bouton value="Calculer"
  onClick="formulaire.resultat.value=somme_N_entiers(formulaire.nombre.value)">
  Résultat = <INPUT type=text name=resultat value="">
 </FORM>
</BODY>
</HTML>
```

- Dans ce dernier exemple, proche du précédent VIII, on reprend la fonction **somme_N_entiers(N)**, en enlevant l'affichage contenu dans l'itération.
- Cette fonction est simplement chargée de donner le résultat du calcul contenu dans la variable **somme**, grâce à l'instruction **return somme**
- Les nouveautés résident dans la façon de dialoguer avec le programme; pour cela on utilise des composants de formulaire.
  - entrée de la valeur du paramètre **N** dans un composant de type champ de texte, nommé **nombre**, avec une valeur par défaut égale à 10. appel de la fonction provoqué par l'événement **onClick** sur le bouton nommé **bouton**.
  - **somme_N_entiers( )** utilise un paramètre dont la valeur est contenue dans le champ **nombre**
  - cette fonction calcule un résultat appelé **somme**qui est affectée à la valeur d'un autre composant de type texte, nommé **resultat**, et qui est chargé de l'affichage.
  - remarquer que tout cela est inclus dans du code HTML, et ne nécessite pas de section SCRIPT
  - en revanche, comme on ne dispose pas de variables JS, on fait référence aux valeurs des contenus de ces composants par l'intermédiaire de leur propriété **value**. Il ne faut pas confondre la valeur et le nom du composant, donné par la propriété **name**.

- bien comprendre la façon de noter la référence à ces valeurs :
  par exemple, **dialogue.nombre.value** signifie :
  "la **valeur** du composant nommé **nombre** situé dans le formulaire nommé **dialogue**".

2 Les structures de données en JavaScript

Les données constantes

JS fournit les constantes suivantes, directement disponibles :

- les constantes numériques : entiers, "réels" écrits en notation décimale, ou flottante (c-à-d en notation scientifique, par ex : 2718E-3)
- les 2 constantes booléennes : *true* ou *false*
- les chaînes de caractères, entourées indifféremment de guillemets " " ou d'apostrophes ' ' (à privilégier).
- la constante spéciale **null** signifie "rien", "absence de valeur". Cette "valeur" est attribuée à toute variable utilisée sans être définie (par exemple prompt() retourne null si on sélectionne le bouton *Annuler*)

Manipulation des chaines de caractères

- JavaScript facilite beaucoup l'affichage des résultats en convertissant automatiquement les entiers en chaînes de caractères, ce qui permet de concaténer des nombres avec des chaînes de caractères (transtypage automatique).
- Dans l'instruction d'écriture dans le document courant, document.write( ), les données à afficher peuvent être séparées par des virgules ou des +.(la concaténation par l'opérateur + est recommandée)
- Des caractères spéciaux peuvent aussi être insérés dans les chaînes : \b (retour arrière), \f (saut de page) , \n (nouvelle ligne) , \r (Entrée), & (tabulation); \' pour une apostrophe
- On peut insérer des codes HTML sous forme de chaines, qui seront bien interprétées à l'exécution comme de véritables balise, et non par affichées telles quelles.

Exemple

```
<script>
document.write("Voici la valeur approchée à ", 1E-3,
            " près de la constante e : ", 2718E-3);
document.write("<P>");
document.write("Mon <i>lycée</i> est situé en
<u>Seine-Saint-Denis</u> dans la ville d\'Epinay-sur-Seine");
</script>
```

Voici la valeur approchée à 0.001 près de la constante e : 2.718

Mon *lycée* est situé en Seine-Saint-Denis dans la ville d'Epinay-sur-Seine

(Attention ! le symbole "exposant" E doit être collé au dernier chiffre.)

Déclaration et affectation de variables

- Types de variables
  On distingue 5 types de variables en JS.
  Les nombres : number, les chaines : string, les booléens : boolean, les objets : object et

les fonctions : function
La fonction **typeof()** appliquée à une variable retourne son type.

```
var chaine = 'bonjour';
document.write(typeof(chaine));
```

En voici l'exécution :

```
string
```

- Notation : **var nom = valeur**
  Déclaration de la variable ***nom*** initialisée avec la valeur ***valeur***; son type est alors celui de la valeur.
  - Le mot **var** est facultatif, mais recommandé.
  - Les variables doivent commencer par une lettre ou par _
  - JS distingue minuscules et majuscules.
    Les variables suivantes *mavariable, maVariable, Mavariable, MaVariable* sont toutes distinctes.
  - Le symbole **=** est réservé à l'affectation; le symbole de comparaison (égalité) se note == (2 symboles = collés).
  - JS attribue *automatiquement* à la variable le type de la valeur affectée.
    Sinon JS lui attribue une valeur et un type indéterminés "**undefined**"

```
var x ;
document.write('Voici la valeur de x : x = '+ x+ "<BR>" );
document.write("et son type  : typeof(x) = " + typeof(x) );
```

En voici l'exécution :

```
Voici la valeur de x : x = undefined
et son type : typeof(x) = undefined
```

- Attention
  L'attribution implicite de type, à première vue pratique, risque d'engendrer confusion et absence de rigueur.
  Il faut s'habituer à **déclarer** les variables et à les **initialiser** (c'est-à-dire leur donner une valeur initiale au moment de leur déclaration, ce qui fixe leur type.
  Par exemple
  nomLycee = "J. Feyder" ; TauxTva = 20.6 ;

  Eviter absolument ce qui suit (c'est compris par JS, mais peu recommandable !)

```
var x = 12.5;
document.write('Ici x est une variable "réelle" : x = '+ x+ "<BR>" );
x = "Bonjour !" ;
document.write("Maintenant  x est une chaîne de caractères : x = " + x  );
```

En voici l'exécution :

```
Ici x est une variable "réelle" : x = 12.5
Maintenant x est une chaîne de caractères : x = Bonjour !
```

Affectation de variables

- Mécanisme de l'affectation
  variable = valeur (de l'expression, de même type)
  Exemple à étudier :

```
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 ;      // nouvelle valeur de b = précédente valeur de b + valeur de a
document.write( " a= "+a + " ;  b= "+b);
```

En voici l'exécution :

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

Portée des variables

- Portée d'une variable
  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.
  Exemple :

```
<HEAD>
<script>
var lycee="J-Feyder";
var ville="Epinay-sur-Seine";
document.write("Mon lycée est situé dans la ville d\'"+ville+"<BR>");
var date = new Date();
</script>
</HEAD>
</BODY>
<script>
document.write("Il porte le nom du cinéaste "+lycee+"<BR>");
document.write("Nous sommes au mois N°",date.getMonth() + 1,"<BR>");
</script>
</BODY>
```

En voici l'exécution :

```
Mon lycée est situé dans la ville d'Epinay-sur-Seine
Il porte le nom du cinéaste J-Feyder
Nous sommes au mois N°12
```

Construction des expressions

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
- expressions **chaînes de caractères**
  - **+** opérateur de concaténation (mise bout à bout) de deux chaînes
  - var aujourdhui = " Lundi " + 3 + " novembre" + 1997;
  - document.write(aujourdhui, "
");
  - **+=** ajoute la chaîne de droite à la chaîne de gauche

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

  - En voici l'exécution :

Bonjour tout le monde !

- expressions **booléennes** ou logiques
  - opérateurs de comparaison : == (égal, même valeur) , != (différent) , > , >= , < , <=
  - 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 = true) ? "bonjour" : "au revoir"

Tableaux JS

Un tableau 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 (chaine 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.

Déclaration

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.

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);

Utilisation

Les éléments d'un tableau de taille *dim*, sont indicés à partir de 0 jusqu'à dim - 1.
Exemples à étudier :

var MonTableau = new Array(8)
Ces 8 éléments sont nommés MonTableau[0] , ... MonTableau[7]
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] )
  }
document.write("</UL>");
```

En voici l'exécution :

Voici les 4 saisons :

- printemps
- été
- automne
- hiver

Tableaux à plusieurs dimensions

Il faut les gérer 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> " );
}
```

1 2 3
4 5 6
7 8 9

Propriétés

- **length** donne le nombre d'éléments.
  var mois= new Array(12);
  document.write("Il y a ", **mois.length**, "mois dans l'année");
  var NbMois = mois.length ; document.write(" partagés en ", NbMois / 3, " trimestres");

Méthodes

- **reverse()** change l'ordre des éléments

- **sort()** trie suivant l'ordre croissant, ou suivant le modèle indiqué en paramètre

Particularités et bizarreries ..

- **var tab = new Array()** crée un objet vide, de type tableau, sans fixer sa dimension
- **var tab= new Array("bonjour", charge_e, 1.6E-19, "C")** est accepté et crée un tableau hétéroclite de 4 éléments

```
var unLepton="un électron";
var tab=  new Array(unLepton, " porte une charge négative égale à ",  1.6E-19," C");
 for( i=0; i<tab.length; i++)  document.write(tab[i] );
```

- En voici l'exécution :

```
un électron porte une charge négative égale à 1.6e-19 C
```

- On peut aussi créer un tableau directement "en extension", sans faire appel au constructeur Array(). On liste les valeurs des éléments dans **[...]**.
  Exemple : les 2 définitions suivantes sont équivalentes

```
var Les4saisons = new Array("printemps", "été", "automne", "hiver");
var Les4saisonsbis = ["printemps", "été", "automne", "hiver"];
document.write("Voici les 4 saisons : <br>")
for (i=0 ; i<Les4saisonsbis.length ; i++)
  document.write(Les4saisonsbis[i], "  " )
```

En voici l'exécution :

```
Voici les 4 saisons :
printemps été automne hiver
```

3 Les structures de contrôle en JavaScript

Structure algorithmique d'un programme

Un programme informatique est assemblé à partir de 3 catégories principales d'instructions, quel que soit le langage utilisé pour le coder :

- instructions séquentielles
- instructions conditionnelles (ou alternatives)
- instructions itératives (ou répétitives)

Nous allons voir en parallèle les notations algorithmiques et leur traduction en JS.

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'instructions	
algorithme	**Code JavaScript**

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

L'instruction conditionnelle if ... [then] .. else

Syntaxe

Structure conditionnelle	
algorithme	**Code JavaScript**

SI condition
ALORS Séquence 1
SINON Séquence 2
FINSI
	if (condition) {
	séquence 1
	}
	else {
	séquence 2
	}

Remarques

1. La condition doit être **toujours entourée de ( )**
2. Le mot *alors* (**then**) est toujours sous-entendue.
3. La séquence *alors* est exécutée si la condition est vraie.
4. La séquence *else* (optionnelle) est exécutée si la condition est fausse.
5. les {} ne sont pas obligatoires qu'en cas d'instructions multiples.

Exemple 1-1

```
<SCRIPT language = "JavaScript" >
 var prixHT =150 ; prixTTC=0;
 if ( prixHT == 0  )
   alert("donnez un prix HT !");
   else
   prixTTC = prixHT * 1.206;
 document.write("Prix HT = " + prixHT + "<BR>");
 document.write("Prix TTC = ", prixTTC, "<BR>");
</SCRIPT>
```

Conditionnelles imbriquées; exemple 1-2

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

```
<SCRIPT language = "JavaScript" >
 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>
```

L'itération contrôlée FOR

L'instruction for permet de répéter une séquence d'instructions tant qu'une condition est vraie;

syntaxe

Structure itérative FOR	
algorithme	**Code JavaScript**

POUR I de *valeur initiale*
à *valeur finale*
Répéter
Séquence d'instructions
 	
FIN POUR
	for (*valeur initiale*; *condition*; *poursuite*) {
	
	séquence d'instructions
	
	}

La condition qui suit **FOR** est composée de 3 éléments :

1. une valeur ou expression initiale portant sur une variable entière appelée compteur.
2. une condition : tant qu'elle est vraie, la répétition est poursuivie.
3. une expression de poursuite qui consiste en la mise à jour du compteur.

Exemple 2

```
<SCRIPT language = "JavaScript" >
document.write("Table des carrés<BR>");
for (var i = 0; i <15; i++) {
   document.write("i = "+i+"        i² = "+ i*i+"<BR>");
   }
</SCRIPT>
```

Exemple 3

```
<SCRIPT language = "JavaScript" >
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 100 nombres au hasard<BR>");
max= prompt("Nombres au hasard de 1 à ","100");
tab = new Array(100);
for (var i = 0; i <100; i++) {
   tab[i]= hasard(max);
   }
for (var i = 0; i <100; i++) {
 document.write("tab [", i,"] = ", tab[i],"    ");
  }
</SCRIPT>
```

- Les accolades sont obligatoires s'il y a plusieurs instructions.
- Chaque instruction se termine par un point-virgule ;
- La séquence sera répétée tant que la condition est vraie, compte-tenu 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 entrainer des "boucles infinies", comme par exemple :
  for (i= 11 ; i>10 ; i ++) { .... }

Variante

for (variable **in** objet) {

séquence instructions
}

Exemple 4

