

Java script

- Insertion d'instructions de programmation directement dans le code des pages HTML
- Exécution de code sur **le poste client** pour
 - améliorer l'interactivité (temps de réponse plus court)
 - améliorer les débits sur le réseau (éviter des envois erronés au serveur)
 - proposer des pages un peu plus dynamiques
 - décharger le serveur de tâches que l'on peut faire sur le poste client
- **Exemples**
 - test d'un formulaire avant envoi
 - calcul effectué puis affiché
 - animation, modification du DOM, etc

Le langage JavaScript

- Créé à l'origine par Netscape (Netscape 2.0)
- Conçu pour traiter localement des événements provoqués par le client
 - déplacement du pointeur de souris ou clic de souris
 - soumission d'un formulaire, ...
- JavaScript est un langage
 - interprété (le code du script est analysé et exécuté au fur et à mesure par l'interprète, partie intégrante du navigateur) - problème en cas d'erreurs !
 - à base d'objets
 - multi-plateforme

Le langage JavaScript

- JavaScript permet
 - de programmer des actions en fonction d'événements
 - si la zone de saisie contient 8 caractères alors passer le focus au champ suivant
 - d'effectuer des calculs (sans recours au serveur)
 - lire la valeur saisie, la multiplier par 3,14 et afficher le résultat

Application

- Petites applications simples (calculatrice, outils de conversions, jeu, ...)
- Aspects graphiques de l'interface (gestion de fenêtres, animations, modification du style d'un objet)
- Test de validité des données sur les éléments de l'interface de saisie. Une partie de validation triviale peut être déléguée à l'HTML 5 mais pas les parties complexes comme vérifier que le mot de passe saisi dans un input est le même que celui de la zone « confirmation mot de passe ».
- Modifier le DOM : par exemple sur la déclaration des impôts on saisi le nombre d'enfant et dès qu'on perd le focus il y a ajout de tout un formulaire au milieu de la page pour saisir leur date de naissance sans avoir à recharger toute la page.

Nom

Prénom

Nb Enfant

Adresse

si on saisi 2 en nombre d'enfants
et on donne le focus à profession
nous aurons deux zone de saisi qui
apparaîtront **au milieu** du formulaire
et recharger toute la page

Nom

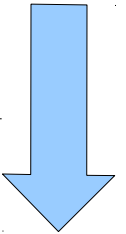
Prénom

Nb Enfant

Date naissance enfant 1

Date naissance enfant 2

Adresse



Démo impot.html dans répertoire du cours

Le noyau JavaScript

- Au niveau du langage, on distingue
 - le noyau JavaScript (cœur du langage) comportant des objets prédéfinis, des opérateurs, des structures, ...
 - un ensemble d'objets associés au navigateur
 - fenêtres,
 - documents,
 - images, ...
- Il existe aussi la possibilité d'exécuter du code JavaScript sur le serveur (communications avec une BD...) mais l'usage est plus restreint.

JavaScript // JAVA

- JavaScript

- interprété
- à base d'objets prédéfinis (pas d'héritage)
- code intégré dans HTML (visible)
- typage faible
- n'existe pas en dehors du Web
- *débogage* difficile

- Java

- compilé
- orientés objets (définition de classes, héritage)
- code dans applets (non visible)
- typage fort
- langage à part entière
- environnement de développement

Insertion de code JavaScript

- On utilise la balise `<script>...</script>` sans `type="text/javascript"` pour Html 5
- Les fonctions sont déclarées dans l'en-tête entre `<head>` et `</head>`
- Les fonctions sont appelées dans `<body>...</body>`

```
<!-- index.html -->
<html>
  <head>
    <script type="text/javascript">
      function essai() {
        alert('BONJOUR TOUT LE MONDE');
      }
    </script>
  </head>
  <body>
    <form>
      <input type="button" value="ok" onClick="essai()">
    </form>
  </body> </html>
```

`alert()` est une fonction prédéfinie qui affiche une boîte de dialogue

`prompt()` idem avec un champ texte à remplir (c'est une lecture)

8

Gestion des évènements : L'appel à `essai()` est déclenché par un clic (événement `onClick`) sur un bouton créé par JS (button):

Insertion à partir d'un fichier externe

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

<body>
  <script type="text/javascript">
    document.write('<em>bien venu sur ce site </em>');
  </script>

  <p><a href="javascript:;" onClick="bonjour();"> cliquez ici</a>
    <br> ou passez la souris sur

    <a href="javascript :;" onMouseOver="bonjour();">ce lien</a>
  </p>
</body>
```

← Fonctions Js dans fichier externe

← Code js de génération de code html

← Appel des fonctions Js se trouvant dans le fichier externe

```
// fichier.js
function fin() {
    alert('bonjour') ;
}
```

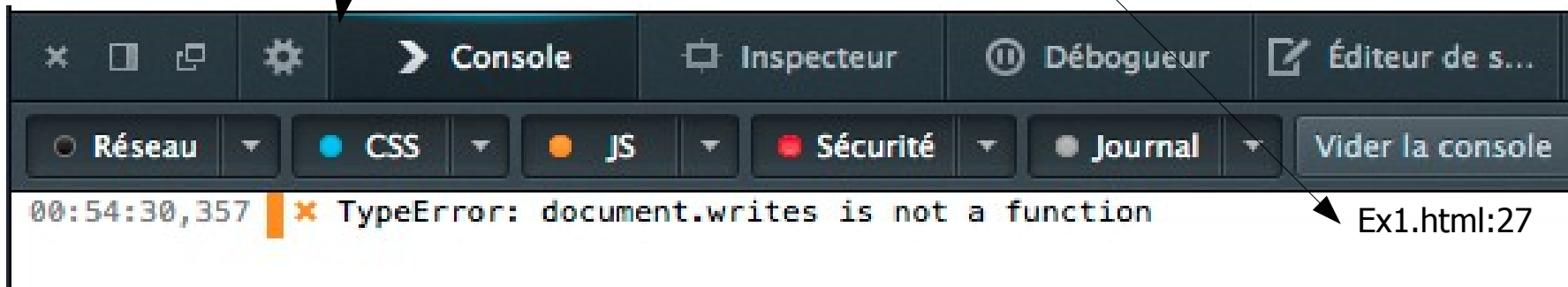
- On préfère ' à " pour ne pas confondre Html et Js dans la manipulation des instructions telles alert, prompt et write
- **Exemple** : `document.write(' <input type="text" />') ;`

`document.write(X)` insère X dans le flux Html. Ce dernier sera **interprété** par le navigateur.

Concaténation via « + » comme en JAVA.

- Attention au fonctionnement particulier de `document.write` en **fin de flux** : c'est-à-dire une fois la page entièrement chargée. Nous y reviendrons dans la section « événement ».

- On utilise `\[caractère]` pour forcer l'interpréteur à ne pas interpréter en tant que partie code le `[caractère]` :
- **Exemple** : `alert(' l'avion ');` \Rightarrow affiche dans une boîte de dialogue le mot : `l'avion`
- Déboguer via la console de l'inspecteur d'élément.
- Par exemple on a écrit « `document.writes` » au lieu de « `document.write` »
Le numéro de la ligne contenant l'erreur se met complètement à droite.



Editeur JS

- Plusieurs outils existent :
- TextMate → mac
- NotePad++ → windows

Auto indentation : complément → plug in manager

installer Text FX

TextFx → TextFx edit → reindent C++

- Emuler NotePad++ sur Linux via wine.
- Emacs, Eclipse, Aptana,... → Linux

La balise <noscript>

- Sert à informer que le navigateur ne supporte pas le JS de la page chargée
- Bonne utilisation de la balise <script> :

<script>

<! --

code javascript

-->

</script>

<noscript>

attention, ce document contient du code javascript non interprété
par votre navigateur...

</noscript>

Le langage JavaScript

- Des variables **faiblement typées**
- Des opérateurs et instructions (**ceux du langage C**)
- Des méthodes
 - **globales** (associées à tous les objets)
 - **fonctions** définies par l'utilisateur
- Des objets
 - **prédéfinis** (String, Date, Math, ...)
 - **liés à l'environnement** (window, document, ...)
- Commentaires comme en langage C `//` ou `/* xxxx */`

- Séparateur d'instructions : **;**

Opérateurs JavaScript

- Ceux du langage C

- arithmétiques : + - * / %

var x=5,y=3 ;

x/y sera à 1,666... bien que x et y soient entiers. On passe par `parseInt(x/y)` pour récupérer la partie entière d'une division.

par contre $x\%y$ sera bien à 2

- l++ ;

- logiques : && (ET) | | (OU) ! (NON)

- comparaisons: == != <= >= < >

- concaténation de chaînes : +

ATTENTION : L'opérateur + est l'addition ou la concaténation

Affectations

- affectation simple

nom=valeur;

- affectation conditionnelle

var = (condition) ? exp_alors : exp_sinon;

X = (a > b) ? "plus" : "moins";

X aura la valeur « plus » si a>b et « moins » sinon

- affectation avec opération : +=, -=, *= ...

X +=3; // équivaut à X=X+3;

- ATTENTION : distinguer l'affectation (=) et la comparaison (==)

Variables JavaScript

JavaScript est sensible à la casse : X différent de x

- Déclaration de variables
 - optionnelle mais fortement conseillée
 - avec l'instruction `var` : `var x ; var y=1 ;`
 - le type n'est pas précisé lors de la déclaration
 - initialisation possible lors de la déclaration sinon valeur `undefined`
 - On peut donc tester `if (x==undefined) {...}`
- Notion de variables locales et globales
 - locales à une fonction
 - globales au document HTML

Variables JavaScript

En JS une variables peut être utilisée dans deux scripts :

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

