

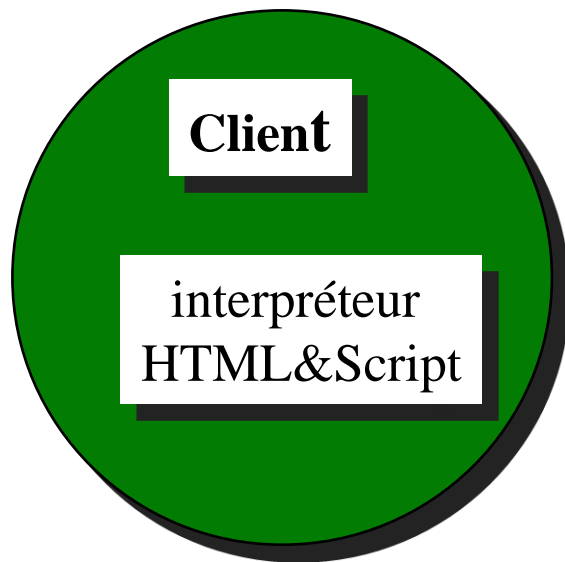
Programmation Client WEB

Cours - Interactions avec l'utilisateur en HTML et JavaScript

Jean-Michel Ilié

IUT Paris 5 - département informatique

Intérêts des scripts côté-clients



La page HTML est vue
comme une fenêtre applicative
avec un traitement local
(sans intervention du serveur web)

- ☞ évaluation d'information locale (devis, jeu, ...)
- ☞ aide à la saisie, ...

Toute application est possible !
Mais réservée à de petits programmes
(mise au point des codes interprétés !)

Avantage de la programmation événementielle HTML

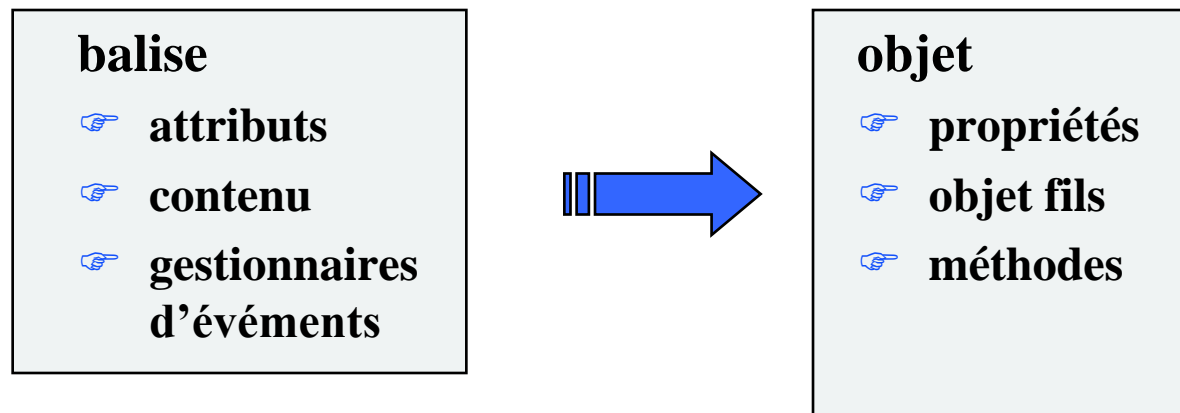
Une approche WEB2

La page chargée
n'est qu'une version initiale
de spécification

- ☞ Ergonomie dynamique et adaptée
 - Modification dynamique de la page et de son contenu
 - (des)-activation de gestionnaires d'événements
 - Gestion multifenêtres (lancement explicite de nouvelles fenêtres, dialogues).
- ☞ AJAX : dialogue explicite et évolué avec le serveur web
 - Requête HTTP explicite

Relation entre balises et données de script

Correspondance directe



Le contenu encadré par un jeu de balises est vu comme :
une **donnée textuelle** et
une **hiérarchie d'objets**,
Chaque objet pouvant être réactif à des événements.

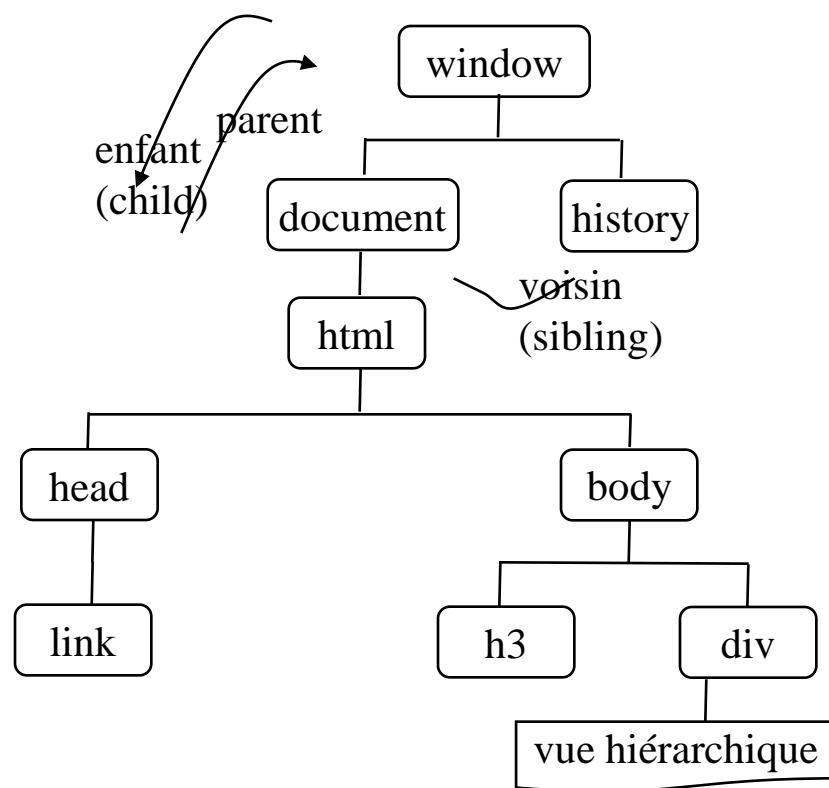
Exemple de hiérarchie d'objets pour une page html

```
<html>
  <head>
    <link/>
  </head>
  <body>
    <h3> vue hiérarchique </h3>
    <div></div>
  </body>
</html>
```

Un objet parent a des références sur ses enfants et vice versa.

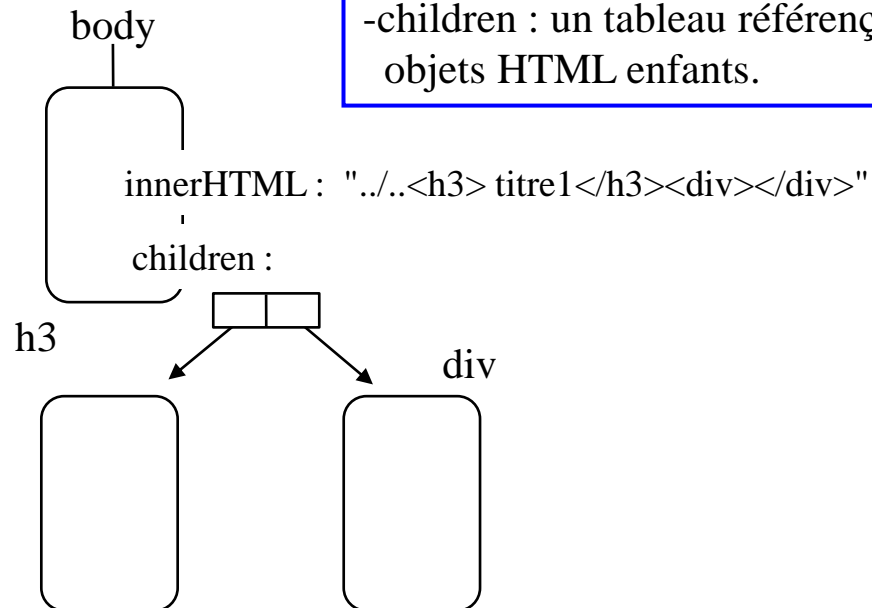
Les frères sont liés entre eux.

Type de noeuds : documents, éléments (balise), texte.



Objets HTML et contenu de balises

```
<html>
  <head>
    <link/>
  </head>
  <body>
    .....
    <h3> titre1 </h3>
    <div></div>
  </body>
</html>
```



Le contenu des objets HTML est stocké sous plusieurs formes dans l'objet lui-même:

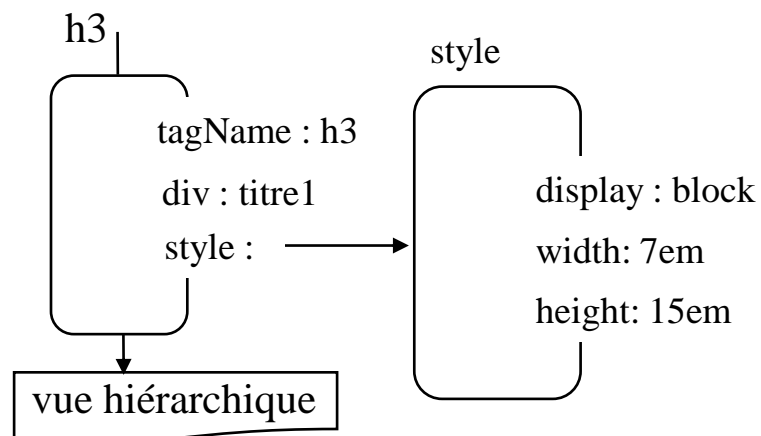
- innerHTML : une chaîne de caractères
- children : un tableau référençant les objets HTML enfants.