```
<SCRIPT language = "JavaScript" >
function hasard(N) {
 return Math.floor(Math.random()*N)+1;
}
document.write("Tableau de 100 nombres au hasard<BR>");
max= prompt("Nombres au hasard de 1 à ","100");
tab = new Array(1000);
for (var i = 0; i <100; i++) {
    tab[i]= hasard(max);
    }
for (var i  in tab ) {
 document.write("tab [", i,"] = ", tab[i],"    ");
  }
</SCRIPT>
```

Les instruction break et continue

L'instruction break permet de sortir de l'itération ou de la séquence en cours.
L'instruction continue sort de la séquence en cours puis reprend à l'itération suivante.

L'Itération WHILE (TANT QUE)

L'instruction répétitive while permet de répéter une séquence d'instructions tant qu'une expression est vraie.

syntaxe

Structure itérative WHILE	
algorithme	**Code JavaScript**

TANT QUE *(condition est vraie)*
REPETER
Séquence d'instructions
FIN TANT QUE
	WHILE (*condition*) {
	séquence d'instructions
	}

Exemple 5

```
<SCRIPT language = "JavaScript" >
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 = 0 ;
 while ( a != max ) {
  a = hasard(max);
  tab[i]= a;
  i ++
  }
 document.write(max, ' a été obtenu au ', i, 'ème tirage <P>') ;
 i=0;
```

```
while ( tab[i] != null ) {
  document.write("tab [", i,"] = ", tab[i],"---");
  if (i %5 ==0 && i!=0) document.write("<br>");
  i ++;
 };
</SCRIPT>
```

4 Les classes prédéfinies de JavaScript

Généralités sur les objets en JS

- JS n'est pas un vrai langage orienté-objet. Des concepts primordiaux ne sont pas implémentés, notamment l'héritage. 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 étudirons les objets du navigateur, puis enfin nous apprendrons comment définir nos propres classes d'objets en JavaScript.
- **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 :
  1. les propriétés (ou attributs) qui décrivent ses caractéristiques
     Une **propriété** est une variable qui est attachée à un type d'objets, et qui est contient une de ses caractéristiques.
  2. les méthodes (procédures ou fonctions) qui décrivent ses comportements et ses actions
     Une **méthode** est une fonction qui ne s'applique qu'à une classe d'objet.
- 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 :
  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()

Classes d'objets prédéfinis

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

I. Math

Les fonctions mathématiques usuelles doivent être préfixées par le nom de l'objet **Math**, desquelles elles dépendent. Ce sont les "**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.

Pour alléger les notations, on peut *"factoriser"* le préfixe **Math** dans une séquence de calcul mathématique, comme dans l'exemple suivant :

```
<script>
r=10;
with (Math ) {
s = PI * pow(r , 2);
theta = PI / 3;
x = r * cos( theta );
y = r * sin( theta);
document.write("PI = ",PI, "<br>");
document.write("s = ", s,"<br> Coordonnées de M : x = ",x, "       y = ",y)
}
</script>
```

En voici l'exécution :

```
PI = 3.141592653589793
s = 314.1592653589793
Coordonnées de M : x = 5.000000000000001 y = 8.660254037844385
```

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), fct 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.

Conversion chaine <--> nombre (entier ou flottant)

- Fonctions parseInt(), parseFloat() et eval() récupère les chaines de caractères passée en paramètre et s'efforce de les évaluer numériquement : voir le paragraphe suivant **String** spéciale **isNaN( )** évalue l'argument pour déterminer s'il s'agit d'un nombre ("NaN" = **Not** a Number). Elle retourne alors TRUE ou FALSE.
  Exemple d'utilisation pour controler la saisie d'un entier :
- <SCRIPT LANGUAGE="JavaScript1.1">
- function testnum(chnum) {
- num=parseInt(chnum);
- if (isNaN(num))
- alert(chnum+ 'n\'est pas pas un entier !');
- }
- </SCRIPT>
- Exercices : Faire afficher un tableau de valeurs pour la fonction ln(x) dans [-10, 10]

II. String

Déclaration

```
var chaine = "<B>Bonjour !</B>";
```

```
document.write ("La longueur de la chaine ",chaine, " est : ",
        chaine.length,". Pourquoi ?<br>");
```

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

var *chaine* = "**<B>Bonjour !</B>**" crée une variable nommée *chaine* et lui attribue :

1. le type String.
2. la valeur "Bonjour !" avec l'attribut **gras**

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 chaine.
- L'opération + concaténe (mise à la suite) 2 chaines pour en former une nouvelle

```
var info = '\"informatique\" ';
var monOption = "l\'option " + info ;
document.write("J\'enseigne encore " + monOption) ;
```

J'enseigne encore l'option "informatique"

Quelques fonctions (ou méthodes)

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

```
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);
for (var i=0; i<255 ; i++)
  document.write("En décimal i = "+i+ " s'écrit "+ i.toString(16)+"
")
```

a = 5007
b = 12
c = 3.1416
En décimal d= 255 s'écrit ff

- Exercice d'entrainement au calcul mental
- **chaine.toUpperCase()**, pour mettre chaine en majuscule.
  Exemple :

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

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

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

```
chaine.substring( 2,6) = jour
```

- **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 :<br>");
for (i=0 ; i< chaine.length ; i++)
document.write (chaine.charAt(i), " - ");
```

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

- **chaine.indexOf(s_ch)** donne la 1ère position du caractère de chaine égal au 1er caractère de s_ch.
  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>");
```

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

- La méthode **chaine.split(séparateur)**
  Appliquée à un texte, elle fournit un tableau de chaines dont les éléments sont les sous-chaines extraites suivant le séparateur.
  Exemples
  Chaine initiale : jules.toto@mail.ac-creteil.fr
  découpage suivant "@" :
  tab_ch1[0] = jules.toto
  tab_ch1[1] = mail.ac-creteil.fr
  découpage suivant "." :
  tab_ch1[0] = jules.toto
  tab_ch1[1] = mail.ac-creteil.fr

II. Exercice

Très classique ! traduction Français ---> Javanais

III. Date

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 millisecondes.

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 .

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) pour obtenir par exemple : *December 25, 1995 13:30:00*)
- Date(année, mois, jour), une suite convenable de 3 entiers, par exemple (2000, 0, 1)
- une suite de 6 entiers (année, mois, jour, heures, minutes , secondes), (même exemple : 95, 11, 25 , 13, 30 , 00

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
- **to...()** : pour retourner une chaîne de caractères correspondant à l'objet 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 numéro du mois, le N° du jour dans le mois, et les heures, minutes et secondes.
- getDay() donne le N° du jour de la semaine (le 0 tombe le dimanche)
- getTime() donne le nombre de millisecondes écoulées depuis le 1/1/1970, très pratique pour calculer des intervalles entre 2 dates.

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 );
```

Nous étions le 24/11/1981
Nous sommes le 3/12/3907

Exercice

Nous avons passé l'an 2000 .. depuis combien de jours ?

5 Procédures et fonctions

Déclaration et appel des fonctions en JS

- 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

  ....

- 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.
- De plus, 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.

Déclaration générale d'une procédure et d'une fonction

```
<HEAD>
<SCRIPT LANGUAGE=JavaScript >
Function nomProcédure(param1, param2, ...)  {
  séquence d'instructions;
  }
Function nomFonction(param1, param2, ...)   {
  séquence d'instructions;
  return nom_variable
  }
</SCRIPT>
</HEAD>
```

Appel d'une procédure et d'une fonction

- 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**.
- Dans l'écriture de l'appel 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(...).

```
nomProcédure(valeur1, valeur2, ...) ;

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

Exemple

Etudier l'exemple suivant et prévoir exactement son exécution.

```
<HEAD>
<SCRIPT>
  // déclaration de la procédure
```

```
 function bonjour(prenom) {
 document.write("Bonjour, comment vas-tu ", prenom," ?<br>");
 }
 // déclaration de fonctions
function volumeSphere(rayon) {
 return 4/3*Math.PI*Math.pow(rayon,3);
 }
function calculerPrix(PrixUnitaire, NbArticles)  {
 return  PrixUnitaire* NbArticles;
 }
</SCRIPT>
</HEAD>
<BODY>
// appel de la procédure
 bonjour("Toto") ;
//appels des fonctions
var montant=CalculPrix( 150 , 4) ;

document.write( "Tu dois payer  ", calculerPrix( 150, 4), "F.<BR>");
document.write( "Sais-tu que le volume de la sphère de rayon unité est ",
            volumeSphere(1)," ?<BR>" );
</BODY>
```

Visibilité des paramètres

- De façon générale, les paramètres formels d'une fonction ne sont connus qu'à l'intérieur de la fonction.
  De même, les **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
- Examiner l'exemple suivant : qu'obtient-on exactement à l'exécution ?

```
<HEAD>
<SCRIPT LANGAGE=JavaScript >
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("à 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>
```

- On ajoute à la fin l'instruction suivante :

```
document.write("après appel de la fonction : "+ ch +" bonjour  "+nom +"<BR>");
```

Fonctions récursives

JavaScript est un langage récursif !
Exemple classique : la **fonction factorielle**

```
<HTML>
<BODY>
<script>
function fact(n) {
if (n==0) return 1
else return (n*fact(n-1))
}
 nb=prompt("N= ","0");
 document.write("Liste des premières factorielles jusqu'à ", nb);
 for (var i=0;i<nb;i++)
 document.write(i+ " ! = "+fact(i)+"<br>")
</script>
</BODY>
</HTML>
```

L'objet Function

- A chaque fonction rencontrée, JavaScript construit un tableau de ces arguments, appelés **arguments**.
  Si la fonction s'appelle calculer, ce tableau a :
  - pour éléments **calculer.arguments[i]**
  - pour taille **calculer.arguments.length**
- Exemple
- function calculerPrix(PrixUnitaire, NbArticles) {
- if (calculerPrix.arguments.length !=2)
- alert("impossible de calculer le prix !")
- else
- return PrixUnitaire* NbArticles;
- }

Dans le cas où l'on ne passe pas 2 paramètres effectifs lors de son appel, cette fonction affiche un message d'alerte et renvoie la valeur undefined
On peut définir aussi une fonction dynamiquement, en cours d'exécution, et avec un nombre d'arguments non précisés au départ !

- Ce nombre d'arguments variables (et non défini à la déclaration), trouve son utilité dans le cas où ces arguments doivent être traités comme les éléments d'un tableau

```
function somme() {
var resultat=0 ;
for (var i=0; i < somme.arguments.length; i++)
  resultat += somme.arguments[i] ;
return resultat ;
}
// quelques appels
document.write("1+2+3 = somme(1,2,3) = " + somme(1,2,3)+"<p>");
document.write("1+2+3+ .. +10 = somme(1,2,3,4,5,6,7,8,9,10) = "
                                    + somme(1,2,3,4,5,6,7,8,9,10)+"<p>");
document.write(" somme(15.4) = " + somme(15.4)+"<p>");
document.write(" somme() = " + somme()+"<p>");
```

- **Création d'instance de la classe Function**

Il existe un constructeur Function(), semblable aux constructeurs Date(), Image(), permettant de définir une nouvelle fonction, lors de l'exécution.
Syntaxe :

- **nom_fct=Function("arg1", .. "argN", "code_fct")**, où :
- arg1, .. argN est la liste des paramètres de la fonction
- code_fct est le code, donc la suite des instructions, séparées par des ;

Exemple :
Les 2 fonctions f1 et f2 sont équivalentes :

```
<script>
function f1(x) {
return x*x
}
var f2 = new Function("x","return x*x")
document.write("f1(12) =" + f1(12) + "<BR>")
document.write("f2(12) =" + f2(12) + "<BR>")
</script>
```

Formulaires dans une page HTML

Présentation

Jusqu'ici, les pages HTML sont des documents passifs, à consulter.
L'utilisateur parcours des pages, mais ne peut pas écrire, fournir des informations, remplir un formulaire et le transmettre, comme cela se fait couramment par Minitel.
Les seules initiatives attendues de la part de l'utilisateur sont des choix de "navigation" qui lui sont proposés sous forme d'hyperliens cliquables.

Les **formulaires** HTML (**FORMS** en anglais) sont des ensembles de **composants** , appelés aussi **champs** qui permettent à l'utilisateur d'entrer des informations, d'exprimer ses choix, de saisir du texte ...

Bien sûr les réactions à apporter à ces informations doivent être prévues à l'avance, et donc doivent être programmées.
Les programmes en général placés sur le serveur gérent le dialogue avec le client- utilisateur.
Le langage JavaScript, plus limité, permet l'insertion de petits programmes dans les pages HTML, les rendant ainsi interactives. Il est ainsi possible d'effectuer des traitements à partir des données entrées par l'utilisateur.

La balise FORM

Toute la partie formulaire de la page doit se trouver entre les marqueurs **<FORM> ....</FORM>**
Cette balise sert de conteneur pour accueillir les éléments ou composants du formulaire, comme par exemple un bouton, une ligne de texte, une liste de choix ..
Chacun de ces composants s'y trouve déclaré et créé individuellement par une balise qui commence généralement par **INPUT** ou par **SELECT**
Voici un exemple significatif qui regroupe la plupart des catégories de composants que peut contenir une formulaire HTML.
Chaque composant sera ensuite étudié en détail.

Un exemple complet