```
var x=1 ;
```

```
</script>
```

<h1> affichage de la valeur de x </h1>

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

```
x++ ;
```

```
alert('x='+x) ;
```

```
</script>
```

On aura l'affichage : **x=2**. Cette variable sera également reconnue dans un fichier JS que ce fichier Html appel via :

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

Variables et types

- Le **typage** a lieu lors de l'initialisation ou d'une affectation
- Le type d'une variable **peut changer** si on lui affecte une valeur d'un autre type
- `alert(typeof nom-var)` : affiche le type de la variable nom-var
- Les types de données simples
 - Nombre (**number**)
 - Booléen (**boolean**)
 - Chaîne de caractères (**string**)
'chaîne' ou "chaîne"
Les codes `\t` (tabulation), `\r` (retour chariot), etc sont reconnus

Conversions de type

- JavaScript fait des conversions **implicites** selon les besoins
- **Exemples**
 - `N=12; // N numérique`
 - `T="34"; // T chaîne de caractères`
 - `X=N+T; // X est la chaîne de caractères "1234"`
 - Il existe des types particuliers : **undefined, objet, function**
 - Il existe des valeurs particulières : Infinity, -Infinity, NaN

- **parseFloat(ch)** et **parseInt(ch,base)** pour les conversions
string->r  el ou string->entier
- **isNaN(expr)** pour tester si une expression n'est pas num  rique.
Attention **isNaN** sur une chaine vide renvoie **false** contre toute attente !!!
- **eval(ch)** si ch est une expression, eval(ch)   value ch. Si ch est une s  ries
d'instruction JS eval(ch)   value et **ex  cute** une expression JavaScript contenu dans
la cha  ne de caract  re ch

eval(" x=10 ; y=20 ; document.write(x*y) ");

document.write("
 " + **eval**(x+17));

200

27

Instructions classiques

- Instructions de branchement

```
if (condition) { instructions; } [ else { instructions; } ]
```

- Boucles

```
for (i=1 ; i<N ; i++) { instructions; }
```

```
while (condition) { instructions; }
```

```
do { instructions; } while (condition);
```

```
for (p in objet_ou_tableau ) { instructions; } :
```

« p » parcourt toutes les propriétés de l'objet ou tous les index d'un tableau.

2

```
for ( i in T ) { x=x+T[i];}
```

 somme toutes les cases d'un tableau T

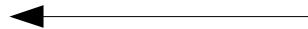
2

Instructions classiques

- Le bloc switch

```
switch (variable) {
```

```
case 'valeur 1':
```



chaîne de caractère

```
Code à exécuter si "variable == 'valeur 1'";
```

```
break;
```

```
case 'valeur 2':
```

```
Code à exécuter si "variable == 'valeur 2'";
```

```
break;
```

```
case 12:
```



entier

```
Code à exécuter si "variable == 12";
```

```
break;
```

```
default:
```

```
Code à exécuter si tous les autres ont échoué;
```

```
break;
```

```
}
```

Les fonctions

```
function nom_f (arg1, ..., argN) {  
  
instruction1;  
  
    ...  
    instructionN;  
  
    return valeur;  
}
```

– arguments non typés

Appel de fonction depuis le script

On peut appeler une fonction lors d'un évènement : click, survol etc.

On peut aussi appeler une fonction directement entre balise <script> et </script>

```
<html>

  <head>
    <script type="text/javascript">
      function essai() {
        alert('BONJOUR TOUT LE MONDE');
      }
    </script>
  </head>
  <body>
    <script>
      essai() ;
    </script> ;
  </body>
</html>
```

Les objets

- Pas de classe mais des pseudo-classes
 - pas de sous-classe
 - pas d'héritage
 - uniquement des créations d'objets et la possibilité de définir des propriétés
 - syntaxe largement inspirée de la programmation objet

L'objet String

Déclaration via « **var** » : `var T = "Bonjour";`

`T.length` --> 7 // le nombre de caractère

`T.indexOf("o")` --> 1 // première occur

`T.lastIndexOf("o")` --> 4 // dernière occur

`T.charAt(3)` --> j // lettre correspondant à T[3]

`T.substring(3,7)` --> jour // sous chaîne index début -> fin (fin non inclu)

`T.toUpperCase()` --> BONJOUR // mettre en MAJ

attention au U et C majuscule



- Mais aussi :
 - `lastIndexOf(chaîne)`,
 - `toLowerCase()`,
 - etc

L'objet Date

- Déclaration via `var maintenant=new Date();`

Les méthodes :

`maintenant.getDate()` // le numéro de jour

`maintenant.getMonth()` // le numéro de mois 0 pour janv

`maintenant.getFullYear()`

`maintenant.getHours()`

`maintenant.getMinutes()`

`maintenant.getSeconds()`

Et encore plein d'autres méthodes à découvrir

(Cf. Slide 54 comparaison de deux dates saisies)

L'objet Math

- A ne pas déclarer : utiliser directement `Math.méthodes`
- Méthodes : fonctions usuelles
 - trigonométriques : `cos`, `sin`, `atan`, `asin`,...
 - `abs` (valeur absolue)
 - `ceil` (entier sup), `floor` (entier inf), `round` (entier + proche)
 - `exp`, `log`, `sqrt`, `pow(x,a)`
 - `max (a,b)`, `min (a,b)`
 - `random()` --> $0 < r < 1$

Les tableaux

- Construire un tableau sans préciser le contenu
`var Tab = new Array();`
- Construire un tableau en précisant la taille
`var Tab = new Array(3);`
- Initialisation du tableau lors de sa création
`var Tab = new Array(t1, ..., tN);`
 - > **les indices varient de 0 à N-1**
 - > **les t_i peuvent être de types différents**
- propriété *length* : taille du tableau
`Tab.length = N;`
- La taille du tableau est dynamique

Les tableaux à deux dimensions

On les crée par partie : colonne par colonne car

certaines versions de JS ne supportent pas la déclaration directe

		X
X	O	
O	X	