Objets HTML et style CSS des balises

```
<html>
  <head><style>
    h3 {
      width : 7em;
      height: 15em;
    }
  </style> </head>

  <body>
    <h3 id="titre">
      vue hiérarchique
    </h3>
    <div></div>
  </body>
</html>
```

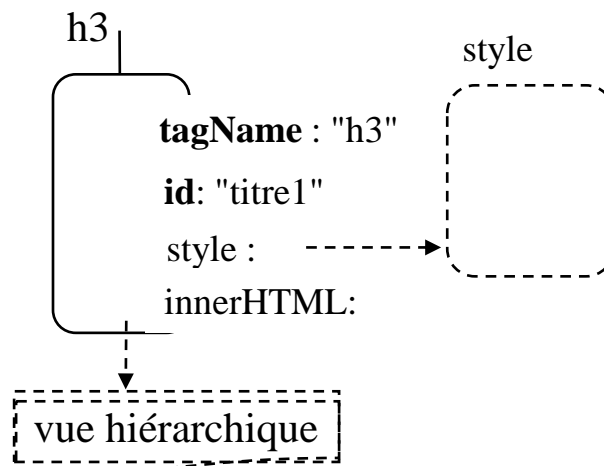
le style d'un objet
est encore un objet



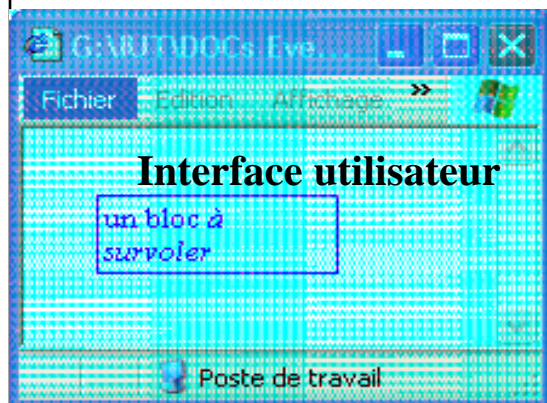
Objets HTML et attributs des balises

```
<html>
  <head>
    <link/>
  </head>
  <body>
    <h3 id="titre">
      <i>titre 1</i>
    </h3>
    <div></div>
  </body>
</html>
```

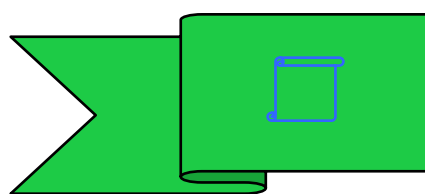
Les attributs d'une balise HTML sont représentés comme autant d'attributs dans l'objet HTML (sous forme de chaînes de caractères).



Rôle du système pour la programmation événementielle



↑ Inréractivité
Utilisateur



Application
Navigateur
en exécution

↑ ↑ ↑
Lecture événement :
Traitement séquentiel

↓ Réactions à l'évt

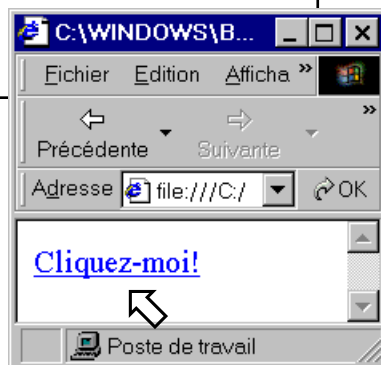
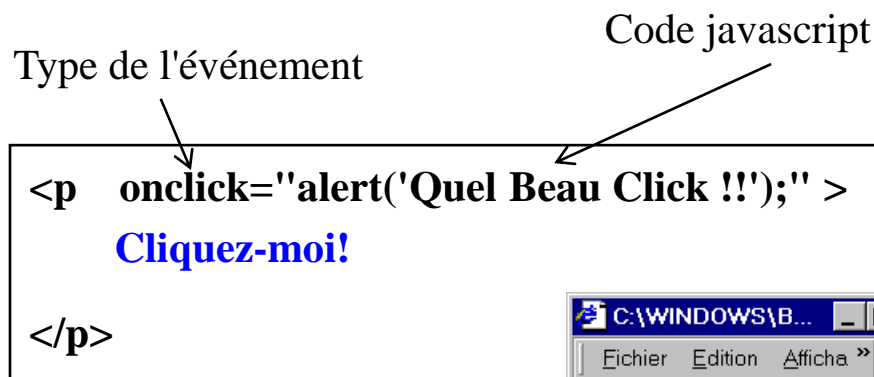
appels systèmes

Système

FIFO

Événements typés, caractéristiques
(Fenêtres -> Applications)

Déclaration des événements et du code réactif en javascript.



Déclaration au niveau de toute balise HTML, sous formes d'attributs spécifiques.

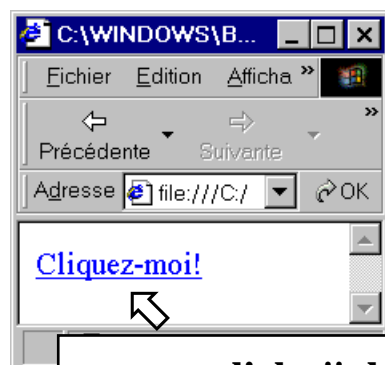
Plusieurs déclarations possibles pour une même balise.

Certains événements requièrent l'action de l'utilisateur sur la partie visible de la balise.

alert(...) affiche une fenêtre d'information.

Exemple de réactions

clic sur un paragraphe simulant un lien



```
<p onclick="alert('Quel Beau Click !!');">
Cliquez-moi!
</p>
```

affiche



alert(...) affiche une fenêtre d'information sur le clic

Propagation des événements (suivant la vue hiérarchique)

Event :

**le navigateur récupère le dernier événement non traité
sous forme d'une structure de données informative**

- ☞ position de la souris (coordonnées),
- ☞ type d'événement (load, click, ...)
- ☞ ...

Bouillonnement :

propagation de Event dans la hiérarchie des objets HTML

- ☞ identification des objets-balises concernés

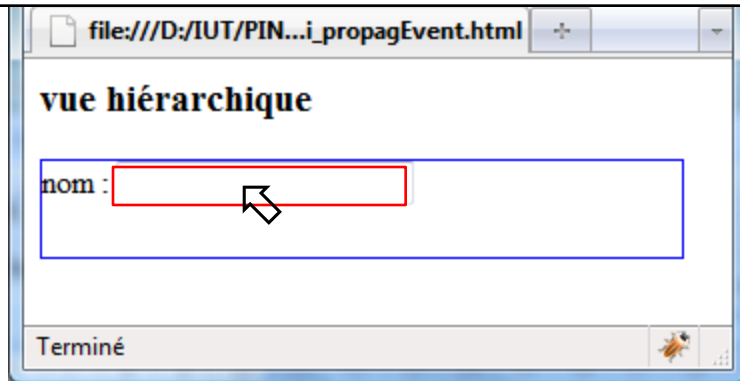
Exécution :

**Phase d'exécution des méthodes de réaction
pour les objets HTML concernés**

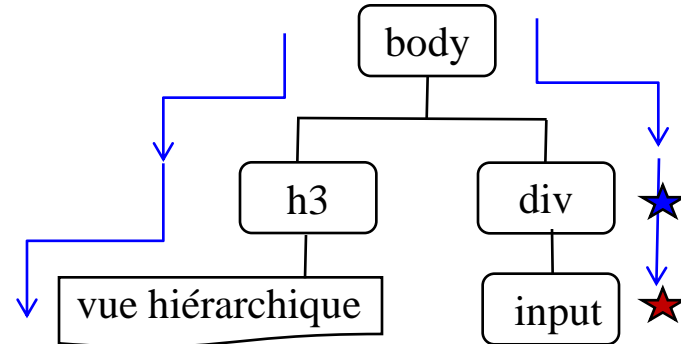
- ☞ par défaut, l'exécution est faite lors d'une propagation arrière dans la hiérarchie.