```
<H2><u><CENTER>Fiche de renseignements</CENTER></u></H2>
```

```
<FORM  NAME="Fiche_inscription">
Nom : <INPUT TYPE ="text" NAME="Nom" VALUE="" >
Prénom : <INPUT TYPE ="text" NAME="Prenom" VALUE="" >
Mot de passe : < INPUT TYPE="password" NAME="Pass" SIZE=5><P>
Vous êtes en terminale S à dominante :
<INPUT TYPE="radio" NAME="option" VALUE="math" checked>Mathématiques
<INPUT TYPE="radio" NAME="option" VALUE="pc">Physique-chimie
<INPUT TYPE="radio" NAME="option" VALUE="svt">SVT
<P>

Quelle est votre discipline préférée ?
<SELECT NAME="disciplines"  MULTIPLE SIZE=3>
  <OPTION>Physique
  <OPTION SELECTED>Informatique
  <OPTION>Philosophie
  <OPTION>Mathématiques
  <OPTION>Langues
  <OPTION>SVT
  <OPTION>Histoire
  <OPTION>EPS
</SELECT>
<P>

Vous vous êtes inscrit(e)s en : <BR>
<INPUT TYPE="checkbox" CHECKED NAME="DEUG" >DEUG<BR>
<INPUT TYPE="checkbox" NAME="CPGE">CPGE<BR>
<INPUT TYPE="checkbox" NAME="BTS">BTS<BR>
<INPUT TYPE="checkbox" NAME="IUT">IUT<BR>
<INPUT TYPE="checkbox" NAME="AUTRE">AUTRE
<P>
Ecrivez ci-dessous le sujet de votre projet informatique :<BR>
<TEXTAREA NAME="Projet" ROWS=8 COLS=55>
Voici mon projet d'option informatique
</TEXTAREA>
<P>
<INPUT TYPE="RESET" VALUE="Effacer"> Pour recommencer<P>
<INPUT TYPE="SUBMIT" VALUE="Valider" >Pour envoyer ces informations<P>
</FORM >
```

Fiche de renseignements

Nom : Prénom : Mot de passe :

Vous êtes en terminale S à dominante : Mathématiques Physique-chimie SVT

Quelle est votre discipline préférée ?

Vous vous êtes inscrit(e)s en :
DEUG
CPGE
BTS
IUT
AUTRE

Ecrivez ci-dessous le sujet de votre projet informatique :

Pour recommencer Pour envoyer ces informations

TP1 de TP Formulaire

Champs de texte

Ligne de texte

```
<INPUT TYPE="text" NAME="Nom du champ" VALUE="texte initial" SIZE=longueur>
```

Les divers paramètres

- TYPE="text" est facultatif, car il s'agit du type de champ par défaut (si on ne précise pas).
- NAME pour préciser le nom du champ (utile pour la programmation).
- VALUE pour donner un texte visible au champ.
- SIZE fixe la longueur visible du champ (éventuellement son contenu défile vers la gauche).
- MAXLENGTH pour limiter le nombre de caractères pouvant être entrés dans le champ.

Exemples

```
Tapez votre nom : <INPUT NAME="Nom" VALUE="toto">
```

Tapez votre nom :

```
Tapez votre adresse e-mail: <INPUT NAME="e-mail" VALUE="...@.....fr" SIZE=40>
```

Tapez votre adresse e-mail:

```
Code Postal: <INPUT NAME="CP" SIZE=5 MAXLENGTH=5 VALUE="">
```

Code Postal: (taper exactement 5 chiffres)

Champ mot de passe

Quelquefois on ne veut pas que le texte tapé par l'utilisateur apparaisse à l'écran.
En particulier s'il s'agit d'un mot de passe ! Pour cela, on utilise le composant dont le type est **TYPE=PASSWORD**.

```
<INPUT TYPE="password" NAME="mot_de_passe" MAXLENGTH=taille maximum>
```

Exemple

```
Entrez votre mot de passe : <INPUT TYPE="password" MAXLENGTH=5>
```

Entrez votre mot de passe :

Zone de texte multiligne

Pour permettre à l'utilisateur de saisir un texte de plusieurs lignes, on utilise le composant <TEXTAREA

```
<TEXTAREA NAME=".." ROWS=.. COLS= ..>
Texte affiché par défaut ...<BR>
</TEXTAREA>
```

Les divers paramètres

- **ROWS** pour préciser le nombre de lignes de la fenêtre de saisie.
- **COLS** pour préciser le nombre de colonnes.
- **<TEXTAREA>texte </TEXTAREA>** le texte inséré est directement affiché tel quel; l'utilisateur peut l'effacer.

Exemple

```
<TEXTAREA NAME="Demande" ROWS=8 COLS=55>
Bonjour,
J'ai une demande très importante à formuler.
J'espère que vous pourrez me dépanner ...
</TEXTAREA>
```

Liste de sélection

Pour permettre un choix dans une liste de plusieurs options présentées sous forme de liste déroulante, on utilise la balise <SELECT>

```
<SELECT NAME="..."  SIZE=...>
<OPTION SELECTED >option 1
<OPTION>option 2
<OPTION>option 3
</SELECT>
```

Les divers paramètres

- **<OPTION>** pour introduire chaque option de la liste.
- **SIZE= ...** pour préciser le nombre de lignes visibles.
  S'il y a plus d'options, la liste pourra être parcourue à l'aide d'une barre de défilement.
- **<OPTION SELECTED>** pour sélectionner cette option par défaut .
- **<SELECT MULTIPLE>** l'attribut *facultatif* MULTIPLE autorise à sélectionner plusieurs options dans la liste.

Exemples

```
Vous êtes inscrit(e) au lycée J-Feyder dans une classe de :
<SELECT NAME="Niveau"  SIZE=3>
 <OPTION> seconde
 <OPTION> première
 <OPTION SELECTED> terminale
 <OPTION> section BTS
</SELECT>
```

Vous êtes inscrit(e) au lycée J-Feyder dans une classe de :

```
<SELECT NAME="pizza" MULTIPLE SIZE=3 MULTIPLE>
 <OPTION>  anchois
 <OPTION SELECTED>tomate
 <OPTION>olives
 <OPTION SELECTED>  fromage
 <OPTION>épices
 <OPTION>viande hachée
 <OPTION>câpres
</SELECT>
```

Que désirez vous dans votre pizza ?

Cases à cocher

Afin de répondre à des questions du type Oui/Non ou Vrai/faux, on utilise des boîtes à cocher avec la balise :

```
<INPUT TYPE="CHECKBOX" NAME= "..." CHECKED>question
```

Les divers paramètres

- **<TYPE ="checkbox" >** est indispensable pour produire une boite à cocher.
- **NAME=".."** s'il y a plusieurs boites, chacune doit avoir un nom spécifique.
- **CHECKED** pour cocher par défaut la boite (donc répondre Vrai par défaut).

Exemple

```
<INPUT NAME="Client" TYPE=CHECKBOX> Déjà client ?<BR>
<INPUT NAME="Brochure" TYPE=CHECKBOX CHECKED>Demande d'une brochure
```

Déjà client ?
Demande d'une brochure.

Boutons de sélection

Le type RADIO vous permet de définir des boutons radio.
Ils sont utilisés lorsque l'utilisateur doit faire un choix entre plusieurs options concurrentes, qui s'excluent mutuellement.
Ils jouent ainsi un rôle analogue aux listes de sélection.
Il faut donc indiquer le groupe des boutons dans lequel ce choix exclusif doit s'exercer, en leur donnant un même nom (la valeur de l'attribut NAME). L'utilisateur ne pourra alors sélectionner qu'un seul bouton radio à la fois dans chaque groupe.

```
<INPUT TYPE="RADIO" NAME="nom" VALUE=".." CHECKED>option 1<BR>
<INPUT TYPE="RADIO" NAME="nom" VALUE=".." >option 2<BR>
<INPUT TYPE="RADIO" NAME="nom" VALUE=".." >option 3
```

Les divers paramètres

- **<TYPE ="radio" >** est indispensable pour produire des bouton radio.
- **NAME="nom"** l'ensemble des boutons doit porter le MEME nom.
- **VALUE="..."** chaque bouton radio doit posséder une valeur spécifique pour différencier les options.
- **CHECKED** pour désigner LE bouton sélectionné par défaut.

Exemple

```
Vous habitez :<BR>
<INPUT TYPE=RADIO NAME="Ville" VALUE="E" CHECKED >Epinay<BR>
<INPUT TYPE=RADIO NAME="Ville" VALUE="SD"  >Saint-Denis<BR>
<INPUT TYPE=RADIO NAME="Ville" VALUE="V"  >Villetaneuse<BR>
<INPUT TYPE=RADIO NAME="Ville" VALUE="S"  >Stains<BR>
```

Vous habitez :
Epinay
Saint-Denis

Villetaneuse
Stains

Boutons de commande

Les boutons servent à recevoir des "clics". Leur principale fonction est de permettre à l'utilisateur de déclencher des événements auxquels seront rattachées des fonctions.

Boutons communs

```
<INPUT TYPE="Button" NAME= ".." VALUE="Cliquez">
```

La valeur de l'attribut VALUE est le texte gravé sur le bouton.
Un bouton sert essentiellement à déclencher des traitements locaux, par appel d'une fonction écrite en langage JavaScript.

Boutons de validation

Ce bouton sert à valider les informations saisies dans les champs du formulaire. Il provoque l'expédition de ces informations soit sur le serveur pour y être traitée, soit en message électronique ou bien

```
<INPUT TYPE="SUBMIT" NAME= ".." VALUE="Envoyer">
```

Boutons de réinitialisation

Ce bouton remet le formulaire dans son état initial, en réinitialisant les valeurs par défaut.

```
<INPUT TYPE="RESET" VALUE="Effacer>
```

TP 2 du TP formulaires

Compléments : transmission des données

La balise **FORM** possède plusieurs attributs, permettant de spécifier ce qui doit être fait lorsque l'utilisateur veut envoyer les données au serveur.

Attribut *action*

<FORM **ACTION="url"**...> permet d'indiquer l'URL (de protocole : *http, ftp, gopher, file, mailto, news, telnet, ...*) qui va recevoir les informations entrées dans le formulaire, lorsque l'on cliquera sur le bouton de validation.
Plus précisément, l'URL est l'adresse d'un programme (un script) qui va récupérer les données et les traiter. Si le champ ACTION est absent, l'URL sera celle du document courant.

Attribut *method*

L'attribut METHOD permet d'indiquer la méthode de transmission des données saisies dans le formulaire.
Il y en a deux:

- La méthode GET, méthode par défaut, consiste à concaténer les données du formulaire à l'URL spécifiée dans ACTION, après avoir inséré un point d'interrogation.
  Ces données sont incluses sous forme d'une liste *nom-champ=valeur-champ*.

On obtient alors une requête adressée au serveur, du genre :http://serveur/chemin/prog.html?champ1=valeur1&champ2=valeur2&....

- La méthode POST génére une requête HTTP spéciale qui envoie les données au serveur (plutôt qu'en l'accolant à l'URL). Il est recommandé d'utiliser la méthode POST

Exemple : *l'action MAILTO:*

C'est le moyen le plus simple pour faire expédier les informations du formulaire par l'utilisateur.
Il consiste à utiliser automatiquement le courrier électronique.
Il suffit de spécifier l'adresse électronique (e-mail) du destinataire après le nom du protocole MAILTO:
.
Si le serveur de messagerie est accessible, (si l'utilisateur est bien connecté), le corps du message acheminé contiendra la liste des couples champ1=valeur1

Par exemple, pour m'envoyer vos fiches de rensenseignements, il suffirait de remplacer, dans le formulaire du début de ce chapitre, la ligne <FORM NAME="Fiche_inscription"> par