- ```
var col0=new Array;
col0[0]=" "; col0[1]="X" ; col0[2]="O";
```

```
var col1=new Array;
col1[0]=" "; col1[1]="O"; col1[2]="X";
```

```
var col2=new Array;
col2[0]="X"; col2[1]=" "; col2[2]=" ";
```

Ensuite le tableau morpion :

```
var morpion=new Array;
morpion[0]=col0; morpion[1]=col1; morpion[2]=col2;
```

- Autres méthodes :

- **join** renvoi une chaine contenant tous les éléments séparés par un séparateur passé en argument.

Exemple : alert(t.**join**(',')) ;



- **reverse** transpose le tableau (1<->N 2<-> N-1...)
- **sort** trie les éléments dans l'ordre lexicographique ou selon la relation d'ordre passée en argument (fonction de comparaison fournie par l'utilisateur)



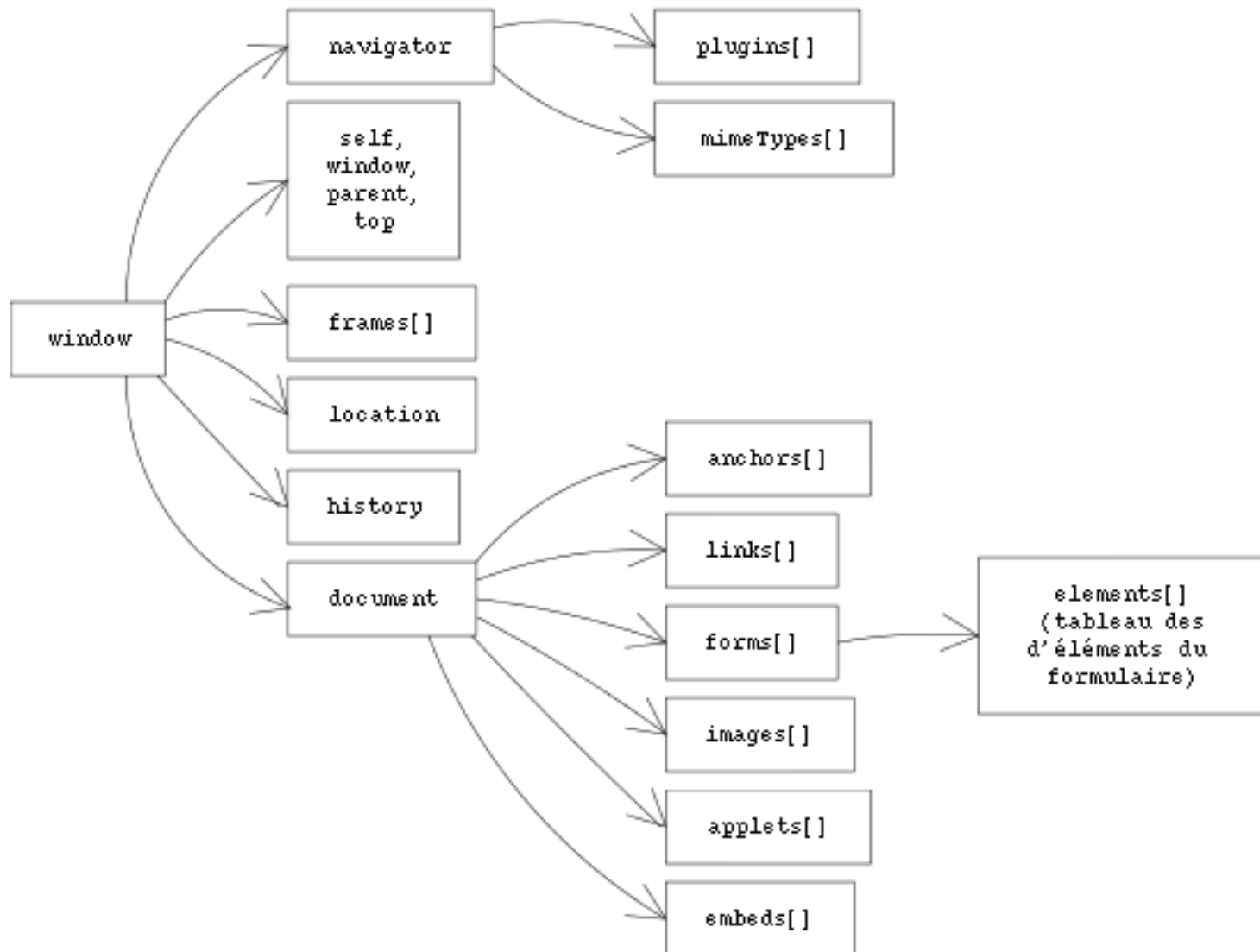
# Le Tri avec la fonction SORT

- On peut définir notre propre notion de comparaison en définissant la fonction `compare(a,b)` et en appelant ensuite la fonction `sort(compare)`
  - `function compare(a,b)`
    - { if (eval(a)<eval(b)) return -1;
    - else if (eval(a)>eval(b)) return 1; else
    - return 0;
    - }
- `T.sort(compare);`

# Les objets prédéfinis

- Les objets **instanciés automatiquement** lors du démarrage du *browser*
  - permettent d'accéder à des informations concernant le navigateur client, les documents HTML affichés, l'écran de la machine
  - la classe **Navigator**
    - une seule instance -> objet *navigator*
    - infos sur nom, version, plug-ins installés,...
  - la classe **Window**
    - une instance par fenêtre et *frame* du document HTML
    - accès à tous les objets créés par des balises HTML
  - la classe **Screen**
    - une seule instance -> objet *screen*
    - infos sur largeur et hauteur en pixels, nombre de couleurs disponibles, ...

# Hiérarchie des objets du navigateur



# L'objet *navigator*

- Très utile pour régler les conflits entre navigateurs. Voir exemple page suivante
- **navigator.appName** == *'Netscape'*  
nom du browser permet de différencier les navigateurs
- **navigator.appVersion** == *'4.7 [fr] (Win95; I)'* informations sur la plate-forme d'exécution
- **navigator.language** == *'fr'* (2 caractères)
- **navigator.platform** == *'Win32'* type de machine
- **navigator.appCodeName** == *'Mozilla'*  
nom de code du navigateur

# Les problèmes d'incompatibilité

L'objet

l'attribut

```
// Vérifie que le navigateur soit au minimum IE4 ou Netscape4
function verifieNav() {
 var oldNav = true;
 navName = navigator.appName;
 navVer = parseInt(navigator.appVersion);
 if (navName == "Netscape" && navVer >= 4)
 oldNav = false;
 else if (navName == "Microsoft Internet Explorer" && navVer >= 4)
 oldNav = false;
 if (oldNav)
 alert('Passez à une version plus récente de votre navigateur !');
}
```