Exemple de propagation événementielle

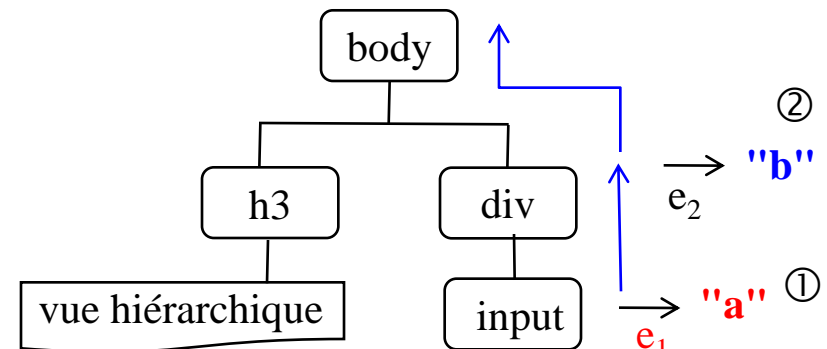
```
<html> <body>
  <h3> vue hiérarchique </h3>
  <div onclick="alert('b')">
    nom : <input onclick="alert('a')"/>
  </div>
</body></html>
```



Propagation



Remontée



Quelques attributs de déclarations événementielles en HTML

Evénements particuliers des balises A, UL-LI, ...

onClick // sur un CLIC gauche

Evénements de survol des conteneurs de texte

onMouseOver // réaction au moment du survol du contenu

onMouseOut // réaction si le contenu n'est plus survolé

onMouseMove // déplacement de la souris (coordonnées)

Evénements de clavier

onKeyDown // réaction quand une touche est enfoncée

onKeyUp // ou quand la touche est relevée

onKeyPress // touche enfoncée

Evénements particuliers de la balise BODY

onLoad // à l'affichage de la page (chargement)

onUnload // au moment de son déchargement

onFocus // activation de la fenêtre

Exemple de réactions évoluées

☞ survol d'une div:

L'événement et sa réaction sont placés dans la balise <div>

```
<div      onmouseover="montre()"
          onmouseout="cache()">
    texte HTML
</div>
```

☞ survol d'une image :

Comme pour un lien, il est possible de réagir dessus par un survol (ou un clic,)

```

```

☞ lancement d'un programme :

on peut initialiser les données d'un programme javascript au chargement de la page HTML

```
<html>
<body  onload="initProg()"
    texte HTML
</body></html>
```

Les fonctions en rouge seront à écrire en javascript.

Déclaration de code JavaScript

Plusieurs zones de scripts possibles

▲ *Séparation de code HTML et javascript :
Préférer les déclarations en fichiers externes
Plutôt que les écritures internes.*

☞ *Script en fichier externe et déclarations*

- *fonctions de réactions aux événements*
- *fonctions secondaires*
- *variables globales globalement utiles*
- *code non-encapsulé, à exécuter directement (rare !)*

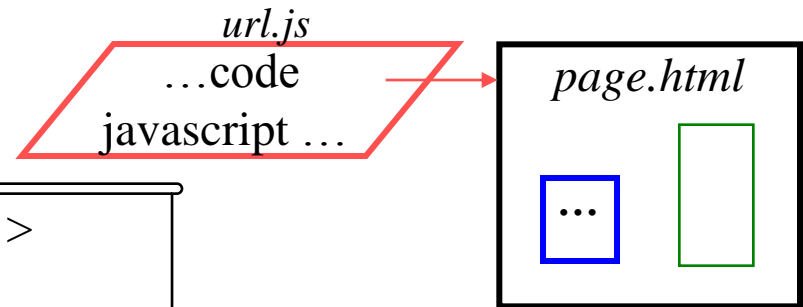
☞ *Script interne au fichier HTML (prioritaire)*

- *souvent placé dans l'entête de la page html*
- *au sein du body, mettre le code du script dans un commentaire HTML*

Déclaration HTML d'inclusion de code JavaScript

👉 *Script en fichier externe,*

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



👉 *Script interne*

```
<script type="text/javascript">  
    <!-- ...place du code JS ...  
    -->  
</script>
```

- 👉 Les 2 écritures sont compatibles.
- 👉 Commentaires HTML si le script est dans body.
- 👉 Priorité au dernier code interprété (cas de conflit).
- 👉 Contexte d'exécution globale sur la page HTML

Contexte d'exécution du code JavaScript

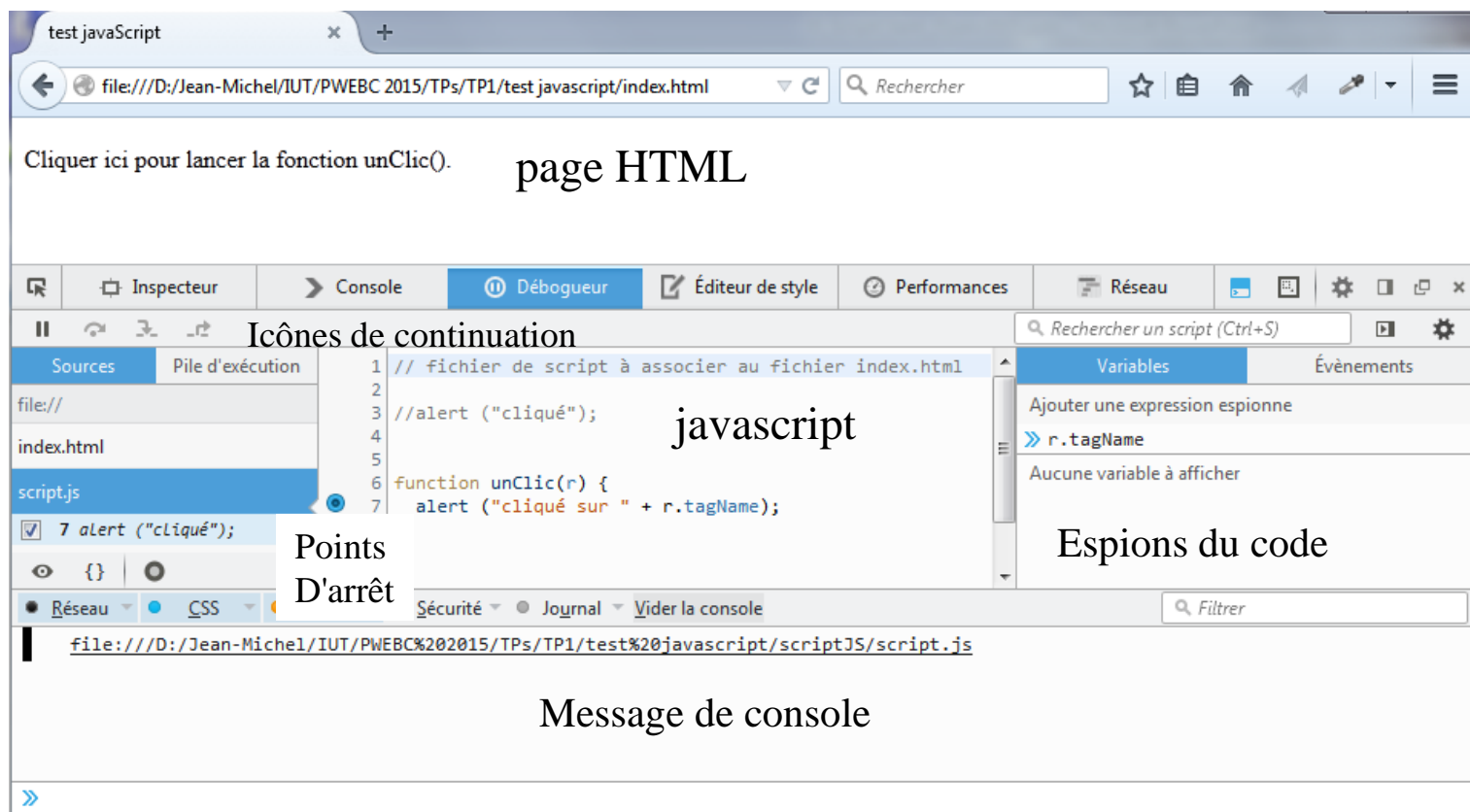
- ☞ Contexte d'exécution globale sur la page HTML
- ☞ L'interprétation javascript suit
 - l'ordre des déclarations de scripts dans le fichier HTML.
 - l'ordre d'apparition des événements (appels fonctions réactives).

Outils de développement HTML et Javascript

- IDE comme **Netbeans** ...
- **Examineur** d'éléments : débogueur interne au navigateur
 - ☞ Points d'arrêts
 - ☞ contrôle du déroulement d'exécution
 - continuation au point d'arrêt suivant, pas à pas
 - ☞ Visualisation des valeurs d'attributs : survol d'attributs
 - ☞ "Espion" : Maintien de visualisation d'une référence ou d'un attribut (décrit par une expression javascript).

Aperçu du débogueur sur le navigateur

Sélection
fichier



Programmation Javascript de la spécification HTML

window : référence primitive sur un objet mémoire

window caractérise les données du navigateur (en fait une hiérarchie d'objets).

Toutes les données du navigateur sont mémorisées comme des attributs d'objets.

- . donnée simple et collection (tableau)
- . référence d'objet (hiérarchie)
- . fonction *//une fonction est défini pour un objet et le nom de la fonction est un attribut
//référençant en mémoire du code à exécuter.*

Accès à un attribut : référenceObjet . nomAttribut (r1 . a2)

Enchaînement possible via les attributs qui sont des références d'objets (r . r1 . a2)

La référence window peut être omise (window.a et a sont identiques).

Suppression dynamique,

- ☞ déréférencement d'un objet mémoire : affectation à **null** des références
- ☞ **garbage collector** : toute zone mémoire non référencée est récupérée.