```
<FORM  NAME="Fiche_inscription" ENCTYPE="text/plain" METHOD="post"
ACTION="MAILTO:jgourdin@ac-creteil.fr">
```

Pour expérimenter cette méthode, voir l'exercice 3 du TP formulaire

Programmation événementielle en JS

Résumé

L'utilisateur déclenche un "événement" (clic, déplacement souris, clic sur un bouton, choix d'un option de liste déroulante etc ...) 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 connaitre la correspondance objet-événement.
S'il prévoit un intérêt à "répondre" à cet événement, le programmeur doit à l'avance, associer du code JS ou une fonction JS à un tel couple objet-événement. L'appel et l'exécution de ce code ou de cette 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 paramétres, souvent un formulaire entier.

Evénements JS

Nous avons déjà vu des exemples d'appels de fonctions JavaScript provoquées par des événements qui surviennent **au moment de l'exécution du programme.**
Ces événements sont des actions qui sont déclenchées le plus souvent par l'utilisateur.
Par exemple, un clic sur un bouton (composant de formulaire) est un événement, comme l'est la validation d'un texte saisi dans une ligne de texte, ou le choix d'une option dans un composant case à cocher
Le navigateur reconnait 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** appelée systématiquement lorsque ces événements sont provoqués.

Un gestionnaire d'événement est une procédure particulière attachée à une balise HTML,

1. prédéfinie par le langage JS (par exemple onClick)

2. déclenchée par l'événement correspondant (click sur un composant de type button)
3. qui apparait dans la balise du composant qui reçoit l'événement
4. et à laquelle on affecte une fonction écrite en JS, déclarée préalablement

définition générale

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

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 code à écrire est : Le paramètre **this.form** fait référence au formulaire qui contient le bouton lui-même.

Remarques

On peut utiliser plusieurs gestionnaires d'événements sur un même composant.
Par exemple : <A HREF="URL" onMouseOver="fct1()" onMouseOut="fct2()">voir ...</A>

Liste des événements

- **onBlur** : se produit quand un champ Textarea, Text ou Select n'est plus activé.
- **onChange** : se produit quand un champ Textarea, Text ou Select est modifié par l'utilisateur.
- **onClick** : se produit quand un composant Button, Checkbox, Radio Link, Reset ou Submit reçoit un click souris
- **onFocus** : se produit quand un composant Textarea, Text ou Select est activé.
- **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
- **onMouseOver** : se produit quand la souris passe sur un Hyperlien ou une zone activable d'une image.
- **onSelect** se produit quand un composant Textarea ou Text est sélectionné.
- **onSubmit** : se produit quand un formulaire est globalement validé appui du bouton *Submit*.
- **onUnload** :se produit quand on quitte un document. L'événement *onUnload* se positionne dans la balise BODY ou dans la balise FRAMESET.
- **onAbort** : se produit quand l'utilisateur interrompt le chargement d'une image
- **onError** : se produit quand le chargement d'une page ou d'une image produit une erreur.
- **onMouseout** : se produit quand la souris quitte une zone Area d'une image ou un hyperlien.
- **onReset** : se produit quand on clique sur le bouton *reset* d'un formulaire

Tableau récapitulatif

Gest. événement	provoqué par l'utilisateur qui ...	sur les objets ...
onBlur	enlève le focus du composant	text, textarea, select
onChange	change la valeur d'un texte ou d'un composant à options	text, textarea, select

onClick	clique sur un composant ou un hyperlien	button, checkbox, radio, reset, submit
onFocus	donne le focus au composant	text, textarea, select
onLoad	charge la page dans le navigateur	balises BODY, FRAMESET
onMouseOut	la souris quitte un lien ou une ancre	balises <A HREF..>, <AREA HREF..>
onMouseOver	bouge la souris sur un lien ou une ancre	balises <A HREF..>, <AREA HREF..>
onReset	efface les saisies d'un formulaire	bouton reset
onSelect	sélectionne une zone d'édition d'un formulaire	text, textarea
onSubmit	soumet un formulaire	bouton submit
onUnload	quitte la page	balises BODY, FRAMESET

Programmer un formulaire

- ***Prérequis: Il est nécessaire de connaitre déjà les champs (ou composants) de formulaires du langage HTML.***
  Pour les découvrir, voir ce chapitre d'introduction aux formulaires HTML
- Un formulaire ***form*** se présente comme un objet inclus dans un objet ***document*** (page HTML).
  On sait que lui-même se conduit comme un *conteneur* d'autres objets que sont les composants usuels d'une interface graphique.
  Ces objets possèdent des propriétés et des méthodes (fonctions) qui correspondent pratiquement aux attributs des balises HTML qui les construisent .
  Pour l'essentiel, la tâche du programmeur est d'analyser les traitements à effectuer sur les informations extraites du formulaire puis d'écrire les fonctions correspondantes, appelées par le déclenchement d'événements par l'utilisateur.

Champ de TEXTE

```
<FORM NAME="Questions">
<INPUT TYPE="text" NAME="classe" VALUE=""  SIZE=10> ;
</FORM>
```

- Le texte entré par l'utilisateur est affecté à la propriété **value** du composant **classe** situé dans le formulaire **Questions** placé dans le document courant appelé **document** (par défaut).
- De l'intérieur de ce document, le contenu du champ de texte est donc "visible" et accessible par son nom complet :
  document.Questions.classe.value
- En particulier, il peut être récupéré et affecté à une variable **MaClasse** par l'expression:
  MaClasse = document.Questions.classe.value
  La référence **F.classe.value** est suffisante car la portée du formulaire F est le document où il est inséré.
- Les formulaires dans un même document sont stockés dans un tableau appelé **forms**; le premier formulaire est alors noté *forms[0]*, la deuxième est *forms[1]* etc..
  Si **Questions** est situé en position de premier formulaire, on peut remplacer l'affectation précédente par :
  MaClasse = document.forms[0].classe.value

Liste de sélection SELECT

```
<FORM NAME="F">
<SELECT NAME="liste"
        onChange='F.resultat.value+=F.liste.options[F.liste.selectedIndex].value' > ;
```

```
<OPTION VALUE="matin ... " >Matin
<OPTION VALUE="midi ... ">Midi
<OPTION VALUE="soir ... ">Soir
</SELECT>
<INPUT TYPE=TEXTE SIZE=40 NAME="resultat" VALUE="Vous avez sélectionné : " >
</FORM>
```

Propriétés de l'objet SELECT

- **selectedIndex** est une propriété dont la valeur est le numéro de l'élément sélectionné dans la liste
  var num = F.liste.**selectedIndex** ---> *num* est le numéro de l'élément sélectionné dans le composant liste situé dans le formulaire F.
- **options[]**est le tableau prédéfini contenant les *objets* de la liste
- **F.liste.options[num]** est l'objet champ situé au N° num (rappel : le 1er a le numéro 0)
- **F.liste.options[num].value** est la valeur de l'option N° num de la liste.
- **F.liste.options[num].text** est le texte suivant le champ <OPTION>.

Boutons de sélection RADIO

```
<FORM NAME="F">
<INPUT TYPE="RADIO" NAME="choix" value="sup. à 10" CHECKED
        onClick='F.resultat.value=this.value'>plus de 10
<INPUT TYPE="RADIO" NAME="choix" value="entre 8 et 10"
        onClick='F.resultat.value=this.value'>entre 8 et 10
<INPUT TYPE="RADIO" NAME="choix" value="inf. à 8"
        onClick='F.resultat.value=this.value'>moins de 8

<INPUT TYPE=TEXTE SIZE=15 NAME="resultat" VALUE="" >
</FORM>
```

plus de 10
entre 8 et 10
moins de 8

Propriétés de l'objet RADIO

- **F.choix[num]** est le bouton radio N° num (le 1er a le numéro 0) de l'ensemble de boutons nommé **choix**
- **checked** est le booléen décrivant l'état d'un bouton : if (F.choix[num].**checked**) ....
- **F.choix[num].value** est la valeur associée au bouton N° num de la série de bouton.

Cases à cocher CHECKBOX

Le traitement est complètement semblable au cas des boutons radio, à la différence que chaque case possède un nom propre distinct des autres. L'état **checked** de chaque case doit donc être testé individuellement.
Voir l'exemple complet qui suit.

Bouton de validation SUBMIT

Lorsque l'utilisateur "soumet" ses réponses en cliquant sur le bouton **submit**, une fonction programmée en réaction au gestionnaire de l'événement **onSubmit** peut analyser ses choix et y répondre.
On peut plus simplement préférer utiliser un bouton normal qui lance une fonction en réaction à l'événement **onClick()**
Voir l'exemple suivant.

Exemple récapitulatif

Voici la reprise du questionnaire présenté en exemple dans le chapitre formulaire du cours HTML.

Examiner en détail cette annexe. Exécuter plusieurs fois et parallèlement étudier le code JS. Comprendre comment on peut effectuer des traitements à partir des choix de l'utilisateur.

Traitement d'un questionnaire (exemple-evenement.html)

Envoyer un formulaire

Pour envoyer la saisie d'un formulaire sans faire appel à des programmes à placer sur le serveur WEB, on peut envoyer le formulaire par courrier électronique de façon simple pour l'utilisateur.
Il suffit d'ajouter un attribut particulier **ACTION=** dans la balise **<FORM >** qui précise le mode de transmission
Voici comment envoyer le contenu du formulaire par e-mail,

```
<FORM ACTION="mailto:nom destinataire@nom serveur" METHOD="POST">
.....
<INPUT TYPE="SUBMIT" VALUE="Envoyer"
<INPUT TYPE="RESET" VALUE="Effacer"
</FORM>
```

Etude des événements *onMouse* sur une image en coordonnées

Ici, on détourne les hyperliens correspondant aux zones de l'image cliquable de façon à ce qu'un clic dans une zone appelle une fonction.
Pour cela, on n'affecte pas d'URL à l'attribut HREF de la balise <AREA>.
A la place, on affecte un appel d'une fonction (déjà déclarée) aux gestionnaires d'événements **onMouseOver** ou **onMouseOut** pour réagir au passage de la souris sur chacune des zones.

```
<MAP NAME="mon_image">
<AREA  HREF="" SHAPE="poly"  COORDS="x,y,..." onMouseOver="fct1()" >
<AREA  HREF="" SHAPE="circle"  COORDS="x,y,..."onMouseOver="fct2()" >
.....
</MAP>
.....
<IMG  SRC="fichier image.gif" USEMAP="#mon_image">
```

Dans l'exemple ci-dessous le passage de la souris sur la zone France de la carte, génére un événement onMouseOver qui provoque l'appel de la fonction affiche(France), tandis que le fait de quitter cette même zone associé à l'événement onMouseOut provoque l'appel de affiche(efface).

```
<AREA COORDS="23,213,48,206,79,202,95,220,115,230,96,254,
  102,260,99,281,104,285,88,293,77,284,65,293,30,272,42,245,23,213"
  SHAPE="poly"  HREF="" onMouseOver="affiche(France)" onMouseOut="affiche(efface)">
```

Exemple (**référence** : JavaScript, Michel Dreyfus, édition Sybex)

Les objets du navigateur

La hiérarchie des objets JavaScript

Voici la hiérarchie de toutes les classes d'objets gérés par le Navigateur
navigator

```
window
  |
  + --parent, frames[], self, top
  |
  + --location
  |
  + --history
  |
  + --document
      |
      + --forms[]
      |     |
      |  elements[] (text, textarea, checkbox, password
      |             radio, select, button, submit, reset)
      + --links[]
      |
      + --images[]
      |
      + --URL
```

Les classes de base du navigateur

- Les objets de ces classes 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 JavaScript 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.
- Remarque
  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 ...
- Objets fondamentaux
  - **navigator** : c'est le logiciel client dont le nom est noté dans navigator.appName
  - **window** : l'objet de plus haut niveau créé par le navigateur, c'est sa fenêtre.
  - **location** : c'est l'objet URL de la fenêtre courante.
  - **history** : ce sont les URL précédemment visitées.
  - **document** : il s'agit du document courant, dont les propriétés sontle titre, les couleurs (fond, texte, lien ...), les formulaires, les images etc..

Accéder aux propriétés d'un objet

- **Une propriété est une donnée qui décrit un objet.**
  Pour désigner complètement et correctement un objet, il faut faire précéder son nom de la suite des noms des objets qui le contiennent (en respectant la hiérarchie des objets); cette liste doit être séparée par l'opérateur . (point).

- Comme la couleur de fond est la propriété *bgColor* de l'objet document, pour mettre le fond en bleu on écrira
  document.bgColor = 'blue' ;
- **Pour la page HTML courante**, nous avons les propriétés suivantes :
  Propriétés de l'objet **WINDOW**
  - Protocole utilisé : **window.location.protocol** = *http:*
  - URL : **window.location.href** = *http://www.ac-creteil.fr/util/programmation/javascript/Jour3/c-objets-navigateur.html*
  - Message de la barre d'état : **window.defaultStatus** =

  Propriétés de l'objet **DOCUMENT**

  - Titre du document : **document.title** = *Cours JavaScript*
  - Couleur du texte : **document.fgColor** = *#000000*
  - Couleur du fond : **document.bgColor** = *#f1f1f1*

Compléments : connaitre les propriétés d'un objet

On peut connaitre toutes les propriétés des objets avec une itération FOR .... IN
Par exemple pour l'objet document :