```
if (navigator.appName == 'Netscape') {
 // Code à exécuter avec le navigateur Netscape
} else {
 // Code à exécuter avec les autres navigateurs
}
```

# L'objet *screen*

- **screen.height** hauteur de l'écran en pixels
- **screen.width** largeur de l'écran en pixels
- **screen.availHeight** nombre de pixels disponibles verticalement (sans les barres de tâches, ...)
- **screen.availWidth** idem horizontalement

# La classe Window

- Un objet `window` pour chaque fenêtre ouverte par le navigateur
  - `window.history` qui contient un tableau des URL déjà visitées dans la zone (historique)  
**Attention** : il n'est pas accessible en lecture pour des raisons de sécurité. On utilise les méthodes de history :
    - `back()` (Retourne à la page précédente)
    - `forward()` (Retourne à la page suivante)
    - `go(-x)` (Redirige vers x page en arrière de l'historique)
  - `window.location` qui contient les caractéristiques de l'URL de la zone. Sa propriété `href` contient l'URL complète. La modification de sa valeur permet de faire une redirection.
  - `window.document` qui contient les caractéristiques et tous les objets de la zone

## La méthode : `window.open()`

`nom = window.open(URL,nom_fenetre,paramètres,hist);`

Ouvre une nouvelle fenêtre avec :

- **URL** : représente sous forme de chaîne de caractères l'URL du document à charger dans la nouvelle fenêtre; cette chaîne peut être vide
- **nom\_fenetre** est une chaîne de caractères permettant d'attribuer un nom.  
A ne pas confondre avec « **nom** » qui sert de base d'appel aux méthodes de cette nouvelle fenêtre, par exemple : `nom.close()`,...
- **Paramètres** : suite d'éléments du type *propriété=valeur* séparés par `' '`  
`width=200, height=100`
- **hist** est une variable booléenne (`true` ou `false`) qui permet de déterminer si l'URL chargée doit ou non créer une nouvelle entrée dans l'historique.



## Les propriétés associées à l'objet window

**left/top** : position horizontale/verticale du coin supérieur gauche

**height/width** : hauteur/largeur totale en pixels

**innerHeight/innerWidth** : hauteur/largeur de la partie utile

**location** : égal à 1 ou 0 pour afficher ou non la ligne de saisie des URLs

**toolbar** : égal à 1 ou 0 pour afficher ou non la barre d'outils

- La méthode **close()**

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

```
w=window.open();
```

```
w.document.open();
```

```
w.document.write("Hello World!");
```

```
w.document.close();
```

```
</script>
```

← Fermeture du document et non pas de la fenêtre

- La méthode **alert()**

```
w.alert(message);
```

- ouvre une boîte d'alerte dans la fenetre « w » où s'affiche la chaine message avec un bouton OK qui permet de fermer la fenêtre

- La méthode **focus()** rend la zone active , par exemple **w.focus()**

- La méthode **blur()** rend la zone non active

- La méthode **prompt()** chaine = **w.prompt**(message, valeur\_defaut);

- ouvre une fenêtre composée d'un message et d'un champ de saisie et retourne la valeur saisie

- La méthode **confirm()**

```
bool = w.confirm(message);
```

- ouvre une fenêtre avec message, boutons OK et Annuler et retourne *true* (OK) ou *false* (Annuler)

# La propriété *document* de window

- Les principales propriétés
  - titre du document : **document.title**
  - couleur du texte : **document.fgColor**
  - couleur du fond : **document.bgColor**
  - couleur de liens : **document.linkColor**
  - couleur de liens visités : **document.vlinkColor**

# La propriété *document* de window

- Les principales propriétés
  - couleur de liens activés : `document.alinkColor`
  - adresse du document : `document.URL`  
**Strictement équivalent à `location`.**
  - URL de la page d'où vient le visiteur : `document.referrer`
  - date de dernière modification : `document.lastModified`
  - ...

# La propriété *document* de window

- **forms[]** tableau des formulaires <form>
- **elements[]** tableau des composants du formulaire <input..>, <select...>, ...
- Ainsi **document.forms[0].elements[0]** représente le premier élément du premier formulaire de la page.
- **anchors[]** tableau des liens internes <a name=...>

# La propriété *document* de window

- **applets[]** tableau des applets <applet code=...>
- **images[]** tableau des images <img src=...>
- **links[]** tableau des liens (hypertextes et images cliquables) <a href =...>

# Accès aux éléments d'un formulaire via document

Accès à un objet du formulaire : on passe par « forms » et « elements » en utilisant soit des indexes soit les noms utilisés dans les propriétés « name »

Dans l'exemple ci dessous on affecte 3 à la zone de texte

```
<form id="mon_form">
 <p><input type="text" name="t1" value="z1"></p>

</form>
```

solution 1 : <script> document.forms[0].elements[0].value = 3; </script>

solution 2 : <script>  
 document.forms['mon\_form'].elements['t1'].value = 3;  
</script>

Notons le .value pour accéder à la valeur d'une zone de texte.

# Accès aux éléments d'un formulaire via document

Les formulaires sont indexés (en commençant par 0) : Ainsi le premier formulaire de la page est : `window.document.forms[0]`

Le nombre d'éléments du premier formulaire est donné par `window.document.forms[0].elements.length`

De la même manière le « i » ème élément du premier formulaire est accessible via `window.document.forms[0].elements[i]`



# Accès aux éléments d'un formulaire via document

Les attributs comme name, value, type;...etc de chaque éléments sont accessibles :

`window.document.forms[0].elements[i].name` → le nom de l'élément de rang 'i'

`window.document.forms[0].elements[i].value` → la valeur de l'élément de rang 'i' par exemple le contenu d'une zone de saisie. Le typage est respecté : si on saisie un nombre entier il sera typé entier et on peut le mettre dans une boucle.

`window.document.forms[0].elements[i].type` → le type par exemple "text" pour une zone de texte.

On peut également appeler des fonctions comme donner le focus :

`document.forms.[0].elements[i].focus()`

On peut construire une boucle de « for » avec « i » allant de 0 à `window.document.forms[0].elements.length` pour accéder à tous les éléments du formulaire via `window.document.forms[0].elements[i]`.

# Passage par Name

On peut accéder aux éléments d'un formulaire également par leur noms au lieu de leur index : c'est-à-dire `elements['zone']` au lieu de `elements[0]` si l'élément contient un attribut html `name="zone"`

Attention aux formulaires qui ne contiennent pas d'attribut name mais un Id. On accède également par `forms['formu']` pour un formulaire avec `id="formu"`.

Chaque élément est donc joignable via  
`window.document.forms[id_formu].elements[nom_element]`

Exemple : **parcours d'un formulaire** et teste si les zones de textes sont vides.

Pour cela on utilise :

- (1) La propriété « **type** » permet savoir le type de l'élément :  
button,checkbox,image,password,radio,select-one (liste déroulante),select-multiple,submit,text,textarea...etc.
- (2) une fonction qui prend en parametre une variable qui représente un formulaire.

```
function verifier(formu){
 var returnval=true
 for (i=0; i<document.forms[formu].elements.length; i++){
 if (document.forms[formu].elements[i].type=="text"
 {
 if (document.forms[formu].elements[i].value=="")
 { alert(' élément vide') ;} }
 }
}
```

Etc...

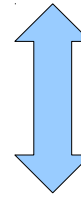
## Exemple : Comparaison de deux dates saisies via des zones de saisie (très utile)

Les dates sont saisies selon le format **YYYY-MM-JJ** pour rester cohérent avec le format DATE de MySQL que l'on utilisera plus tard.