Intérêt de l'objet window

Référence sur l'objet
mémoire représentant
la page HTML,
entête comprise.

window

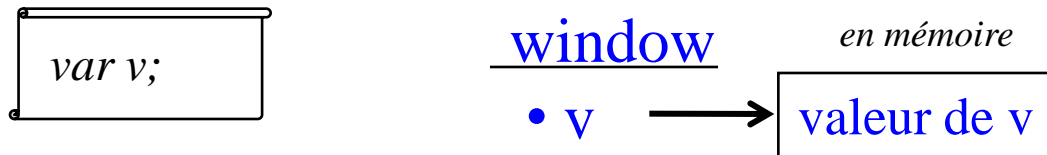
☞ attributs

- ☞ variables globales du programme javascript
- ☞ variables-référence des premiers objets de la page HTML

☞ méthodes

- ☞ variables-référence des fonctions javascript,
- ☞ il y a une fonction de réaction associée au chargement (**load**) et déchargement (**unload**) des objets balises en mémoire.

Variables globales et objet window



☞ Accès équivalent au contenu d'une variable (référencement de la zone mémoire):

`v` ou

`window.v`

☞ une **seconde déclaration globale** de la variable remplace la première.

Fonctions JavaScript et objet window

☞ Référencements équivalents

```
function F1 () { ... }
```

```
window.F1 = function () { ... }
```

☞ Référencements équivalents

```
F1
```

```
window.F1
```

☞ Exécutions équivalentes :

```
F1(a0,a1,..)
```

```
window.F1(a0,a1,..) .
```

```
function F1(arg0, arg1, ...) {  
  var v; ...  
  inst; ...  
}
```

window
• **F1**

en mémoire

```
function (arg0, arg1, ...) {  
  var v; ....  
  inst; ...  
}
```

Le nom d'une fonction (globale) est un attribut dans l'objet window,
Cet attribut référence la fonction en mémoire.

Le nom de fonction est la **signature** de la fonction.

Une **seconde déclaration** de la fonction pour un objet remplace la première.

Référence sur l'objet HTML courant

this

Approche dynamique :

Au moment d'un événement,

l'objet source de l'événement est l'objet courant.

Exple : `<a href ="http..//"
 onMouseOver="alert (this.text)"
 onClick="v=this.href"
 > cliquer`

Exemple de correspondance mémoire

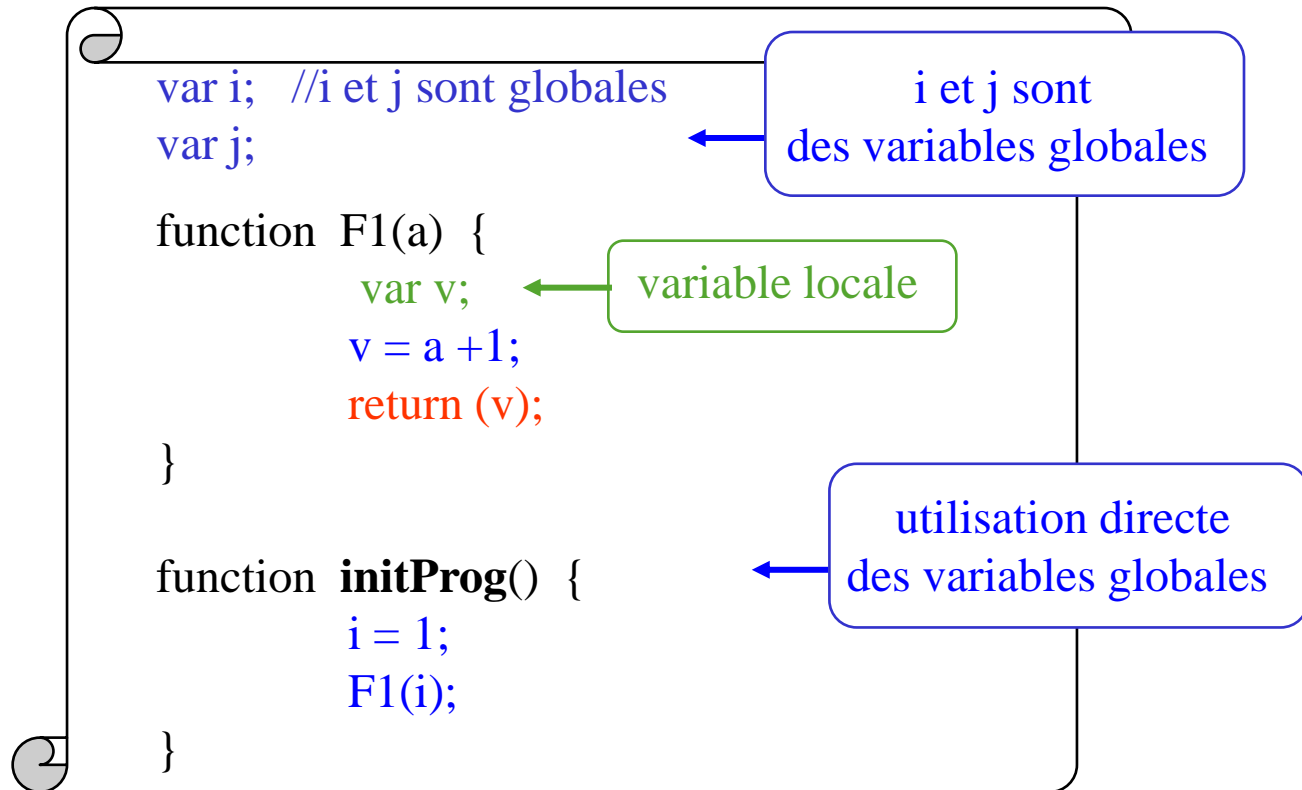
```
var i; //i est globale  
function F1() {...}
```

```
function F2() {  
  {  
    var window.j= 0; //j est globale  
    var v=F1; // v et w sont loc  
    var w= window.F1;  
    ....  
  }  
}
```

déclarations globales
en externe ou interne

utilisation directe
de variable globale

Ecriture classique du code javascript



Référence aux objets HTML (2)

Référence sur l'objet document HTML Référence sur l'objet document
☞ **window.document** ——— (inclut une référence sur body et head)
☞ **document** ——— Equivalent : l'objet document est toujours
supposé être un objet référencé par window.

Méthodes d'accès direct définies comme attributs de l'objet document

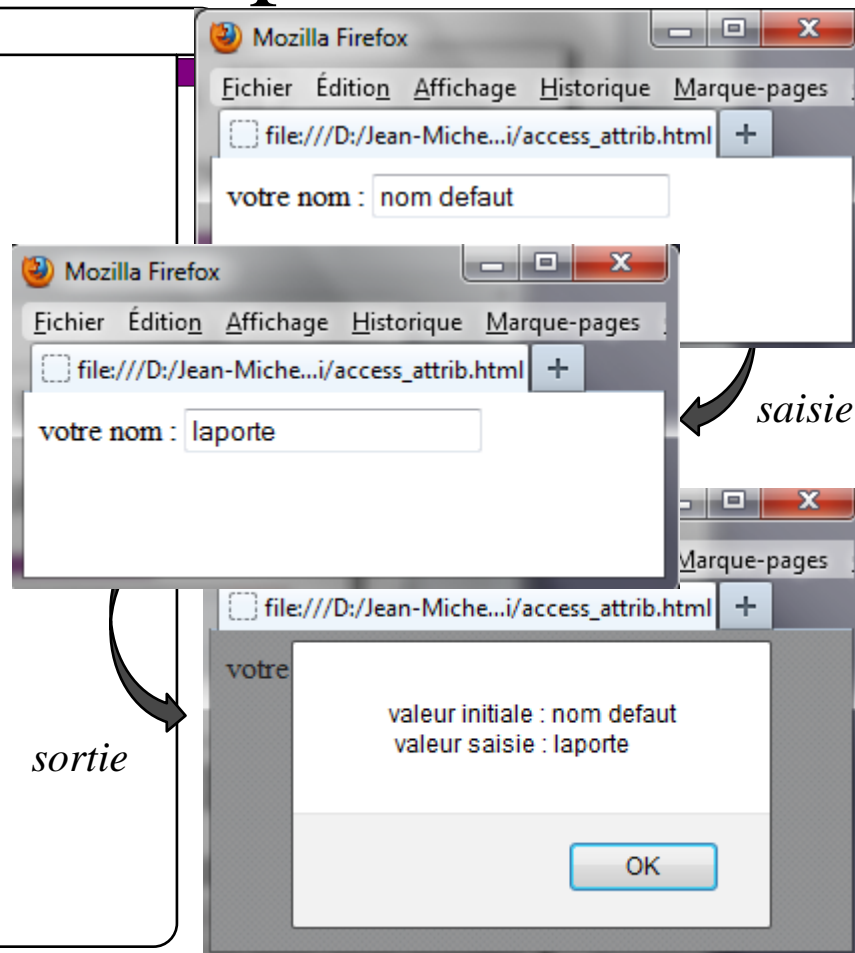
☞ document.**getElementById**('i') ——— Rend une référence sur l'objet (id="i")
☞ document.**getElementsByTagName**('n') ——— Rend un tableau de références sur la
collection d'objets (name="n").
☞ document.**getElementsByTagName**('t') ——— Rend un tableau de références sur la
collection des objets balises de nom "t"

```
document.getElementById('nom'); //sur <... id="nom">  
document.getElementsByTagName('nom')[0]  
document.getElementsByTagName('div')[1]
```