```
function proprietes() {
var objet;
var n=document.ppte.liste.selectedIndex;
switch (n) {
case 0:  objet=navigator; break;
case 1:  objet=navigator.plugins; break;
case 2:  objet=navigator.mimeTypes; break;
case 3:  objet=window; break;
case 4:  objet=window.location; break;
case 5:  objet=window.history; break;
case 6:  objet=window.document; break;
case 7:  objet=document.forms[0]; break;
case 8:  objet=document.forms[0].elements; break;
case 9:  objet=document.images; break;
case 10:  objet=document.links; break;
case 11:  objet=forms[0].liste.options; break;
}
var nom=document.ppte.liste.options[n].text;
fen=open("","Proprietes","width=600,height=250,toolbar=yes,
 directories=no, menubar=no,scrollbars=yes,status=yes");
fen.focus();
var texte = "";
 for (var i in objet )
 texte +=nom+"."+i+" = " +objet[i] +"
";
fen.document.write(texte);
fen.document.close();
}
```

La classe navigator

Cette classe ne contient qu'un seul objet, appelé **navigator**, qui est créé au démarrage du logiciel.
Voici l'ensemble de ses propriétés, avec leur valeur, pour le navigateur actuel.
navigator.userAgent = Mozilla/4.7 [fr] (Win95; I)
informations générales envoyées au serveur HTTP à chaque requête du navigateur
navigator.appCodeName = Mozilla
nom de code
navigator.appVersion = 4.7 [fr] (Win95; I)
informations sur la plate-forme d'exécution
navigator.appName = Netscape
navigator.language = fr
navigator.platform = Win32
type de machine
navigator.securityPolicy = France policy
navigator.plugins = [object PluginArray]
tableau des plug-ins installés
navigator.mimeTypes = [object MimeTypeArray]
tableau des types mimes (voir Edition/préférences/navigator/applications

Exemples d'utilisation

1. Il est nécessaire de tester le navigateur "client", pour pouvoir adapter le code si nécessaire aux 2 navigateurs Netscape et Explorer.
   Par exemple, voici un script qui renverrait à la page précédente s'il ne décèle pas Netscape.
2. if (navigator.appName !='Netscape')
3. window.history.back();
4. else
5. document.write('Vous avez fait le bon choix !')
6. Tout aussi important est la nécessité de savoir si le navigateur posséde un plug-in, module externe, pour interpréter certains types de fichiers comme les fichiers sons, vidéos, pdf ..
   Exemple :
   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 : posséde t-il un élément indexé par 'Shockwave' ?
7. if (navigator.plugins['Shockwave'])
8. document.write('<EMBED NAME="acte_1" SRC="acte_1.swf"
9. WIDTH="100%" HEIGHT="100%" ALIGN="LEFT" QUALITY="high" SALIGN="lT"> ');
10. else
11. document.write('Désolé, votre navigateur ne sait pas afficher *Shockwave*');

La classe WINDOW de JavaScript

Présentation

Il s'agit de la classe située au sommet de la hiérarchie.
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()***, il faut préfixer par son nom.

Voici les principales propriétés et méthodes

Propriétés simples

- **defaultStatus** : représente le message de défaut qui sera affiché dans la barre de statut <body onLoad="defaultStatus='Bonjour à tous'">
- **status** est un message affiché dans la barre de statut de la fenêtre. window.status="N\'oubliez pas de fermer vos fenêtres !"
- **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 éventuellement 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** booléen qui indique si la fenêtre a été fermée. utilisation : if (! fen.closed) fen.close();

Propriétés objets

Ces propriétés sont en fait elle-même des objets dotées de propriétés et de méthodes

- **document** c'est le document HTML étudié
- **history** liste des documents chargés dans la fenêtre (simule l'action des boutons suivant et précédent)
  - history.back() : document précédent
  - history.forward() : le suivant
  - history.go(n) : recharge le document situé à n étapes du présent document (n est de signe qcq)
- **location**
  - location.href : chaine contenant l'url
  - location.hash : partie de l'url après # (lien interne)
  - location.host : nom serveur et port
  - location.protocol : http, ftp, javascript, ..
  - location.reload() : recharge le document courant = history.go(0)
  - location.replace(url)
- **frames[]** est un tableau représentant tous les cadres dans une fenêtre Les fenêtres-cadres peuvent être nommées au moment de leur définition par un nom, sinon on peut les référencer par frames[index]

Méthodes de gestion d'objets fenêtres

Avec *open()*et *close()*, le programmeur dispose de moyens très souples avec les gestionnaires d'événements, d'ouvrir et de fermer des fenêtres auxiliaires.

nomFenetre =open("URL","nom_fenetre", "options"); ouvre une nouvelle fenêtre

- "URL" est l'adresse du document à charger dans la nouvelle fenêtre.
- "nom_fenetre" est le nom de la fenêtre, à donner à l'attribut TARGET des balises <FORM> ou <A>
- "options" est une liste d'éléments optionnels qui précisent l'aspect de la fenêtre.

- On utilise ensuite **nomFenetre** pour faire référence à un objet ou une propriété de la nouvelle fenêtre.
  Par exemple, pour écrire dans le composant message du formulaire formu situé dans le document de la fenêtre fen, créee par fen=open(...), on écrit :
  fen.document.formu.message.value= "...";

Paramètres 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 séparées par des virgules

barre d'outils	toolbar[=yes|no] | [=1|0]
url visible	location[=yes|no] | [=1|0]
répertoire ?	directories[=yes|no] | [=1|0]
barre d'état	status[=yes|no] | [=1|0]
barre de menu	menubar[=yes|no] | [=1|0]
barre de défilement	scrollbars[=yes|no] | [=1|0]
redimensionnement	resizable[=yes|no] | [=1|0]
visibilité permanente ?	alwaysRaised[=yes|no] | [=1|0]
fermée avec la fen. parente	dependant[=yes|no] | [=1|0]
hauteur	height= dimension en pixels
largeur	width= dimension en pixels
positionnement X	screenX = nb pixels
positionnement Y	screenY = nb pixels

Autres méthodes de window

- **alert(), prompt() et confirm()**, ont déjà été utilisées
- nb= prompt('Donner un nombre : ', 7)
  if (isNaN(nb)) alert(nb+" n\'est pas un entier !");

- méthodes focus(), blur() qui active ou désactive la fenêtre
- moveBy(dx, dy)) déplace la fenêtre par translation et moveTo(x,y) déplace le coin gauche, haut au point (x, y).
- resizeBy(dl, dh) et resizeTo(l, h) jouent des rôles analogues, quant à la taille de la fenêtre.
- les méthodes de temporisation setTimeout() et clearTimeout()

Exemples de base

Apprenons à utiliser les méthodes **open()**, et **close()**

Exemple 1 : ouvrir

La fonction **OuvrirFenetre()**, déclenchée par l'événement *onClick* sur le bouton, utilise la fonction **open()** pour ouvrir une nouvelle fenêtre, nommée **fen1**

```
<SCRIPT>
var hauteur=200;   // hauteur de la fenêtre à créer
var largeur=500;
options="width="+largeur+",height="+hauteur+"toolbar=yes,
directories=no, menubar=no,scrollbars=yes,status=yes";

function OuvrirFenetre1() {
fen1 =open("","Nouvelle_fenetre1",options);
}
</SCRIPT>
<form>
Pour créer une fenêtre "enfant", cliquer sur ce bouton :
<INPUT type =Button value="Nouvelle fenêtre 1"
 onClick="OuvrirFenetre1()">
```

```
</form>
```

Exemple 2 : *"Une fenêtre doit être ouverte ou fermée"*

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>
var hauteur=200;   // hauteur de la fenêtre à créer
var largeur=500;
options="width="+largeur+",height="+hauteur+"toolbar=yes,
directories=no, menubar=no,scrollbars=yes,status=yes";

function OuvrirFenetre1() {
fen1 =open("","Nouvelle_fenetre",options);
}
</SCRIPT>
<body><form>
Pour créer une fenêtre "enfant", cliquer sur ce bouton :
<INPUT type =Button value="Nouvelle fenêtre 1"
            onClick="OuvrirFenetre1()">
<INPUT type =Button value="Fermer"  onClick="fen1.close()">
</form></body>
```

Pour vérifier :

Exemple 3 : afficher un document dans une fenêtre

Chargement d'un document à la création d'une nouvelle fenêtre
Cet exemple est semblable au précédent, à la différence que le fichier *tp-window.html*, passée en 1er paramètre de OPEN(), va être affichée dans le document dela nouvelle fenêtre.

```
function OuvrirFenetre2() {
fen2 =open("tp-window.html","Nouvelle_fenetre",options);
// les 2 lignes suivantes sont équivalentes
fen2 =open("","Nouvelle_fenetre",options);
fen2.location="tp-window.html";
}
```

Pour vérifier :
Ecriture dans la nouvelle fenêtre après sa création
Supposons avoir créé une fenêtre vide, nommée Fen3.
Pour écrire dans cette fenêtre, il suffit de passer une chaine de caractères à la méthode document.write de Fen3 : *Fen3.document.write("texte")*.
Attention ! message ne sera effectivement affiché dans Fen3, que s'il est impérativement suivi de code comme
 ou si le document (pas la fenêtre) est fermé Fen3.document.close();

```
function  EcrireFenetre3(message) {
fen3 =open("","Nouvelle_fenetre",options);
if (message !="")
  fen3.document.write(message+"</BR>");
  else
  alert ("message vide !");
}
<FORM>
Ecrire un court texte dans le champ de saisie ci-dessous
<TEXTAREA NAME="zone" ROWS=8 COLS=55 VALUE="" WRAP="PHYSICAL">
Je suis très fier de ma nouvelle fenêtre !
</TEXTAREA>
puis le faire afficher dans <TT>MaFenêtre </TT>
```

```
en cliquant sur :
<INPUT type =Button name="Ecrire" value="Ecrire"
 onClick="EcrireFenetre3(this.form.zone.value)">
</FORM>
```

Exemple 4 : charger un document en ligne

```
<script>
function fermer() {
if (fen1 !=null) fen1.close();
if (fen2 !=null) fen2.close();
}
function Connexion(moteur) {
if (moteur==1) {
            fen1=open("","YAHOO",opts);
            fen1.focus();
            fen1.location.href='http://www.yahoo.fr';
            }
            else  {
            fen2=open("","NOMADE",opts);
            fen2.focus();
            fen2.location.href='http://www.nomade.fr';
            }
}
<form>
<input type=button value="Connexion à Yahoo" onClick="Connexion(1)">
<input type=button value="Connexion à Nomade" onClick="Connexion(2)">
<INPUT type =Button value="Fermer"  onClick="fermer()">
</form>
```

Méthodes de temporisation setTimeout() et clearTimeout()

- La méthode (de l'objet Window) **setTimeout()** déclenche un miniteur, un compte à rebours au bout duquel un expresion JS passée en 1er paramètre sera exécutée; le temps d'attente est passé eb 2ème paramètre.
- Syntaxe : **TimeoutID=setTimeout(expression, ms)**
    - expression : c'est une chaîne de caractères contenant la formule de calcul ou 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 le minuteur avant sa fin. Surtout utilisé pour arrêter des appels récursifs d'une fonction passée dans setTimeout()
- Exemple simple
- <HTML><HEAD>
- <script>
- var w, id
- function ouvrir() {
- w=open("","Essai","width=50,height=100,resizable=1,scrollbars=1")
- }
- function essai() {
- w.focus();
- w.document.write("bonjour
")
- id= setTimeout ("essai()",1000)
- }
- function arreter() {

```
 clearTimeout(id);
 w.focus();
 }
</script></HEAD>
<BODY>
<form>
<INPUT type =Button value="Démarrer" onClick="ouvrir();essai()">
<INPUT type =Button value="Arrêter" onClick="if (w != null) arreter()">
<INPUT type =Button value="Fermer" onClick="if (w != null) w.close()">
</form>
```

Connaitre les propriétés de window

```
function  propriétés(objet, nom) {
var texte = "";
 for (var i in objet )
 texte +=nom."i"+i+" " +objet[i] +"<br>;
return texte
}
document.write("<HR>Propriétés <BR>"+
 propriétés(document,"document")+"<HR>")
```