```
function comparer(){
```

```
 var sdate1=document.forms['f'].elements['d1'].value;
 var sdate2=document.forms['f'].elements['d2'].value;
```

```
 var date1 = new Date();
 date1.setFullYear(sdate1.substr(0,4));
 date1.setMonth(sdate1.substr(5,2));
 date1.setDate(sdate1.substr(8,2)) ;
```



création d'un objet Date avec  
chargement des valeurs  
à partir de la zone de d1

```
 var date2 = new Date();
 date2.setFullYear(sdate2.substr(0,4));
 date2.setMonth(sdate2.substr(5,2));
 date2.setDate(sdate2.substr(8,2));
```

```
 if(date1>=date2) alert('d1>=d2') ; else alert('d1<d2') ;
 }
```

```
<form id='f' >
```

```
<p> date <input type="text" name="d1"/></p>
```

```
<p> date <input type="text" name="d2" /></p>
```

```
<p> <input type="button" onclick="comparer();" /> </p>
```

```
</form>
```

# Passage des éléments d'un formulaire en parametre

- si on passe des éléments de formulaire en parametre d'une fonction en le met entre ' et '

Exemple 1 : Mise en majuscule d'une zone de texte après appuie sur un bouton (exo3 série JS)

- `function` transforme(formulaire,source,destination) {  
    document.forms[formulaire].elements[destination].value=  
    (document.forms[formulaire].elements[source].value).toUpperCase();  
}
- La fonction « transforme » prend en parametre un formulaire et deux zones de texte : une source et une destination
- Ci dessous le formulaire avec l'appel de la fonction : notons les ' sur chaque parametre :

```
<form id="formu">
```

```
<input name ="min" type="text" >
```

```
<input name ="maj" type="text" >
```

```
<input type="button" value=" maj" onClick="transforme('formu','min','maj')" >
```



# Passage des éléments d'un formulaire en parametre

A l'intérieur d'une fonction il ne faut jamais utiliser de référence par nom direct aux éléments d'un formulaire passé en paramètre.

Reprenons l'exemple précédent :

```
function transforme(formulaire,source,destination) {
 document.formulaire.destination.value=
 (document.formulaire.source.value).toUpperCase();
}
```

Il y aura **une erreur** après interprétation ! Il faut passer par forms et elements (voir diapo précédent).

```
<form id="formu">
<input name ="min" type="text" >
<input name ="maj" type="text" >
<input type="button" value=" maj" onClick="transforme('formu','min','maj')" >
```

## Exemple 2

- Dessin automatique d'un tableau

Ligne :  Colonne :

(1,1) (1,2) (1,3)  
(2,1) (2,2) (2,3)  
(3,1) (3,2) (3,3)  
(4,1) (4,2) (4,3)

<form id="f">

<p> Ligne : <input name="l" type="text" size="5">  
Colonne : <input name="c" type="text" size="5">

<INPUT type="button" value="Dessiner" onClick="dessin('f','l','c')">

</form>

## Example 2

```
function dessin(formu,ligne,colonne)
{
 var x=document.forms[formu].elements[ligne].value;
 var y=document.forms[formu].elements[colonne].value;

 if (isNaN(x)||isNaN(y)) {alert('Erreur');} else
 {
 document.write('<table>');
 for (i=1;i<=x;i++)
 { document.write('<tr>');
 for (j=1;j<=y;j++)
 {document.write('<td> ('+i+', '+j+ ')</td>');
 }
 document.write('</tr>');
 }
 document.write('</table>');
 }
}
```



## Exemple 2

- Dessin automatique d'un tableau

Ligne :  Colonne :

(1,1) (1,2) (1,3)  
(2,1) (2,2) (2,3)  
(3,1) (3,2) (3,3)  
(4,1) (4,2) (4,3)

Pourquoi les combinaisons apparaissent dans la même fenêtre mais l'IHM a disparu ?

A cause du fonctionnement de `document.write` une fois le flux de la page entièrement lu : l'affichage se fait dans un `nouveau document` (même page mais nouveau document).

Attention à la syntaxe :

```
document.write('<table>
 <tr>
 <td>
 </td>
 </tr>
 ');
```

n'est pas reconnue comme étant

```
document.write('<table><tr> <td></td></tr>') ;
```

C'est même source d'erreurs lors de l'interprétation due aux sauts de lignes

# L'objet **this**

Il est fastidieux d'accéder aux éléments de formulaire par toute la chaîne `document.forms.elements`.

- Un objet JavaScript **this** permet de raccourcir ce chemin d'accès. `this` représente l'objet javascript en cours.
- Dans l'exemple on met 5 dans la valeur du champ dès que la zone a le focus.
- ```
<form id="general">  
<input type="text" name="champ1" value="valeur initiale"  
onFocus="this.value=5">  
</form>
```

L'objet **this**

L'autre avantage du **this** est la possibilité de manipuler directement les propriétés des éléments passés par un **this** sans passer par **document.forms....** :

```
function afficherCinq(X) {  
    X.value=5 ; // et non pas document.forms...  
}
```

```
<form id="general">  
<input type="text" name="champ1" value="valeur initiale"  
onFocus="afficherCinq(this)">  
</form>
```

Du fait que le X représente le « **this** » alors à l'intérieur de la fonction **afficherCinq** on accède directement à l'attribut « **value** » sans écrire **document.forms....**

Exemple : désactivation d'un bouton

Les boutons : pour désactiver un bouton on passe par la propriété

chemin.disabled=true

```
<form name="formu" ... >
```

```
<input type="button" name="bouton" onClick="this.disabled=true ;">
```

```
</form>
```

Ou bien `document.forms['formu'].elements['bouton'].disabled=true ;`

Accès aux éléments d'un formulaire via document

Les cases à cocher : pour détecter qu'une case est cochée, il faut utiliser sa propriété checked.

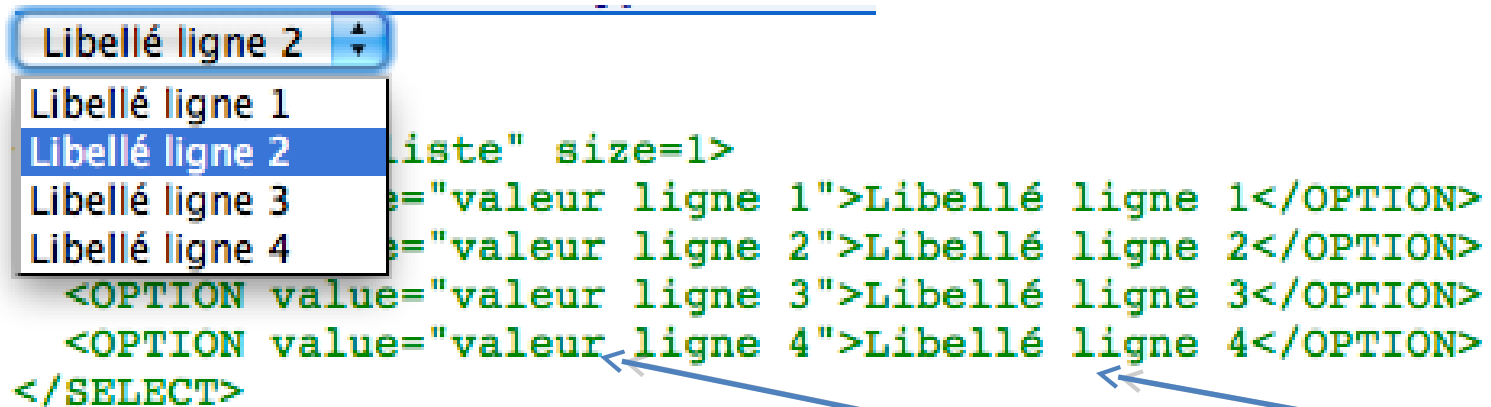
Dans le code ci dessous on a deux cases a cocher et on affiche la valeur booléenne de chaque cas lors du click sur le bouton « tester »