Exemple d'accès par ID à une balise input

```
<html> <head>
<script type="text/javascript">
  function acces() {
    var o=document.getElementById('N');
    alert("valeur initiale :" + o.defaultValue +
        "\n valeur saisie :" + o.value);
  }
</script> </head>

<body><form>
  <input id='N' value='nom default'
    onchange="acces();" />
  nom<br>
</form></body></html>
```



Exemple applicatif (1): modification de contenu div

```
<html><head><script type="text/javascript">
function changeBulle(MsgBulle) {
    var o=document.getElementById('MSG');
    o.innerHTML=MsgBulle;
}</script></head>

<body><h2> Ayez la bonne Bulle !</h2>
<p> Choisissez votre texte :
<ul><li><a onClick="changeBulle('Bien !')">Bien !</a>
<li><a onClick="changeBulle('Zut !')">Zut !</a></li>
<li><a onClick="changeBulle('arg !')">arg !</a></li>

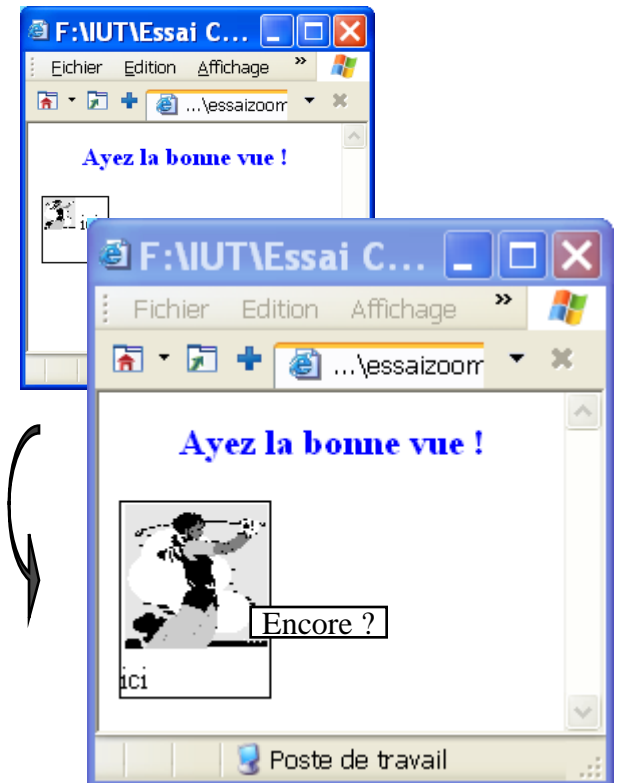
<div id="aff">
    
    <p id="MSG">....?...</p>
</div> </body> </html>
```



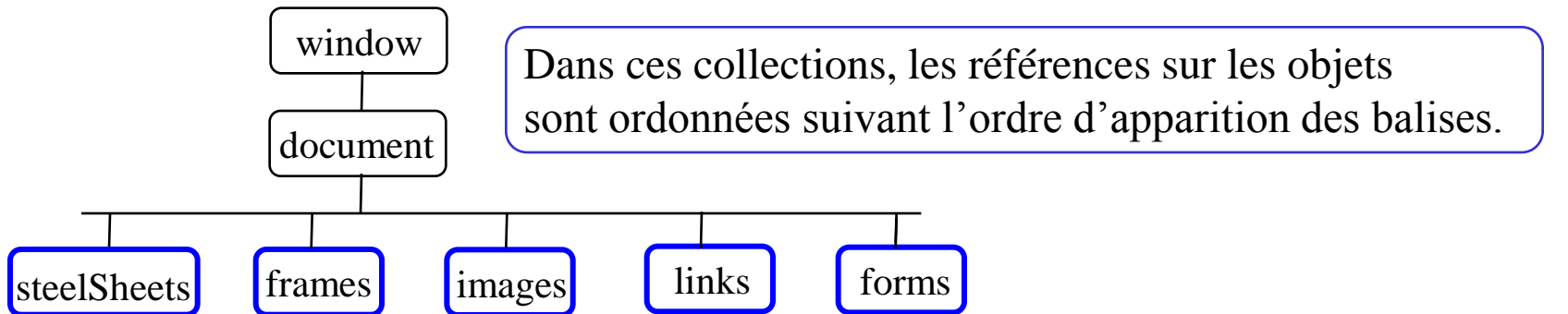
Exemple applicatif (2) : un effet de zoom

```

<html> <head>
<link href="style.css" rel="stylesheet" type="text/css">
<script type="text/javascript">
var pas=50;
function zoomImg(idImg) {
    var o=document.getElementById(idImg);
    o.width+=pas ;
    o.height+=pas ;
    o.alt="encore ?";
}
</script> </head><body>
<h3 id="titre">Ayez la bonne vue !</h3>
<div id="aff" onclick="zoomImg('golf');">
     ici
</div> <!--fin div "aff"-->
</body></html>
    
```



Accès aux balises via des objets-collections



Recherche d'un objet dans la collection "col"

- 👉 `document.col [indice]`
- 👉 `document.col ["idObjet"]`

Une collection de références sur des objets est vue comme

- 👉 un tableau indicé
(1er indice 0)
- 👉 un tableau associatif
(sur l'attribut ID des balises)

Exemple de réactions sur les collections inversions d'images en JavaScript

```
<html><head>
```

```
<script type="text/JavaScript">
```

```
function flip() {
```

```
// on intervertit les 'src' des images n°0 et 1 du document
```

```
var tampon=document.images[0].src;
```

```
document.images[0].src=document.images[1].src;
```

```
document.images[1].src=tampon;
```

```
}
```

```
</script> </head>
```

```
<body> .....
```

```
<a href="#" onmouseover="flip();">
```

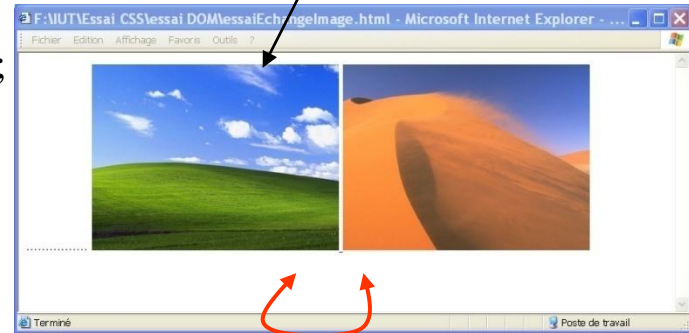
```
     </A>
```

```
<a href="#" onmouseover="flip();">
```

```
     </A>
```

```
</body> </HTML>
```

*document.images[0].src=
document.images['img0'].src*



Méthodes "événements" des objets-balises

Forme générale `o.typeEvt()`

// ici 'o' la référence de l'objet balise.

Ce sont de méthodes prédéfinies
des objets-balises
pour lancer directement / simuler
la réaction à un événement sur la balise

o.click() *//réalise la réaction au clic sur le contenu de la balise*

o.focus() *//met le focus sur le contenu (exple un champ de saisie)*

o.select() *//réalise une sélection partielle ou totale du contenu*

o.blur() *//dé-sélection du contenuZ*

o.submit() *//soumission du formulaire ...*

Exple: soumission du 1er formulaire (numéro 0)

`document.forms[1].submit();`

Méthodes "événements" des objets-balises

Forme générale `o.on`typeEvt()

//balise référencée par 'o'

`o.onclick() {}`

`o.onfocus() {}`

`o.onselect() {}`

`o.onblur() {}`

`o.onSubmit() {}`

Partie programmable
des méthodes de réaction prédéfinies.
Correspond au code de réaction précisé
dans les balises (onclick,).

Note : une telle fonction qui se termine en
retournant **false** stoppe tout :

- le processus d'exécution
- la propagation de l'événement

Exple de formulaire qui ne partira jamais :
<form id='frm' **onsubmit**="return false;">
</form>

Comment externaliser la déclaration des gestionnaires d'événements

Principe de séparation du code HTML et du Javascript

- ✓ **Gestionnaires d'événements**
à déclarer dans un fichier secondaire javascript
et non plus dans les balises.

Déclaration des gestionnaires des objets HTML en JS

☞ pour ces objets, redéfinir le code des fonctions **onclick()**,

Interprétation de ces déclarations au chargement de la page HTML

☞ redéfinir la fonction **onload()** de l'objet window
avec les redéfinitions des gestionnaires

Comment déporter les déclarations de gestionnaire d'evt dans un script

```
<html> <head>
<script type="text/javascript">
window.onload = function() {
    var o=document.getElementById("blocA");
    o.onclick= function () {
        alert ('unClic');
    }
}
</script></head>
<body >
<div id="blocA" style="width=20em">
    cliquer ici !</div>
</body> </html>
```

Au chargement,
la fonction onload
redéfinie est exécutée.
Elle va servir à
redéfinir la fonction
onClick
de l'objet référencé
(ici un div)

Aucune déclaration
de gestionnaire
dans la balise html

Exemple plus structuré mais ... plus bavard

```
<html> <head>
<script type="text/javascript">
function afficheCLIC () {
    alert ('un clic');
}
window.onload = function() {
    o=document.getElementById("blocA");
    o.onclick= afficheCLIC;
}
</script></head>

<body > <div id="blocA">
    cliquer ici !
</div>

</body> </html>
```

fonction dite anonyme
car non rattachée à un
objet
particulier.

o.onclick et affichClic
réfèrent la même zone
de code mémoire.