La classe document

La hiérarchie de la classe document

- Un document HTML peut contenir diverses balises d'insertion d'images, de formulaires, d'hyperliens, etc .. qui sont autant d'objet du point de vue du navigateur.
- JavaScript les classe dans des tableaux d'objets au fur et à mesure, dans l'ordre de leur apparition dans le document. Chaque objet peut posséder des propriétés et des méthodes.
- Par exemple, soit le 1er formulaire déclarée par <FORM NAME="formu">
  On pourra le nommer aussi bien par document.formu que par document.forms[0]
- De même images[2] est la 3ème image insérée, grâce à une balise <IMG SRC=..>
- Autre exemple :
  Soit un champ de texte nommé lycee. Il a été défini dans le formulaire nommé formu par
  <INPUT type="text" name="lycee" value="" size=15>
  Pour changer le texte on écrira : document.formu.lycee.value="J-Feyder";

Le programmeur JS peut avoir accès en lecture à ces tableaux.
Leur taille est obtenu avec la propriété length; ainsi forms.length est le nombre de formulaires inclus dans le document.
En voici la liste, et les balises correspondantes :

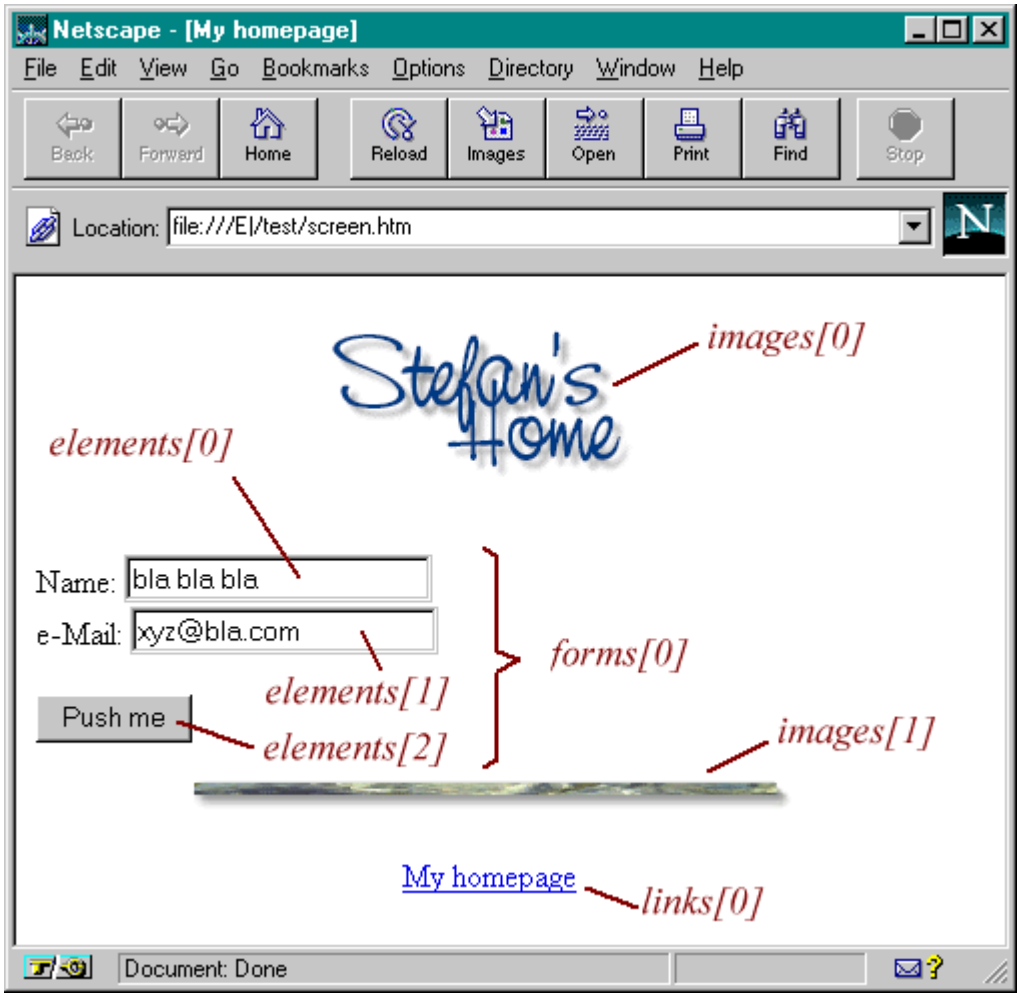
- anchors[] tableau des liens internes <A NAME= ...>
- applets[] tableau des applets <APPLET CODE=...>
- embeds[] tableau des objets insérés <EMBED SRC=..>
- forms[] tableau des formulaires <FORM >
  - elements[] tableau des composants du formulaire <INPUT ..>, <SELECT ..>, ..
- images[] tableau des images <IMG SRC=...>
- links[] tableau des liens hypertextes <A HREF= ...>

Un exemple concret

Soit le code d'une page d'accueil :
(Source *Stefan Koch* http://rummelplatz.uni-mannheim.de/~skoch/js/tutorial.htm)

```
<html>
<head>
<title>My homepage</title>
</head>
<body bgcolor=#ffffff>
<center>
<img src="home.gif" name="image1" width=200 height=100>
</center>
<p>
<form name="formu">
Name:  <input type="text" name="nom" value=""><br>
e-Mail: <input type="text" name="email" value=""><p>
<input type="button" value="Push me" name="bouton" onClick="alert('Salut')">
</form>
<p>
<center>
<img src="ruler.gif" name="image4" width=300 height=15>
<p>
<a href="http://rummelplatz.uni-mannheim.de/~skoch/">My homepage</a>
</center>
</body>
</html>
```

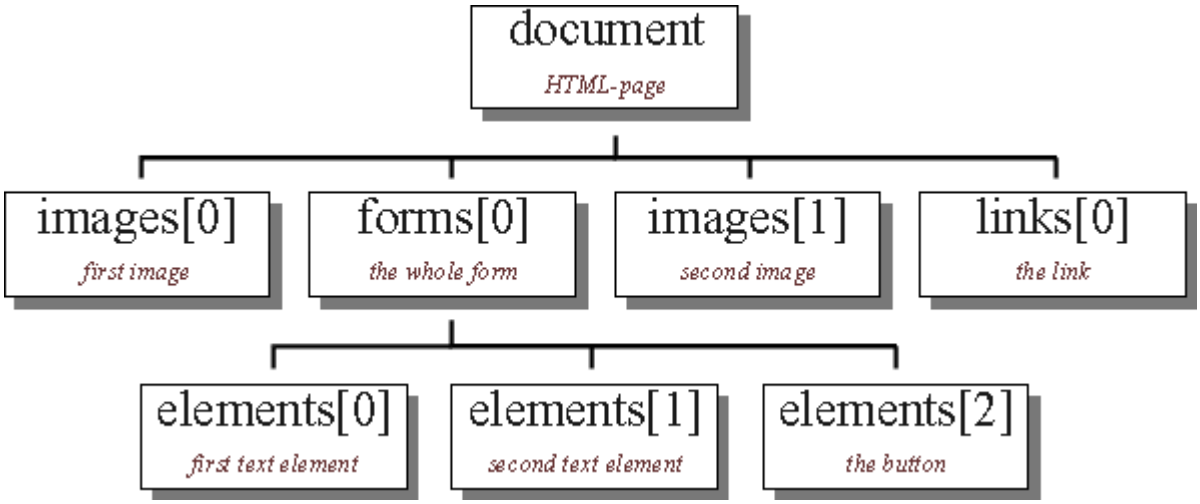
Voici son aspect



Dans cet objet-window, cet objet-document posséde en tout 2 images, un lien et un formulaire contenant 2 champs de texte et un bouton

Le contenu saisi dans le 2ème champ de texte peut être obtenu par :
var mail = document.forms[0].elements[1].value ou
var mail = document.formu.nom.value;

Voici la hiérarchie des objets construits par JS sur ce document



Principales propriétés

Voici leur valeur pour le présent document :

- Titre du document : **document.title** = *Cours JavaScript*
- Couleur du texte : **document.fgColor** = *#000000*
- Couleur du fond : **document.bgColor** = *#f1f1f1*
- Couleur de liens : **document.linkColor** = *#0000a0*
- Couleur de liens visités : **document.vlinkColor** = *#cc3366*
- Couleur de liens activés : **document.alinkColor** = *#cc3366*
- Adresse du document : **document.URL** = *http://www.ac-creteil.fr/util/programmation/javascript/Jour3/c-document.html*
- Date de dernière modification : **document.lastModified** = *12/26/2001 16:26:10*
- Le cookie du document : **document.cookie** =

Principales méthodes

- write(ch1, ch2, ..) affiche les chaines ou des arguments traduits en chaines.
- open() crée un nouveau document, ne pas confondre avec la création d'une fenêtre !
- close() ferme le document, impossible d'y ajouter alors du texte.
- getSelection() donne le texte sélectionné

Traitements des images en JS

Rappels HTML

- la balise <IMG NAME= .. SRC="URL" WIDTH=.. HEIGHT=.. ALIGN= ..>
- URL est l'adresse internet (en général relative) du fichier image à charger (.gif ou .jpg)
- l'alignement de l'image par rapport au texte voisin est réglé par l'attribut align
- une image peut servir d'ancre pour un hyperlien
- le navigateur classe les images insérées dans le document dans un tableau **images[]**

L'objet Image

Il fait partie des objets (ou plutôt des classes d'objets) prédéfinies du langage (à partir de la version 1.2). Contrairement à l'usage de la balise IMG, on peut précharger l'image et la stocker en cache (mémoire+disque) dans un objet de type Image, avant de l'afficher.

Contructeur Image()

Il s'agit d'une fonction spéciale sans paramètre qui sert à créer un objet.
var monImage = new Image()

Propriétés

- Elles correspondent aux attributs de la balise <IMG>
  name, src, lowsrc, height, width, border, hspace, vspace ..
- La propriété la plus importante, **src** permet de charger le fichier image à partir d'une URL et de le stocker dans l'objet.
- var monImage = new Image();
- monImage**.src** = "moi.jpg";
- Mais l'image n'est pas affichée. Elle pourra l'être dans une balise <IMG>
  <IMG SRC=...>
- De plus, malheureusement, les propriétés height et width ne sont pas modifiables

Méthodes

Exemples de manipulations d'images

Evénements onMouse

Soit une hyper-image nommée **img** (une image qui sert d'ancre à un lien interne ou externe). Lorsque l'utilisateur promène sa souris sur une telle image, il provoque un événement onMouseOver qui doit changer la source de l'image.
Lorsqu'il la quitte, l'événement onMouseOut est provoqué et l'image permanente sera de nouveau affichée

Indication : affecter directement les gestionnaires d'événements par l'instruction **img.src='fichier-image'**.

Solution proposée

```
<BODY>
<A HREF="javascript:void(0)"
 onMouseOver="img.src='../images/connect.gif' "
 onMouseOut="img.src='../images/image.gif' ">
  <IMG NAME="img" WIDTH=60 HEIGHT=61 BORDER=0
  SRC="../images/image.gif"></A>
 ..........
</BODY>
```

Version améliorée

Voici une solution plus générale et satisfaisante, qui consiste à charger d'abord les images dans un tableau d'images, dès l'affichage du document.

```
<SCRIPT>
var commentaire=new Array(2);
var images=new Array(2);

function init() {
 commentaire[0]="Et voici mon chat jeune .."
```

```
commentaire[1]="C'est moi en pleine action .."
for (var i=0 ; i <images.length; i++)
 images[i]= new Image();
 images[0].src="../../images/monchat.gif"
 images[1].src="../../images/etmoi.jpg"
}
function changimage(n) {
document.image.src=images[n].src;
status=commentaire[n];
}
</SCRIPT>

<BODY onLoad="init()">
<H2>Evénements onMouseover et onMouseout sur un lien-image</H2>

<A HREF="javascript:void(0)"
 // return true est obligatoire, pourquoi ?
 onMouseOver="changimage(1); return true";
 onMouseOut="changimage(0)">
<IMG NAME="image" width=640 height=480 SRC="../../images/monchat.gif">
</A>
```

Cliquer pour changer d'image

Solution proposée

```
<script langage="JavaScript1.2">
var tab_image = new Array(5);
for (var i=0; i< tab_image.length;i++)
            tab_image[i]= new Image();
tab_image[0].src='../images/dalhias.jpg';
tab_image[1].src='../images/eglantines.jpg';
tab_image[2].src='../images/roses rouges.jpg';
tab_image[3].src='../images/roses jaunes.jpg';
tab_image[4].src='../images/chrysanthemes.jpg';

var n=0;

function change() {
n= ++n;
if (n == tab_image.length) n=0;
window.document.image.src=tab_image[n].src;
}
</script>
</HEAD>

<BODY>
<H4>Evénements onClick sur un lien-
image</H4>
<A HREF="JavaScript:change()">
<IMG NAME="image" WIDTH=200 HEIGHT=150
SRC="../images/fleurs1.jpg">
Cliquer pour changer d'image
</A>
```