```
<form id="f">  
  <input type="checkbox" name="majeur">majeur  
  <input type="checkbox" name="etudiant">etudiant  
  <input type="button" value="tester" onClick="alert('majeur :'  
' + document.forms['f'].elements['majeur'].checked + 'etudiant :'  
+ document.forms['f'].elements['etudiant'].checked);">  
</form>
```

☐ Majeur ☒ Etudiant

Accès aux éléments d'un formulaire via document

Les liste déroulante : supposons que la liste est de name="liste" :



Pour récupérer l'indice la ligne sélectionnée :

(les préfixes).`liste.selectedIndex`

Pour récupérer le nombre de lignes :

(les préfixes).`liste.options.length`

Pour récupérer la valeur de la ligne sélectionnée :

(les préfixes).`liste.options[(les préfixes).liste.selectedIndex].value`

Pour récupérer le libellé de la ligne sélectionnée :

(les préfixes).`liste.options[(les préfixes).liste.selectedIndex].text`

Les événements

- Attention au fonctionnement particulier de `document.write()` en **fin de flux** : c'est-à-dire une fois la page entièrement chargée.

Si une page **a fini de charger** son flux et que l'on a un bouton avec un événement :

onclick : `document.write('chaine')`

```
<h1> Teste </h1>
<form>
<input type="button" onclick="document.write('bonjour');"/>
</form>
```

Lorsqu'on click on aura un document vierge avec uniquement
« bonjour »

Les événements

Exemples d'événements

- un lien hypertexte est sensible au clic ou au passage de la souris

```
<a href="javascript : ;" onMouseOver="alert('bonjour');">
```

cliquez ici

```
</a>
```

- un formulaire est sensible au fait d'être soumis
- une page est sensible au fait d'être chargée
- Une zone de texte est sensible à chaque caractère entré (très utile pour changer automatiquement de champ) une fois le champ rempli sans avoir à cliquer ou appuyer sur TAB

Attribut	Balises concernées	Déclenchement de l'événement
onAbort	IMG	L'utilisateur interrompt le chargement d'une image
onBlur	BODY, FRAMESET, <INPUT txt,pwd>, <TEXTAREA>	Un élément n'est plus actif (perd le focus)
onChange	<INPUT txt,pwd>, <TEXTAREA>	L'utilisateur à changer la valeur d'un champ de saisie
onClick	<A>, <AREA>, <INPUT others>	L'utilisateur clique sur un objet
onDbClick	<A>, <AREA>, <INPUT others>	L'utilisateur double-clique sur un objet
onError	IMG ou window.onerror	Une erreur se produit lors du chargement d'une image ou de l'exécution d'un code JavaScript
onFocus	BODY, FRAMESET, <INPUT txt,pwd>, <TEXTAREA>	Un élément devient actif (reçoit le focus)
onKeyDown	<INPUT txt,pwd>, <TEXTAREA>	L'utilisateur appuie sur une touche
onKeyPress	<INPUT txt,pwd>, <TEXTAREA>	L'utilisateur appuie, puis relâche une touche
onKeyUp	<INPUT txt,pwd>, <TEXTAREA>	L'utilisateur relâche une touche
onLoad	BODY, FRAMESET, IMG	Un document ou une image commence à se charger
onMouseDown	<A>, <AREA>, <INPUT others>	L'utilisateur appuie sur un bouton de la souris
onMouseMove	seulement interceptable	L'utilisateur déplace la souris
onMouseOut	<A>, <AREA>, <INPUT others>	L'utilisateur fait ressortir la souris de l'objet
onMouseOver	<A>, <AREA>	L'utilisateur place la souris sur un objet
onMouseUp	<A>, <AREA>	L'utilisateur relâche un bouton de la souris
onMove	BODY, FRAMESET	L'utilisateur déplace une fenêtre
onReset	FORM	Un formulaire est réinitialisé par un bouton reset
onResize	BODY, FRAMESET	L'utilisateur redimensionne une fenêtre ou un cadre
onScroll	BODY, FRAME	L'utilisateur se sert des barres de défilement de la page
onSelect	<INPUT txt,pwd>, <TEXTAREA>	L'utilisateur sélectionne du texte dans un champ
onSubmit	FORM	Un formulaire est soumis (bouton submit)

Tester les données d'un formulaire avant de les
envoyer au serveur

Validation des formulaire

- On va tester le formulaire localement en bloquant l'envoi si le résultat d'une `fonction vérifier(...)` est faux.
- ```
<form method="post" action="page.php" onSubmit="return verifier(this);">
 <!-- ici, le contenu du formulaire -->
 <input type="submit" value="Envoyer" /></form>
```
- ```
function verifier(f) {  
    if ( /* le formulaire est bien rempli */ )  
        { return true;  
          }  
    else {alert('Le formulaire est mal rempli');  
          return false;}  
}
```

Manipulation du DOM

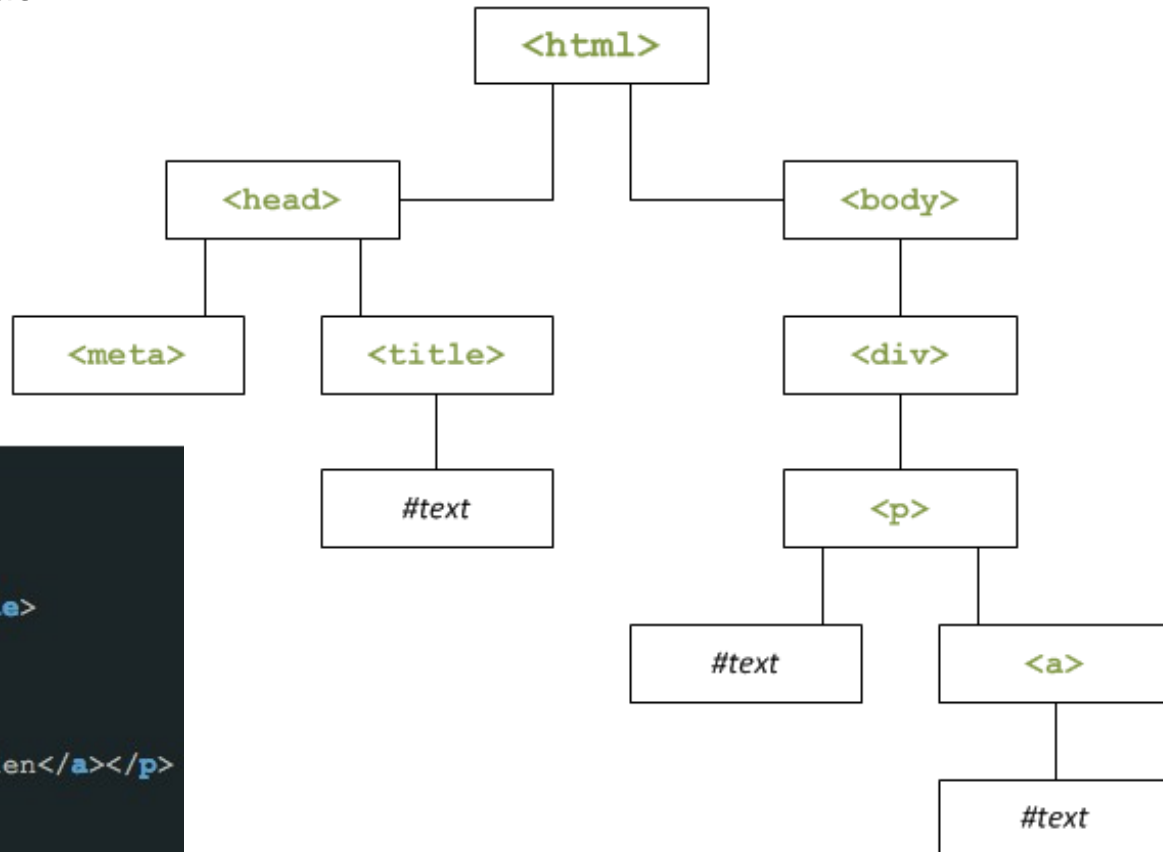
DOM : Document Object Model

Une page Web est vue comme un **arbre** → une hiérarchie d'éléments.

Il existe des instructions pour **modifier le DOM** en particulier pour

- **Ajouter/ Supprimer** des nœuds
- **Modifier** des noeuds

le tout à la volée **sans**
recharger la page