D'un point de vue
générique,
affichClic pourra servir
à spécifier onclick de
plusieurs objets HTML.

Déclaration avancée des gestions d'événements

Sur tt objet-balise référencée (o)

o.**addEventListener**

o.**removeEventListener**

3 paramètres :

- type evt
- référence à 1 fct
ou déclaration directe par fct anonyme
- booléen.

Exemple : 2 fonctions lancées après chargt des objets balises

```
document.addEventListener("load", fonction_1, false);  
document.addEventListener("load", fonction_2, false);
```

Déclaration avancée : les propriétés de l'événement

Sur tt objet-balise référencée (o)

o.**addEventListener**

o.**removeEventListener**

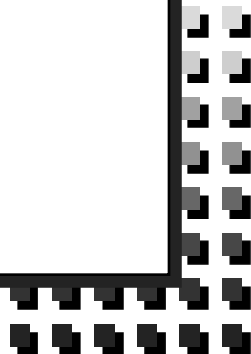
3 paramètres :

- type evt
- référence à 1 fct
ou déclaration directe par fct anonyme
- booléen.

Exemple : 2 fonctions lancées après chargt des objets balises

```
document.addEventListener("load", fonction_1, false);  
document.addEventListener("load", fonction_2, false);
```


Bases de la programmation en javascript



Données élémentaires des programmes JavaScript

Types élémentaires

- ☞ **booléens** true ou false
- ☞ **nombres** 10 ou 3.14 //pas de différence entre entier et réel
- ☞ **chaîne** "ch" ou 'ch' //à utiliser les deux en alternance
- ☞ **chaîne formatée** //"\f nvle page \n \t nvle ligne décallée
- ☞ **valeur nulle** null //la valeur nulle (pour objet, tableau, ...)

nom_variable
=
<lettre ou _>.<lettre ou _ ou chiffre>

```
•      var ch="01";  
•      var nb=20;  
  
var x=ch+nb  
var y=nb+ch  
var z=nb+"_err"
```

Variables implicitement typées

```
// x="0120" chaîne  
// y=2001    entier  
// provoque une erreur
```

Opérateurs Javascript (similaire au C ...)

☞ Opérateurs arithmétiques

++, -- i++
- (expr_arithmétique)
x%y Reste de la division de x par y
eval(expression_chaine)

☞ Opérateurs d'affectation

=, +=, // x +=y donne x=x+y
autre : -=, *=, /=, %=

☞ Opérateurs logiques

&& ET logique
|| OU
!(expression) négation
== et != test d'égalité et d'inégalité

☞ Opérateurs logiques de bit

&	ET
	OU
^	OU exclusif
<< ou >>	décalage

Expressions conditionnelles et itératives classiques

if (condition) {instructions_si_ok;}

else {instructions_sinon;};

Ou (condition) ? (instructions_si_ok;) : (instructions_sinon;);

case var

val1: ...; break;

val2: ...; break;

default;

while (condition) {instructions;}

☐ **break**; //fin de bouclage

☐ **continue**; //saute l'itération courante

for (expr_initiale; condition; mise_à_jour) {instructions;}

Objets en JavaScript

propriété et méthodes

Création d'un objet

👉 `stg1 = new nomClasse(args);`

Le constructeur crée un objet et affecte `this` comme référence à cet objet.

Accès à la propriété *prop* d'un objet *stg1*

👉 `stg1.prop`

👉 `stg1['prop']` //syntaxe aspect tableau

👉 `stg1[ind]` //indice ind de la prop prop

Invocation de méthode

👉 `stg2.affiche();` //lance la méthode affiche

Propriété par défaut :

👉 `var nb = stg1.length;` // nombre d'éléments de l'objet

Tableaux en JavaScript

Un tableau est un objet mémoire

👉 **new** sur le type `Array()`

Taille d'un tableau

👉 Propriété **length**

```
var lesMois = new Array("Jan", "Fév", ... "Déc");
```

```
var lesMois = new Array() ;  
lesMois[0]="Jan"; lesMois[1]="Fév"; ...; lesMois[11] = "Déc";
```

Tableau indicé

```
var lg= lesMois.length;
```

```
var tabAssoc = new Array();  
tabAssoc['nom']= 'Meurisse';  
tabAssoc['prenom']= 'Paul';
```

Tableau associatif

//1er index : nom

//2ème index : prenom

Notations JSON des objets et tableaux

Notations JSON

$[v_1, v_2, v_3]$ tableaux

$\{a : v_1, b : v_2, c : v_3\}$ objets ou tableaux indicés

Les notations peuvent s'imbriquer dans les valeurs.

```
var t= ["Jan", "Fév", ... "Déc"];
```

```
var t={nom: "Meurisse" , prenom: "Paul"};
```

```
var t= [ { nom: "Meurisse" , prenom: "paul" },  
         { nom: "Cardinal" , prenom: "Claudia" }  
       ] ;
```

Tableaux JavaScript

Un tableau dynamique est un objet

👉 **new** sur le type `Array()`

Taille (comme tout objet) :

👉 Propriété **length**

```
var lesMois = new Array("Jan", "Fév", ... "Déc");
```

```
var lesMois = new Array() ;  
lesMois[0]="Jan"; lesMois[1]="Fév"; ...; lesMois[11] = "Déc";
```

Tableau indicé

```
var lg= lesMois.length;
```

lesMois

0	Janv
1	Fév
...	...
11	Déc

```
var tabAssoc = new Array();  
tabAssoc['nom']= 'Tisse';  
tabAssoc['prenom']= 'Paul';
```

Tableau associatif

//1er index : nom

//2ème index : prenom

cv

<i>nom</i>	<i>Tisse</i>
<i>prenom</i>	<i>Paul</i>

Technique d'itérations sur des objets ou tableaux

for (indice **in** tableau)
{instructions;}

for (indice **in** objet)
{instructions;}

Parcours indicé
(indijage automatique
avec la boucle for).

```
var result="r&ecute;sultat:", i="";
```

```
for (i in cv)
```

```
    result += "\n" + "cv" + "." + i + " = " + cv[i];
```

```
document.write("result");
```

```
//résultat :  cv.nom = "Tisse"  
              cv.prenom = "Paul"
```

Technique d'itérations sur des objets ou tableaux

for (element **of** tableau)
{instructions;}

for (element **of** objet)
{instructions;}

Parcours non indicé,
obtention des éléments
de la collection

```
var result="r&ecute;sultat:", e="";
```

```
for (e of cv)  
    result += e + " " ;  
document.write("result");
```

```
//résultat : "Tisse" "Paul"
```

Gestion explicite des exceptions (récupération sur erreur)

try

{... instructions possiblement erronées...}

catch (err)

{... instructions exécutées en cas d'erreurs...}

try :

permet d'échapper
une erreur

catch (err) :

Fonction exécutée

en cas d'erreurs dans try.

- **err.name** : nom de l'erreur
- **err.message** : message d'erreur
(err paramètre formel du catch) .

Exemple de récupération sur erreur

```
<SCRIPT><!--
```

```
try
```


```
{ document.writeln("Pour l'instant tout se passe bien...  
    eval("15+*3")  
    document.writeln("affichage si ok<br/>")  
}
```

```
catch (err)
```

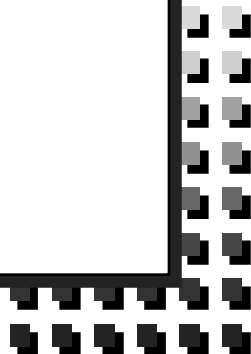
```
{  
    document.writeln("Une exception a eu lieu !<br>")  
    document.writeln("Nom de l'exception : " + err.name + "<br>")  
    document.writeln("Message d'erreur reçu : " + err.message)  
}  
// --> </SCRIPT>
```



Quelques annexes sur les objets javascript prédéfinis



- ☞ Gestion des chaînes de caractères
- ☞ Gestion des fenêtres et des documents chargés
- ☞ Accès aux informations du navigateur
- ☞ Gestion des temporisations



Annexe : Objet chaînes de caractères

- String -

- ➡ + // -----> concaténation de chaîne (aussi `ch3= ch.concat (ch2)`)
- ➡ `ch.charAt(indice)` //Extraction d'un caractère sur indice (de 0 à `length-1`)
- ➡ `ch.indexOf(ss_chaine)` //indice_début de la 1ère sous chaîne
- ➡ `ch.indexOf(ss_chaine, indice_début)` //idem à partir de `indice_début`
- ➡ `ch.lastIndexOf(ss_chaine)` //indice_début de la dernière sous chaîne
- ➡ `ch.slice(indice_dep, indice_fin)` //extraction de sous chaîne - anciennement ***substring()***
- ➡ `ch.toLowerCase()` ou `toUpperCase()` //vers minuscule ou Majuscule
- ➡ `eval (chaîne_expressionJavaScript)` //évaluation d'une 'expressionJavaScript'
//donnée sous forme de chaîne de caractères