Evénements onClick sur un lien-image

Cliquer pour changer d'image

Défilement contrôlé d'images

Description

Ici le changement d'image n'est pas lié au déclenchement d'un événement provoqué par l'utilisateur (onMouseOver ci-dessus), mais est automatiquement démarré au chargement de l'image initiale par la balise IMG, grâce à l'événement onLoad qui appelle la fonction anime(). Celle-ci se charge de changer la valeur de la propriété src donc de provoquer aussitôt l'affichage d'une autre image.
Plus précisément :

- création d'une variable-objet de type image : *var monImage = new Image();*
- le chargement d'une image s'effectue dans le cache, sans affichage immédiat (comme c'est le cas avec la balise IMG) : monImage.src = "../images/Monchat.jpg"
- L'URL de l'image est spécifiée comme valeur de la propriété src

Exemple de code

```
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
d = 100 ;
n = 0 ;
var im = new Array(10) ;

function init(){
 for (var i=0; i<im.length;i++){
  im[i] = new Image();
  im[i].src= "../images/"+i+".gif" ;
  }
}
function anime() {
 document.monImage.src= im[n].src ;
 n++;
 if (n == im.length) n=0;
}
</SCRIPT>
</HEAD>

<BODY onLoad="init()">

<IMG SRC="i0.gif" NAME="monImage"
          BORDER=0 WIDTH=58 HEIGHT=76
          onLoad="setTimeout('anime()',d)">
</BODY>
```

Exemple de réalisation

Programmation multi-cadres

Rappels HTML

- La balise <FRAMESET> est une directive de partage de la fenêtre initiale du navigateur en sous-fenêtres
- Les documents Elle remplace la balise <BODY>
- Chaque balise décrivant chaque "frame", cadre est <FRAME>
- Chaque document est décrit par

Un exemple

Divisons la fenêtre en deux colonnes principales d'égales largeurs. Ensuite, la colonne de droite est elle-même divisée en 3 rangées d'égales hauteurs.

code	représentation

```
...
<FRAMESET COLS="50%,50%">
  <FRAME SRC="fichier1.htm">
  <FRAMESET ROWS="33%,33%,33%">
     <FRAME SRC="fichier2.htm">
     <FRAME SRC="fichier3.htm">
     <FRAME SRC="fichier4.htm">
  </FRAMESET>
</FRAMESET>
```

fichier1.htm	fichier2.htm
	fichier3.htm
	fichier4.htm

Bien sûr, dans les 4 cadres, ce sont les contenus des 4 documents fichier1.htm , fichier2.htm...etc.. qui seront affichés.

Les principales balises

1. **<FRAMESET > </FRAMESET>**

   <FRAMESET ROWS=description-partage-rangées COLS= description-partage-colonnes
   Cette balise commande la division de la fenêtre du navigateur en plusieurs fenêtres juxtaposées et indépendantes.

   - Les paramètres ROWS et COLS permettent de partager une zone en plusieurs fenêtres disposées (respectivement) horizontales **OU** verticales.
   - Ils sont suivis d'une liste de valeurs, séparées par des virgules, qui détermine le fractionnement de la fenêtre, soit en hauteur (pour ROWS), soit en largeur (pour COLS). Le plus souvent, ces valeurs sont exprimées pourcentage de l'espace disponible.
   - Quelques exemples
     ROWS= "30%, 70% " signifie un partage en 2 rangées dont les hauteurs sont les 30% et 70% de la hauteur disponible dans la fenêtre.
     *ROWS= "30%, 50%, * "* signifie un partage en 3 rangées de hauteur (de haut en bas) 30%, 50% et le reste, indiqué par * (c'est-à-dire ici 20 %).
     *COLS = "150, * , * "* signifie un partage de la largeur disponible en 3 colonnes : celle de gauche mesurera 150 pixels exactement en largeur, et les 2 autres auront la même largeur en se partageant le reste de l'espace disponible.
     *COLS = " *, * , *, * "* provoque un partage de la largeur disponible en 4 colonnes de même largeur.
     *COLS = "10%, * , 2 * "* signifie un partage en 3 colonnes de largeurs 10%, 30% et 60 %.

2. **<FRAME .. >**

   <FRAME SCR = "fichier.htm " NAME=nom-cadre ... > </ FRAME>

   - *SRC* définit le contenu du cadre. Il faut y indiquer le nom du fichier HTML à charger dans le cadre, éventuellement sous forme d'URL pour un fichier non local
   - *NAME* = "nom du cadre", indispensable pour JS
   - *SCROLLING = "yes" | "no" | "auto"* impose ou non la présence de barre de défilement.
     Si on choisit SCROLLING = "auto" , les barres de défilement apparaîtront en cas de besoin.
   - *NORESIZE* interdit la modification de la taille par l'utilisateur (par défaut, tous les cadres sont modifiables.

L'attribut target dans les hyperliens

- En l'absence d'indication complémentaire, une marque habituelle d'hyperlien <A HREF="URL" > .... </A>, placée dans un cadre commandera l'affichage de ce document à l'intérieur du cadre même.
- Si l'hyperlien doit provoquer l'affichage dans un autre cadre, il faut préciser le nom du cadre de destination, indiqué comme valeur du paramètre **TARGET**
- Ainsi : **<A HREF=" doc1.htm" TARGET = "cadre3" >voir ce document </A>** affichera le document *doc1.htm* dans le cadre nommé *cadre3*.
- Pour adresser le document au cadre parent TARGET = "_parent" et pour revenir à la fenêtre complète, indiquer TARGET = "_top"

Petit exemple :

code	représentation
<- extrait document principal -> <FRAMESET COLS="50%,50%"> <FRAME SRC="fichier1.htm" name="cadre1"> <FRAMESET ROWS=" 30% , * " > <FRAME SRC="fichier2.htm" name="cadre2"> <FRAME SRC="fichier3.htm" **name="cadre3"**> </FRAMESET> </FRAMESET> <- extrait document **fichier2.htm** -> <A HREF="monchat.gif" **TARGET = "cadre3"** > et voici mon chat</A> <- extrait document **fichier3.htm** -> <PRE> Cliquez vite dans le cadre au dessus, pour admirer ... </PRE>	Ma page personnelle et voici mon chat Cliquez vite dans le cadre au dessus, pour admirer ... Ma page personnelle et voici mon chat 

Gestion des cadres en JS

L'objet frames[]

- Le navigateur crée un tableau **frames**, dont les éléments **frames[x], x=0,1,**, où 0,1, 2 est l'ordre d'introduction des cadres dans le fichier "parent". sont les cadres rangés dans leur ordre de déclaration. Le nombre de cadres est alors accessible par frames.length
- On peut ainsi nommer les cadres par frames[i] ou par leur nom. La référence complète pour utiliser les propriétés et les méthodes des objets situés d'un cadre, est :
  - **window.nom-cadre.objet.propriété**
  - **window.frames[num].objet.méthode()**
- Mais comment faire référence à une objet inclus dans un cadre à partir de code situé dans le document d'un autre cadre ?
  Les propriétés **self, parent, top** font référence respectivement au cadre courant, au cadre parent et à la fenêtre d'origine, ou au cadre partagé, dans le cas d'imbrication.

- Exemples :
  - Pour afficher "Bonjour !" dans la zone de texte nommée message, située dans le formulaire formu du document du cadre nommé cadre_bas :
    parent.cadre_bas.formu.message= "Bonjour !"
  - Pour afficher l'image N°4, déjà chargée en mémoire dans un tableau d'images du document courant, dans le 2ème cadre :
    parent.frames[1].location.href= tabImages[3].src

Gestion des liens

- On sait que l'objet documents posséde une propriété-tableau **links[]** qui rassemble tous les liens du document.
  Sa propriété target correspond à l'attribut target de la balise <A HREF=url ..>
- Exemples
  - Envoyer le document référencé par le lien dans le cadre nommé cadre_haut
    document.links[1].target = "cadre_haut"
  - Afficher le document (ou l'image) dans la fenetre nommée maFenetre window.fen = open("c-frames.html","**maFenetre**", options);
    document.links[0].target = "maFenetre";

Programmation objet en JS

Introduction

- Nous avons vu 3 classes définies dans le langage JS : **Date, String, Image**.
  On peut créer des objets de ces types, dans la terminologie des langages orientés-objets, on dit des instances de ces classes. En revanche, on ne peut pas créer d'objet de la classe **Math**
- Après avoir étudié ces objets prédéfinis et gérés par le langage lui-même, nous allons construire nos propres objets.
  Si la syntaxe et le vocabulaire sont proches de la programmtion-objet, JS n'est pas un langage orienté objet car il n'en admet pas les principaux concepts comme l'héritage, le polymorphisme ...
- Les objets JS sont surtout utilisés pour créer des structures de données plus complexes et plus souples que les tableaux de variables.

Pourquoi définir une classe de personne ?

Comment représenter et gérer un ensemble de N personnes, dotées d'un ensemble d'attributs comme nom, prénom, num tél, adresse complète, sexe, age, profession ...
Actuellement, il est inévitable de créer autant de tableaux de taille N que d'attributs.

```
var tabNom = new Array(N);
var tabPrenom = new Array(N);
var tabAnnee_naissance = new Array(N);
```

L'émiettement des données relatives à une personne dans de nombreux tableaux complique les traitements et apparait artificiel : que représente au fond le tableau des années de naissance ? et l'unité d'une personne tient au choix du même numéro d'indice dans tous les tableaux !

Dans la conception objet, la personne redevient le centre des préoccupations. On en fait une description générale dans la déclaration d'une **classe d'objets**, qui se compose de 2 parties qui décrivent ses propriétés et ses actions.

Exemple élémentaire

```
// La classe Rectangle posséde 2 propriétés
// Voici son constructeur
function Rectangle ( L, l) {
 this.largeur = l;
 this.longueur = L ;
}
// rect1 et rect2 sont des instances de la classe Rectangle.
// Les objets sont crées avec l'opérateur new.
var rect1 = new Rectangle(10 , 6)
var rect2 = new Rectangle(15 , 10.5)

// Lors de la création de rect1 par new, this est "remplacé" par rect1.

// Exemple d'utilisation
document.write("largeur = "+ rect1.largeur)
document.write("longueur = "+ rect2.longueur)

// la fonction suivante calcule l'aire du rectangle
function calcule_aire () {
  return this.largeur * this.longueur
}

/* On peut se servir de la fonction calcule_aire() pour intégrer
* une méthode dans la définition de la classe Rectangle
* la méthode est appelée aire
*/
function Rectangle ( L, l)  {
 this.largeur = l ;
 this.longueur = L ;
 this.aire = calcule_aire ;
}

// utilisation de la méthode
var a2 = rect2.aire())
document.write("L'aire du rectangle rect1 = "+rect1.aire() );
```

Définition d'une classe d'objets

On choisit tout d'abord un nom identifiant la classe de l'objet, souvent commençant par une majuscule. Par exemple Personne, Voiture, Maison, Article ...
Les éléments de ces classes autrement dit les objets, seront identifiés en minuscules

Propriétés
Puis, les noms des propriétés (ou variables d'objet) qui décrivent les caractéristiques stables des objets, même si les valeurs de ces propriétés changent.
Ainsi pour Personne, les propriétés retenues peuvent être nommées : nom, prenom, adresse, codePostal, ville, numTel, annee_naissance, solde ...
A tout moment un objet, c'est-à-dire une personne particulière, possède une valeur pour chaque propriété, valeur qui peut varier au cours du temps, comme son adresse ou le solde de son compte bancaire.

Méthodes
A coté de ces caractéristiques, les méthodes de l'objet décrivent son comportement, plus précisément les traitements auquels tout objet est soumis.
Par exemple, une personne peut calculer son âge, déménager, changer de N° de téléphone, avoir son compte crédité ou débité ...et avoir un nouveau-né !
Le plus souvent, ces méthodes d'objet agissent sur les propriétés et en modifient les valeurs.
Elles sont donc décrites par du code de fonction, intégré à la définition de l'objet.

Constructeur d'objets
En JS, la classe d'un objet est complètement décrite par une (seule) fonction particulière appelée **constructeur de l'objet**
La classe de l'objet et son constructeur portent le même nom.
Son écriture va contenir 2 parties : les déclarations des propriétés et des méthodes.