```
1 <!doctype html>
2 <html>
3   <head>
4     <meta charset="utf-8" />
5     <title>Le titre de la page</title>
6   </head>
7
8   <body>
9     <div>
10      <p>Un peu de texte <a>et un lien</a></p>
11    </div>
12  </body>
13 </html>
```

Nom

Prénom

Nb Enfant

Adresse

si on saisi 2 en nombre d'enfants
et qu'on donne le focus à profession
nous aurons deux zone de saisi qui
apparaîtront **au milieu** du formulaire
il faut recharger toute la page

Nom

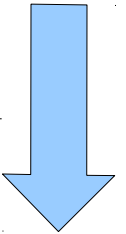
Prénom

Nb Enfant

Date naissance enfant 1

Date naissance enfant 2

Adresse



Démo impot.html dans répertoire du cours

Solution 1

Dans l'exemple précédent le DOM est modifié. Deux solutions sont possibles :

- on positionne un **DIV** vide entre « nb enfant » et « profession » et on fait une fonction qui va le surcharger en **y ajoutant du code HTML** modélisant les X zones de saisie des dates des enfants une fois que l'utilisateur aura saisi le nombre X d'enfants

Avant

```
Nb enfant <input type="text"/>
<div>
</div>
Adresse <input type="text"/>
```

Après

```
Nb enfant <input type="text"/>
  <div> date naissance enfant 1
    <input type="text"/>
  </div>
Adresse <input type="text"/>
```

Solution 1 : getElementById et innerHTML

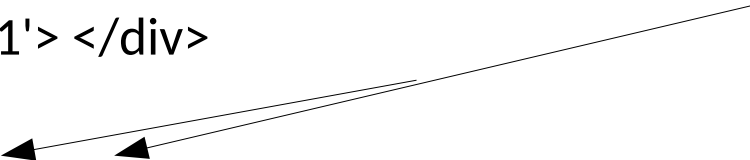
Il nous faut donc une instruction capable de prendre un nœud et de lui affecter du code HTML

Pour choisir un nœud on utilisera : `getElementById(chaine)` retourne le noeud du document dont l'attribut `id` est égale à `chaine` (null sinon)

Pour modifier la valeur HTML **du fils** d'un nœud on utilise l'attribut `innerHTML` du nœud père.

Dom initial : `<div id='id1'> </div>`

Utilisez les cotes si vous voulez mettre la valeur de l'id



JS : `getElementById('id1').innerHTML='<p> bla bla </p>';`
donc noeud père : `<div id='id1'>` et nœud fils : `<p> bla bla </p>`

ça aura pour conséquence de produire le DOM suivant

`<div id='id1'> <p> bla bla </p> </div>`

Solution 1 : getElementById et innerHTML

En partant donc du code initial :

Nb enfant <input type="text" name="nbEnfant" />

<div class="enfant">

</div>

Adresse <input type="text" />

il nous faut construire une fonction JS qui à la perte de focus de la zone de texte « nbEnfant » récupère X (le Nb enfant saisi) et effectue une boucle de 1 jusqu'à X dans lequel elle construit une chaîne « CH » de caractères contenant le code HTML de X zones de texte. Une fois cette boucle finie, on utilise l'instruction

```
getElementById('enfant').innerHTML=CH ;
```

La blise DIV aura donc dans le DOM un nœud fils représenté par la chaîne CH qui représente la liste des zones de saisie des enfants.

Solution 2 : naviguer dans le DOM

Avant

Nb enfant <input type="text"/>

Adresse <input type="text"/>

Après

Nb enfant <input type="text"/>

<div> date naissance enfant 1
<input type="text"/>

</div>

Adresse <input type="text"/>

Une page standard **sans rajouter de DIV vide** au départ et créer une fois le focus perdu un nouveau nœud DIV dans le DOM dans lequel on fera les mêmes étapes précédentes.

Difficulté : il faut créer **au bon endroit** le nœud DIV puis le remplir

Avantage : pas de gadget ou d'astuce qui dégrade la qualité du code comme les DIVs vide de la solution 1.

Solution 2 : naviguer dans le DOM

L'instruction `document.createElement(chaine);` crée un nœud DOM étiqueté par chaine. Ce nœud n'est pas encore relié au DOM

Par exemple : `var noeudDIV=document.createElement('div')`
crée un nœud `<div>`

L'instruction `noeud.prop=valeur` affecte valeur à l'attribut prop de nœud.
Par exemple sur le noed construit ci-dessus `noeudDIV.id='iddiv'` ;
On aura alors un nœud `<div id="iddiv">`

Il faut maintenant lier ce nœud quelque part au DOM !

Solution 2 : naviguer dans le DOM

L'instruction `noeud1.appendChild(noeud2)` ajoute noeud2 comme dernier fils du noeud1 dans le DOM

Par exemple :

HTML initial : `<div id="id1"> </div>`

JS : `noeudP=document.createElement('p');`

`noeudP.id='idp';`

`noeudDIV=document.getElementById('id1');`

`noeudDIV.appendChild(noeudP);`

HTML résultat `<div id="id1"> <p id="idp"> </p> </div>`

Et si je ne veux insérer **au milieu** et pas à la fin ?

Solution 2 : naviguer dans le DOM

Soit `noeud2` un nœud fils de `noeudPère`.

L'instruction `noeudPère.insertBefore(noeud1,noeud2)` insère le `noeud1` comme fils de `noeudPère` à la position qui est juste avant le `noeud2`

Par exemple :

HTML initial : `<div id="id1"> <p id="id2p"> </p> </div>`