Chaîne de
caractères
(objet String)

```
ch='verif_' + 1 + '()';  
res = return eval("ch");
```

Traitement des tableaux de chaînes

Transformation entre chaîne et tableau

☞ `ch. split(tab, "separateur" [, NbMaxSplit])` //créer un tableau 'tab' des sous chaînes
//suivant la chaîne 'séparateur'

☞ `ch=tab.join("separateur")` //linéarise tab en une chaîne 'ch', en intercallant 'séparateur'

<code>tab=['a','b/c'];</code>	<code>ch= tab.join(":=");</code>	<code>//ch = "a:=b/c"</code>
	<code>ch.split (tab, '/');</code>	<code>//tab=["a:=b", "c"]</code>

Concaténation et extraction de tableaux

☞ `ch3= ch.concat (ch2)` //les tableaux 'ch' et 'ch2' sont concaténés pour donner 'ch3'

☞ `tab.slice ('debInd', 'FinInd');` //isole un sous tableau du tableau 'tab' de 'debInd' à 'FinInd-1'

Tri d'un tableau

☞ `tab.sort (fctCompare [,param[,param2]]);` //tri le tableau 'tab' suivant la fct 'fctCompare'
//Cette fct est à définir et peut posséder 2 paramètres

☞ `tab.reverse ()` //inverse l'ordre

Exemple de vérification

- adresse mail -

```
function VerifEmail (email) {  
    if (email.indexOf('@',0) == -1 || email.indexOf(';',0) != -1)  
        {alert("adresse incorrecte");  
        return false;}  
    else {return true;}  
}
```

```
// continuer pour tester l'absence de  
/ : < > | ' ' & $ ! " ( ) [ ] { } #
```

```
<form><input name=Email  
    onFocus="aideContextuelle('votre email professionnelle, svp');"  
    onBlur="verifEmail(this.value);"</form>
```

aideContextuelle est une fonction supposant lancer une fenêtre d'aide

Exemple de vérification

- saisie des nombres -

```
function VerifNombre (chaine, min, max) {  
  if (chaine=="") {alert ("un numéro svp."); return false; }  
  for (var i=0; i<chaine.length; i++) {  
    var ch = chaine.substring(i,i+1);  
    if (ch <"0" || ch>"9") {alert ("un numéro svp."); return false; }  
  
    var num = 0 + chaine ; //conversion équivalente à num=eval(chaine);  
  
    if (num<min || num>max) {  
      alert("svp. un nombre entre " +min+ " et " +max);  
      return false;  
    } else return true;  
  }  
}
```

Expression régulière de chaîne de caractères

Définition de modèles pour tester les chaînes

CREATION d'UN MODELE (*différentes méthodes*)

☞ `chReg = /modele/ ;` *//exploitation prévue pour la 1ère occurrence*
`chReg = /modele/g ;` *//exploitation pour toutes occurrences*
`chReg = /modele/i ;` *//exploitation sans tenir compte de la casse*
`chReg = /modele/gi ;`

idem

`chReg = /+@+/g`

☞ `chReg = new RegExp('modele') ;`
`chReg = new RegExp('modele', 'gi') ;`

Construire un modèle de chaîne

Indication de positionnement :

[^]m *modèle commençant par ch*

^{\$}m *modèle finissant par ch*

[^]m^{\$} *modèle ch exactement*

ch1 | ch2 *choix entre 2 modèles*

Classes pour un (type) de caractère

• // décrit un caractère quelconque.

[a-zA-Z_.\-] | [0-9] // un caractère parmi les lettres et {'_', '.', '-'} **ou** parmi les chiffres.

[^a;\|] // ^ pour la négation : un caractère sauf 'a' ';' '\' et '/'

\d et \D // équiv à [0-9] et l'opposé d'un chiffre

\w et \W // équiv à [a-zA-Z0-9_] et l'opposé

\s et \S //(espace, retour, tabulation) et l'opposé.

Précision d'une multiplicité

• {nb} //il faut une succession de 'nb' caractères quelconques

\. {inf,sup} // une série de (vrai) '.' en quantité au moins 'min' et au plus 'sup'

[0-9]{inf,} //au moins 'inf' chiffres

abréviations : ?={0,1}, +={1,} et *={0,}

Modèle de chaîne :

- succession de caractères et classes de caractères
- multiplicité de répétition
- groupement de sous-série entre parenthèses

[^]\-?[0-9]⁺^{\$}
[^](.+)@(.+)\.(.+) ^{\$}

Travailler avec les modèles de chaîne

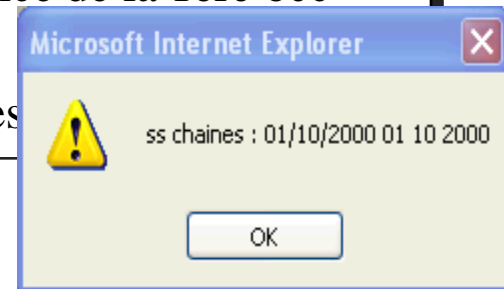
EXPLOITATION

- ☞ `ch2 = ch.replace(model, ch1)` //remplace le modèle *model* par *ch1* dans *ch*
- ☞ `ind = ch.search(model)` //-1 si aucune correspondance ou l'indice de la 1ère occ
- ☞ `tab= ch.match (model)` //tableau des sous-chaînes du modèle
//-> ss chaînes données par parenthèses

```
<html><body onLoad="
  model = /_at_/g;
  ch='jmi_at_jmi.fr';
  alert(ch.replace(model,'@'));">
</body>
</html>
```



```
<html><body onLoad="
  var tabch=new Array();
  var ch='date : 01/10/2000';
  model = /(\d{2})\(\d{2})\(\d{4})$/;
  tabch=ch.match(model);
  alert('ss chaines : ' + tabch[0] + ' ' +
    tabch[1]+ ' ' + tabch[2]+ ' ' + tabch[3]);
"></body></html>
```



Annexe : Gestion des fenêtres et du navigateur



Méthodes d'ouverture de fenêtres filles

fen = **window.open** ("URL_document", "nom_fenetre|_top|_blank", "paramètres")

- Ouverture d'une nouvelle fenêtre de nom symbolique *nom_fenetre* (ou sans nom avec *_blank*, *_top*)
- avec une liste de paramètres d'affichage (séparateur ",")
- et accède au document d'adresse URL.

Ex pour 'paramètres' : *resizable, copyhistory, width=valL, height=valH, status=yes*

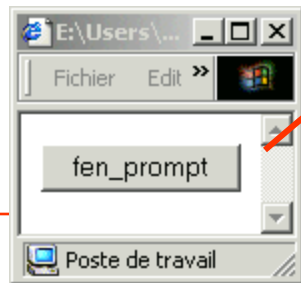
fen.focus() //met au premier plan la fenêtre de nom 'fen'

fen.close() //ferme la fenêtre

idée : La référence fen peut être stockée dans une variable globale

Ouverture de fenêtres standard

```
fen.alert("message")           //fenêtre d'information avec le btn OK, fille de 'fen'
repons= fen.confirm("message") //fenêtre de confirmation (OK, Cancel)
repons= fen.prompt("message", "message_default") //fenêtre de saisie avec (OK, Clear Cancel)
//et une chaine par défaut
```



```
<html>
<body> <center>
<input type="button" value="fen_prompt" onClick="prompt('Est-ce ok ?', 'message_default')">
</center> </body> </html>           //idem window.prompt(...)
```

Propriétés d'un objet fenêtre

fen

screen	//information sur le navigateur
document	//l'objet document de la fenêtre
status	//barre d'état (partie basse de la fenêtre)
location	// adresse URL courante de la fenetre courante

//affichage dans la barre d'état
window.status="une façon d'offrir une aide contextuelle";

fen.location=URL //charge dans 'fen' le document dont l'adresse est "URL"
fen.location.reload() //force le chargement du document
fen.replace(URL) //chargement à nouveau ou remplacement

Information sur l'écran d'affichage d'une fenêtre

fen.screen

—	☞ height	//hauteur et largeur de l'écran en pixels
—	width	
—	☞ availHeight	//Nb de pixels disponibles verticalement et horizontalement
—	availWidth	//après retrait des objets systèmes : barre de tâches, ...
—	☞ pixelDepth	//Nb de bits pour coder la couleur d'un pixel
—	☞ colorDepth	//Nb de couleurs disponibles

```
alert ('largeur : ' + window.screen.height);
```

Propriétés d'un objet document

fen.document

- 👉 **location** URL du document (utile pour recharger un autre document)
- 👉 **title** titre du document
- 👉 **lastModified** date de dernière modification

- 👉 **bgColor et fgColor** couleurs (fond et texte) sous forme d'un triplet RGB
- 👉 **vlinkColor** couleurs des liens (ici celles des liens visités)

```
window.document.location = 'URL' //charge la page d'adresse URL  
dans la fenêtre courante
```

Propriétés permettant les accès entre fenêtres

Windows //liste des fenêtres du navigateur

window //représentation la fenêtre active dans le navigateur

fen //Navigation autour de la fenêtre identifiée 'fen' (mêmes propriétés avec 'window')