Schéma de constructeur
Le mot **this**, comme dans tous les langages orientés objets, permet de référencer les futurs objets de la classe.

```
//constructeur de la classe
function nom-classe(param1, param2 ...) {
// déclaration et initialisation des propriétés
this.propriété1 = param1 ;
this.propriété2 = param2 ;
.....
// affectation des méthodes
this.méthode1 = fonction1 ;
.....
}
```

Exemple de constructeur de *Personne*

```
//déclaration des fonctions-méthodes, elles peuvent suivre la fonction-constructeur
function calculer_age() {
.....
}
function mouvement_compte() {
.....
}
function nouvelle_naissance() {
.....
}
// Constructeur de l'objet Personne
function Personne(n, p, ad, cp, v, t, a, nb) {
// Definition des propriétés
this.nom = n ; this.prenom = p ;
this.adresse=ad ; this.codePostal=cp; this.ville=v; this.numTel=t;
this.annee_naissance = a; this.nb_enfant = nb ;

// Définition des méthodes
this.age = calculer_age ;
this.compte = mouvement_compte ;
this.naissance = nouvelle_naissance;
}
```

Utilisation de la classe d'objets

Construction d'objets
La classe d'objets étant décrite dans la fonction constructeur (plus les fonctions annexes, pour les méthodes de l'objet), comment l'utiliser ?
On doit distinguer 2 étapes :

1. création des objets
2. accès, modification de leurs propriétés, appel à leurs méthodes

Cette classe est une sorte de modèle selon lequel on va pouvoir créer des objets concrets, manipulables.
Ces objets concrets _ exemplaires tirés du modèle_ s'appellent les **instances** de la classe.

Pour créer une telle instance, on effectue un appel à l'aide de l'opérateur **new** au constructeur de l'objet, et on affecte le résultat à un identificateur de variable.
(Remarque : newa déjà été utilisé dans ce rôle pour créer des objets du type Date, Array..)

Ecriture générale
var instance1 = **new** nom-classe(liste de valeurs)
Les valeurs passées en paramètre vont affecter les propriétés de l'instance (et ainsi les initialiser), conformément au code du constructeur : ils doivent donc correspondre exactement, en place et en nombre, aux paramètres formels du constructeur.
Mais on peut alternativement appeler le constructeur par défaut, sans paramètre, reportant à plus tard l'initialisation des propriétés.

Exemple : création et utilisation d'instances de *Personne*

```
// création et initialisation complète d'une instance
var p = new Personne('Toto','Jules',1,'10 rue V.Hugo','93000','Bobigny','01222222',1950,3);
// ou création, puis initialisation d'une instance
 var p = new Personne();
 p.prenom= "Jules";
 p.nom= "Toto";
 .....
```

Prolongements

I. Tableau d'objets

Le type tableau en JS admet tout type d'éléments. On peut donc remplir un tableau de *Personne*

```
var personnes = new Array (N);
for (var i=0; i<personnes.length; i++)
   personnes[i] = new Personne();
personnes[0].nom = "Toto";
.....
```

II. Utilisation de fichier de code .js

- Il est très utile d'insérer du code standard dans plusieurs pages.
  En particulier les déclarations de classes peuvent reservir !
- Pour cela, par exemple sauvegarder tout le code de la classe Personne, en enlevant toutes les balises HTML, y compris <SCRIPT>, dans le fichier *personne.js*
  Dès lors il suffira d'importer ce code avec la directive <SCRIPT SRC=personne.js"> </SCRIPT>
- Remarques :
  On peut faire importer plusieurs fichiers, avec des directives semblables, ne pas énumérer les fichiers .js; de plus on peut écrire un bloc <SCRIPT> pour les définitions locales de fonctions.

III. Pour connaitre une classe

Pour connaitre les propriétés/méthodes d'un objet, on peut utiliser une fonction qui parcourera toutes ses propriétés, comme nous l'avons déjà fait pour les objets prédéfinis window, document ..

```
<SCRIPT LANGUAGE="JavaScript">
```

```
function  proprietes(objet,nom) {
var texte = "";
 for (var i in objet )
 texte +=nom+"."+i+" = " +objet[i] +"
";
document.write(texte)
}
// parcours des propriétés des instances de Personne
for (var i=0; i<personnes.length; i++)
   proprietes(personnes[i], personnes[i].nom)
</SCRIPT>
```

IV. Classe définie à l'aide d'autres classes

Soient 2 classes ClasseA et ClasseB. On suppose que :
ClasseB contienne une propriété de "type" ClasseA, ou plus exactement qu'une de ses propriétés soit construite par affectation d'une instance de ClasseA.

```
function ClasseA( n, ........) {
// Definition des propriétés de ClasseA
this.nom = n;
.....
}

function ClasseB( a, b, objetA, ....) {
// Definition des propriétés de Classe2
this.ppa = a;
this.ppb = b;
// objetA sera remplacé à la construction par
// une instance existante de la ClasseA
this.ppc = objetA;
.....
}
var instA = new ClasseA( ...);
var instB = new ClasseB( aa, bb, instA, ...);

document.write('La valeur de la ppté ppc de instB est : ',
   instB.instA.ppc);
```

Exemple
Soit une classe **Voiture** dotée des propriétés suivantes :

```
function Voiture(ma, mo, a, p) {
this.marque=ma;
this.modele=mo;
this.annee=a;
// le proprietaire est implicitement un "objet" de type Personne !
this.proprietaire = p; // p sera une instance de Personne
this.presenter=presenter_voiture;
}

var toto = new Personne('Toto', 'Jules', .....);
toto.presenter();
// Par cette construction, la propriété proprietaire devient une instance de Personne !
var auto = new Voiture('Renault','Clio',1997, toto);

// Maintenant posons quelques questions importantes !
// toto et auto se présentent avec leur méthodes particulières
```

```
toto.presenter();
auto.presenter();
// combien d'enfants a le propriétaire de auto ?
document.write('Le propriétaire de la voiture auto a ',
  auto1.proprietaire.nb_enfant ,' enfant(s)');
// quel est l'âge du propriétaire de (l') auto  ?
document.write('Voici l\'âge du propriétaire de la voiture auto : ',
  auto.proprietaire.age());
// mais laissons le  propriétaire de auto se présenter lui-même !
document.write('Présentons l\'heureux propriétaire de la voiture auto');
            auto1.proprietaire.presenter();
```

V. La propriété prototype

Cette propriété spéciale s'applique à une classe déjà construite et permet de lui ajouter de nouvelles propriétés et méthodes
On peut effectuer ces ajouts sur des classes prédéfinies ou définies par le programmeur
Il faudra ensuite attribuer une valeur à cette propriété, car on ne dispose pas d'un constructeur d'objet qui l'intégre. Syntaxe
nomClasse.prototype.nomPropriété = valeurInitiale ;

Exemples

1. ajout d'une propriété sexe à la classe Personne

Personne.prototype.sexe = "f" ;

2. ajout d'une méthode à la classe String

```
<SCRIPT>
// ajout d'une méthode à la classe String
function fin() {
return this.charAt(this.length-1);
}
String.prototype.fin = fin ;
</SCRIPT>
</HEAD>

<BODY>
<SCRIPT>
var ch1 = "Bonjour à tous"
var ch2 = "et à demain !"
document.write("Le dernier caractère de "+ "\""+ ch1+ "\"" + " : "+ch1.fin()+"<br>");
document.write("Le dernier caractère de "+ "\""+ ch2+ "\"" + " : "+ch2.fin()+"<br>");
</SCRIPT>
</BODY>
```

Gérer les cookies en JS

Pourquoi les cookies ?

On peut définir les cookies comme la "mémoire du WEB".

- Ils constituent la seule façon de s'affranchir de l'impossibilité d'écrire sur le disque du navigateur client (on rappelle que JavaScript ne possède pas d'instructions d'entrées-sorties fichiers).
- Le mécanime de pose de "cookies" a été imaginé pour permettre au moins à un serveur de retrouver certains résultats de sa consultation précédente du client.

- Pour cela, le serveur écrit sur le disque client quelques informations qu'il pourra chercher et réexaminer lors d'une prochaine visite du même "client".
- Ces renseignements sont stockés sous forme de chaine de caractères, dans un seul fichier au format texte, appelé cookies.txt par Netscape.
- Et ils restent stockés, à l'insu de l'utilisateur client le plus souvent pendant toute leur durée de vie, le "risque" dans ce dispositif étant l'effacement du fichier **cookies.txt** !

Observations

Vérifier le paramètre suivant de votre navigateur
Menu Edition/Préférences/Avancées/, activer :

- *"accepter uniquement les cookies quiseront renvoyés à leur serveur d'origine"*
- *"M'avertir avant d'accepter un cookie"*

La présente page abrite un cookie, à des fins de démonstration, nommé CompterPassageCoursCookie.

- Il contient seulement la chaine de caractères CompterPassageCoursCookie = n, où n est un entier qui compte le nombre de chargements de la présente page, par la même machine cliente.
- Recharger la page. A chaque fois on reçoit une demande du navigateur .
  Lisez bien ce message : il contient
  - exactement la chaine qui sera stockée et qui constitue le cookie
  - sa date de fraicheur
  - il sollicite votre décision : l'acceptez vous ?

Comment le cookie CompterLesPassages a t-il été noté ?

- Ouvrir le fichier *Program Files/Netscape/Users/default/cookies.txt*
- On y trouve une 1ère ligne du genre
- FALSE /D|/javascript/Jour4 FALSE 959810400
  CompterPassageCoursCookie 4

Enregistrement d'un cookie

Durant la session Web, chaque cookie est conservé en mémoire, il y a écriture sur disque à la fermeture du navigateur. L'écriture s'effectue sur instruction du serveur utilisant la directive HTTP suivante (incluse dans l'en-tête)
Set-Cookie
nom=*valeur*; [expires =*date*; path= *chemin-client*; domain= *nom-domaine*; secure]

- Seul le 1er champ, le **nom** identifiant le cookie et sa valeur (chaine de caractères stockée) sont obligatoires.
- date est la date d'expiration, au delà de laquelle le cookie sera supprimé du fichier
- chemin nom du rép. serveur
- nom-domaine nom du domaine Internet dans lequel le cookie peut être traité, les autres serveurs devant l'ignorer
- secure le cookie ne peut être consulté qu'au cours d'un échange sécurisé.

Ecrire un cookie

Voici le code gérant le cookie CompterPassageCoursCookie

```
<SCRIPT LANGUAGE="JavaScript">
// la date d'expiration
```

```
var futur = new Date(2005,1,1);

function CompterLesPassages () {
// fonction pour mettre en évidence le cookie
n = RechercherLeCookie("CompterPassageCoursCookie");
opts="width=200,height=20,,screenX=400,screenY=10"
fen = open("", "Delicious_Cookies",opts);
fen.document.write("Page chargée "+ n + " fois !<P>
          Vous reprendrez bien un de ces délicieux cookies ...");
fen.document.close();
id=setTimeout("fen.close();MiseAJourCookie (n);",2000);
}

function MiseAJourCookie (nb) {
// fonction de mise à jour du cookie
nb = eval(nb) + 1;
document.cookie = "CompterPassageCoursCookie=" + nb + ";
           expires=" + futur.toGMTString();
}

function RechercherLeCookie (unArgument) {
// recherche de notre cookie dans la chaine gérée par le navigateur
// cette chaine est une propriété de l'objet document
var chaine = document.cookie ;
unArgument = unArgument + "=";
var longueur = unArgument.length ;
var resultat;
if(chaine.length > 0) {
 debut = chaine.indexOf( unArgument , 0 );
 if (debut >= 0)   {
   fin = chaine.indexOf( ";" , debut+longueur );
   if (fin >= 0)
          resultat = unescape(chaine.substring(debut+longueur,fin));
   else
          resultat = unescape(chaine.substring(debut+longueur,chaine.length));
   }
 else resultat = 0;
 }
else resultat = 0;
return resultat;
}
```

Supprimer un cookie

Comment supprimer notre cookie CompterPassageCoursCookie ? La méthode la plus simple consiste à le réenregistrer en lui attribuant une date d'expiration dépassée !

```
<SCRIPT LANGUAGE="JavaScript">
// la date est dépassée !
var date = new Date(1999,0,1);
function EffacerCookie() {
//nom est le nom du cookie
document.cookie = "CompterPassageCoursCookie=" + ";expires=" + date.toGMTString();
}
```

Pour supprimer le cookie -->

Exercice

Etudions un autre exemple de cookie (réf. JavaScript/M. Dreyfus/Sybex)

- Dans une page page.html, lors de sa première visite, on demande à l'utilisateur sa couleur préférée, dans une liste incluse dans la page choixCouleur.html, on change alors en conséquence la couleur de fond du présent document.
- Un cookie, nommé couleur_du_fond est alors noté avec pour valeur le nom de la couleur choisie, il sera alors facile de la retrouver ... et de l'offrir au "client" lorsqu'il reviendra sur la même page !
- On demande d'étudier les scripts de page.html et choixCouleur.html (dossier Jour4/exo-cookies) et d'éventuellement les compléter.