```
JS : noeudPère=document.getElementById('id1') ;  
     noeud2= document.getElementById('id2p') ;  
     noeud1=document.createElement('p') ;  
     noeud1.id='id1p' ;  
     noeudPère.insertBefore(noeud1,noeud2) ;
```

HTML résultat `<div id="id1"> <p id="id1p"> </p> <p id="id2p"> </p> </div>`

Solution 2 : naviguer dans le DOM

Appliquons alors ce principe à l'exercice initial.

Avant

Nb enfant

Adresse

Après

Nb enfant

date naissance enfant 1

Adresse

Si le nom `<div>` n'existe pas alors on crée un nœud `<div>` à qui on affecte un id.

Pourquoi tester son existence ? Si on a déjà inséré un nombre d'enfant puis valider et que l'on souhaite changer ce nombre d'enfant. La page contient alors déjà un DIV et il ne faut pas en créer un second.

On fait une boucle comme pour la première solution pour générer le code HTML des zones de saisie des dates de naissance.

On **insère** le DIV juste avant la balise contenant l'**adresse**

On affecte via **innerHTML** ce code au DIV

Et plein d'autres instructions autour du DOM

`getElementsByName('mon_form')` retourne le tableau des éléments dont l'attribut name vaut "mon_form"

`getElementsByTagName('h1')` retourne le tableau des éléments du document de type "h1" (l'argument est un nom de balise)

`setAttribute`, `parentNode`, `firstChild`, `lastChild`, `nodeValue`, `data`, etc

Mal supporté encore : `querySelector()`, `querySelectorAll()`, ...

Changer dynamiquement le style d'un objet

```
document.getElementById(chaine).propriété=valeur;
```

Affecte « valeur » à la propriété « propriété » de l'objet dont l'id est chaine

On peut donc modifier dynamiquement le style de chaque élément de la manière suivante :

```
<div id="test">Je test</div>
```

```
document.getElementById('test').style.color='red' ;
```

Style contient plusieurs propriétés comme « color » pour manipuler dynamiquement le style d'un élément.

Il existe plusieurs propriétés comme « style » qui permettent de manipuler dynamiquement une élément (**positionnement**, etc...).

Survol de JQuery

Chargement de JQuery

- Récupérer la bibliothèque sur le site officiel jquery.com.
- Sélectionnez la version de production (qui pèse très léger)
- Le fichier "jquery-2.1.1.js" (la version est susceptible d'évoluer) s'affiche alors dans votre navigateur. Il ne vous reste plus qu'à sauvegarder le fichier sur votre ordinateur.

<head>

<script type="text/javascript" src="jquery-2.1.1.js"></script>

Chargement de JQuery

- Il faut s'assurer que les éléments de la page ont terminé d'être chargés
- jQuery fournit la fonction `ready()` qui permet de savoir le plus tôt possible quand la page est prête pour le JavaScript.
- Insérons le code correspondant dans la zone JavaScript de notre page HTML:

```
<head><script type="text/javascript">
```

```
$(document).ready(function(){
```

```
    // le code JQ à exécuter
```

```
});
```

- Pour l'instant notre page ne fait pas grand chose. Utilisons `alert()` pour afficher un message

```
$(document).ready(function(){
```

```
    alert("Jquery est lancé");
```

```
});
```

- Ecriture simplifiée pour la fonction ready qui permet d'économiser quelques caractères :

```
$(function(){ alert("Jquery est lancé"); });
```

Les événements

- Comment capter un **évènement utilisateur** en jQuery?

```
<div id="div1">
```

```
<p>Entete de ma page avec un <a href="#">lien 1</a></p>
```

```
</div>
```

```
<div id="div2">
```

```
<p>Contenu de ma page avec un <a href="#">lien 2</a></p>
```

```
</div>
```

```
<div id="div3">
```

```
<p>Pied de page avec un <a href="#">lien 3</a></p>
```

```
</div>
```

- Si je click sur un le lien 1 j'aurais le message

Entete de ma page avec un lien 1

Contenu de ma page avec un lien 2

Pied de page avec un lien 3

Vous venez de cliquer sur lien 1

OK

```
<script type="text/javascript">
$(document).ready(function(){
    $("a").click(function(){
        alert("Vous venez de cliquer sur "+ $(this).html());
        return false;
    });
});
</script>
```

- La fonction `$()` représente la fonction principale du framework jQuery → récupérer un élément de votre page HTML en fonction de sélecteurs CSS. Elle ajoute également de nouvelles propriétés à l'élément récupéré
- La fonction `$("a")` récupère tous les éléments HTML de la page qui vérifient le sélecteur CSS à savoir toutes les balises de type "a" : ce sont les liens 1, 2 et 3.
- La fonction `click()` est appliquée à chaque élément récupéré. On peut aussi changer le sélecteur pour récupérer uniquement le lien 2 dans le contenu de la page `$("#div2 a")`

```
<script type="text/javascript">
$(document).ready(function(){
    $("a").click(function(){ alert("Vous venez de cliquer sur "+ $(this).html()); return false; }); });
</script>
```

- Lorsque vous appliquez la fonction `click()` à un élément récupéré avec la fonction `$()`, vous lui ajoutez ce qu'on appelle un programme d'écoute ou **listener d'évènements**.
- Le traitement que vous voulez effectuer en réponse devra être décrit dans une fonction qui sera passée en paramètre de la fonction `click()`.
- Noter l'utilisation de l'objet `this` qui représente l'élément source de l'évènement. Dans notre cas `this` représente le lien `<a>` sur lequel l'utilisateur vient de cliquer.
- La fonction `click()` est un cas particulier de la fonction générale `bind()` qui permet d'ajouter un listener d'évènements pour n'importe quel type d'évènement. Le type d'évènement est passé comme **premier argument** de la fonction.

```
$("a").bind("click",function (){ ...} );
```

Si nous avions voulu ajouter un évènement lors du survol de la souris, nous aurions écrit :

```
$("a").bind("mouseover",function (){ ...} );
```

Ou la version plus simple

```
$("a").mouseover(function (){...} );
```


Animations JQuery

- JQ fournit un ensemble de fonctions qui permettent d'appliquer des animations à des éléments HTML
- Un lien qui permet d'afficher ou de cacher une zone de texte ou un objet div.

```
<body>
<a class="visible" href="#">Afficher</a>
<div id="cacher" style="display:none">
  <p>Mon texte caché s'affiche ou disparaît en fonction de mes clicks</p>
  <a class="nonvisible" href="#">Fermer</a>
</div>
</body>
```

Au lancement de la page on obtient Afficher du fait de la **propriété CSS** « display:none » qui cache le contenu du div dont l'id est « text-hidden »

- Si je click sur « afficher » j'obtiens Afficher
Mon texte caché s'affiche ou disparaît en fonction de mes clicks

```
<script type="text/javascript">
$(document).ready(function(){
  $("a.visible").click(function(){
    $("#cacher").show();
    return false;
  });
  $("a.nonvisible").click(function(){
    $("#cacher").hide();
    return false;
  });
});
</script>
```

- La fonction `show()` permet d'afficher des éléments cachés. Dans notre exemple, on arrive à afficher l'élément dont l'id est "cacher" qui était initialement caché grâce au `style="display:none"`.
- La fonction `hide()` permet de cacher des éléments visibles.
- Il existe plusieurs autres effets : fade, slide, jQuery animate, etc.

Modification dynamique du code HTML

- A chaque fois que je click sur un lien une zone de texte s'ajoute à mon formulaire. Par exemple je souhaite inscrire la liste des loisirs que je pratique.

Click ici pour ajouter un loisirs

- On utilise la fonction `appendTo()` qui signifie *ajouter à*.
`$(A).appendTo(B)` : ajoute tous les éléments spécifiés par le sélecteur A à ceux spécifiés par B

```
$(document).ready(function(){  
  $("a.add").click(function(){  
    $("<input type='text'></input>;").appendTo($("#form"));  
    return false;  
  });  
});
```

Le code Html de base :

```
<a href="#" class="add">Click ici pour ajouter un loisirs</a>  
<form>   </form>
```

Et encore plein d'autres choses possibles : remove, toggleClass, animate, JQuery UI, plein de plug-in etc.