- **parent** //fenêtre parent (frameset)
- **frames** // liste des frames (fenêtres filles)
- **self** // la fenêtre courante elle même (si ambiguïté)
- **history** // historique des fenêtres ouverts dans le navigateur

//navigation : *history.back()*, *go(position / ss chaine)*, *forward()*

fen.opener //donne la référence de la fenêtre génitrice (cas d'un open)

```
if (window.frames!=null) { //affichage des noms des cadres de la fenêtre
    for (i=0; i<window.frames.length; i++)
        window.alert("les noms des fenetres filles "+i+" sont : "+window.frames[i].name);}
```

```
window.history.back() //équivalent de précédent dans le navigateur
```

Méthodes de gestion dynamique du document d'une fenêtre

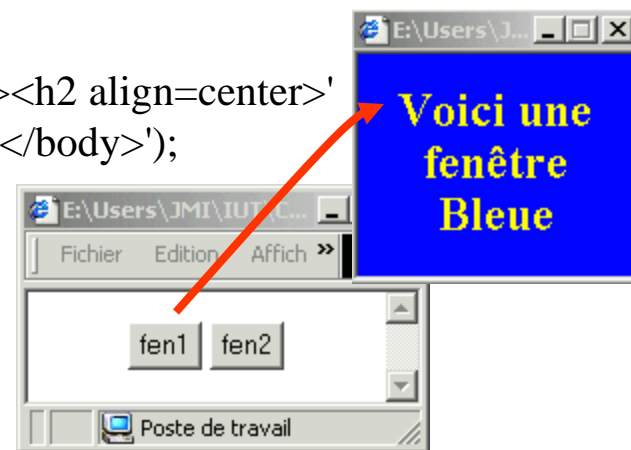
- ☞ **fen.document.open("type MIME")**
 // ouvre un buffer d'écriture de document dans la fenêtre
 // pour un type MIME donné.
- ☞ **fen.document.write(ch)** et
 fen.document.writeln(ch) écriture
- ☞ **fen.stop()** *//simule l'arrêt de chargement*
- ☞ **fen.document.clear()** efface le contenu d'une fenêtre
- ☞ **fen.document.close()** ferme le buffer

Exemple d'ouverture de fenêtre et document

```
function fen_vol (mes,coul,larg,haut)
{
  //Netscape : screenX=200,screenY=150 au lieu de left et top
  w=window.open ("", 'Exemple', "outerWidth="+larg+", outerHeight="+haut+
',left=200,top=150");
  w.document.open();
  w.document.write('<body bgcolor='+coul+' text=yellow><h2 align=center>'+
mes+'</h2></body>');

  w.document.close();
  w.setFocus(); //pour que la fenêtre apparaisse dessus
}
</script>
```

```
<body> <center>
<input type="button" value="fen1" onClick="fen_vol ('Voici une fenêtre bleue','blue',200,100);">
<input type="button" value="fen2" onClick="fen_vol ('Voici une fenêtre Rouge','red',100,50);">
</center> </body>
```



Annexe : Accès au navigateur

- objet navigator -

Attributs

☞ appName	//nom du navigateur : Netscape, IE, ...
☞ appVersion	//version du navigateur
☞ language	//langage sur 2 caractères : 'en', 'fr', ..
☞ Platform	//exple 'win32', Linux
☞ userAgent	//chaîne de caractères de reconnaissance envoyée au serveur par http exple 'Mozilla/4.01 [fr] (Win95:I)'
☞ plugins	//tableau associatif des plugins installé sur le navigateur <u>name</u> -> nomPlugin //propriétés définies pour chaque elt du tableau <u>description</u> -> description du plugin
☞ mimeTypees	//tableau associatif des types MIME reconnus <u>type</u> -> nom du type MIME <u>description</u> -> sa description <u>enabledPlugin</u> -> référence vers le plugin qui gère ce type <u>suffixes</u> -> collections des suffixes de fichiers gérés

Exemple d'utilisation de l'objet navigator

Test du navigateur

```
var question= 'pour cette page, Netscape est conseil&ecute; ';  
if ((navigator.appName != 'Netscape') && !confirm(question))  
    window.history.back;  
else  
    document.write ('on continue');
```

Test de plugin installé

```
if (navigator.plugins['Schockwave'])  
    document.write(' <embed src='anim.dir');  
else document.write ('svp, charger Schockwave');
```

Test de type MIME

```
if (navigator.mimeTypes['video/msvideo'])  
    document.write(' <A Href=clip.avi>  
    voir le clip de mes vacances </A>');  
else document.write ('svp, definir le type 'video/msvideo');
```

Annexe : gestion des liens

``

Propriétés

👉 hash	INTERNET_LOG
👉 pathname	/iut/stage.html
👉 port	2001
👉 hostname	up5iut4
👉 host	up5iut4:2001
👉 protocol	http
👉 href	http://up5iut4:2001/iut/stage.html#INTERNET_LOG
👉 search	chps1=v1& chps=v2

```
var port= document.links[1].port;  
var param=window.location.search;
```


Annexe : gestion du formatage HTML

Exemples

link --> "Lelien".link(URL) donne : Lelien

anchor --> idem avec :

small ou **big**

fixed

fontsize(taille1..7) --> ch.fontsize(2) donne ch

fontcolor(indice_couleur)

bold -> ch.bold.italics donne <I>ch</I>

italics

☒ Possibilité de formater les chaînes en interne

```
function affiche_titre(niveau,titre) {  
    document.write("<H" +level+ ">" +titre+ "</H" +level+ ">" + "<HR> <P>"); }
```

Annexe : gestion de la temporisation

Les temporisations sont rattachées aux fenêtres
(exple référence fen ; défaut : l'objet window)

Méthodes :

- 👉 `timer=fen.setTimeout(expression,msec)` //expression JS évaluée au bout de 'msec' millisecondes
- 👉 `timer=fen. setInterval (expression,msec)` //expression JS évaluée périodiquement
- 👉 `timer=fen. setTimeout(ref_fct,arg1, ...arg)` // cas des fcst paramétrées
- 👉 `timer=fen. setInterval (expressionJS,'msec')`
- 👉 `clearTimeout(timer)` //arrêt impératif du comptage pour la variable 'timer'.
- 👉 `clearInterval(timer)`

```
Timer= setTimeout ("echec()", 60000); //lance la fonction echec() au bout de 60s
```

Exemple de jeu temporisé

"trouver l'intru"

Style de programmation : événementielle

Page html

Quel nom faut'il écarter ?

Oeil ✓ Oreille ✓ Joue ✓

Nez ✓ Lèvre ✓ Sourcil ✓

ⓘ JavaScript Alert
perdu !!

ok

0- Affiche
HTML

2- evt
OnClick

3- script
Verif
+alert()

Client

interpréteur
HTML&Script

Code de l'exemple

jeu temporisé "trouver l'intru"

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-15" />
<title>jeu</title>
<script type="text/javascript" src="jeu.js">
</script>
</head>

<body onload="initJeu()">
    <div id='QUEST'>
        </div>        <!--bloc d'affichage dynamique-->
</body> </html>
```

Code de l'exemple (suite)

*fonctions
à rajouter dans jeu.js*

```
var timer; //variable globale référençant le temporisateur
var temps_imparti = 60000; //temps imparti pour donner la réponse (10s)
var q; //référence à une fenêtre d'affichage
//var eaigu = '\351';

function initJeu() {
    q = document.getElementById('QUEST'); //affichage de la question dans QUEST
    var msgInit = "Vous avez une minute pour r\351pondre";
    msgInit += "apr\350s avoir d\351marrer le test <br/>";
    msgInit += "<a onclick='question(temps_imparti); return false'>
                d\351marrer</a>";
    q.innerHTML = msgInit;
}
```

Code de l'exemple (suite)

*fonctions
à rajouter dans jeu.js*

```
function reponse(rep) {  
    clearTimeout(timer);    //stoppe le compte à rebours  
    if (rep != 'r4') {//Désolé, la réponse est NEZ !  
        alert ('D\351sol\351, la r\351ponse est NEZ. \n Les autres vont par deux');  
    } else {  
        alert ('Bonne r\351ponse !'); //Bonne réponse !  
    }  
    initJeu();  
}  
  
function echec () {  
    alert('d\351sol\351');    //désolé !  
    initJeu();  
}
```

Code de l'exemple (suite)

*fonctions
à ajouter dans jeu.js*

```
function question () {  
    var quest = "<h3 align='center'> QUESTION : </h3><hr>";  
    quest += "Quel nom faut il \351carter ? ";  
    quest += "<ul style='list-style-type:none'>";  
    quest += "<li><input type='radio' name='rd' value='r1'  
                onclick='reponse(this.value)' /> Oeil</li>";  
    quest += "<li><input type='radio' name='rd' value='r4'  
                onclick='reponse(this.value)' /> Nez</li>";  
    quest += "<li><input type='radio' name='rd' vvalue='r6'  
                onclick='reponse(this.value)' /> Sourcil</li>";  
    quest += "</ul>";  
    q.innerHTML = quest;  
    timer= setTimeout('echec()', temps_imparti);           //lancer echec() à échéance du  
    tps imparti  
}
```