

# Programmation événementielle : javascript

A.M PUIZILLOUT

Département MMI de l'IUT de Laval

1ere année

# Organisation

- 4 CMs

- 7 Tds

- 4 Tps

1 examen théorique (1h30)

1 examen pratique (2h)

Ressources : UMTICE - MMI - Semestre 2- Programmation Multimédia

**Clé:** MMIMULTIMEDIA

# Pré-Requis

Cours d'algorithmique S1 (variable, structure conditionnelle, structures répétitives, fonction, paramètre de fonction, résultat de fonction )

Cours HTML,CSS S1 (DOM, sélecteur, attribut, propriété de style)

Cours POO S2 (notion d'objet : instanciation, propriété, méthode)

Cours PHP : déterminer les parties de code qui s'exécutent sur le serveur , les parties de code qui s'exécutent sur le client

# Objectifs

Ce cours a pour objectif de :

- comprendre les mécanismes de la programmation événementielle
- comprendre les concepts de base du javascript afin de
  - créer des interactions simples
  - réutiliser de **manière pertinente** du code déjà écrit
  - être capable de découvrir par soi-même de nouveaux concepts
- comprendre les bases d'un framework javascript afin de :
  - créer des interactions simples
  - réutiliser de **manière pertinente** du code déjà écrit
  - être capable de découvrir par soi-même de nouveaux concepts

# Présentation

Javascript est un langage de programmation, créé par les sociétés Netscape et Sun Microsystems vers la fin de l'année 1995 (sous le nom de livescript).

**Son objectif principal** : introduire de l'interactivité avec les pages HTML, et effectuer des traitements **sur le poste de travail de l'utilisateur**. Jusqu'alors la programmation ne pouvait être exécutée que sur le serveur.

Le moyen utilisé : introduire des petits "bouts" de programme, appelés **SCRIPTS**, dans les pages HTML.

# Présentation

**Interactivité (n.f)** : activité de dialogue entre un individu et une information fournie par une machine (Petit Robert 1992).

**interactif** : Qui permet d'utiliser le mode conversationnel (terme informatique) .

Pour nous, cela consiste à réagir à des demandes de l'internaute en modifiant le contenu et/ou la mise en forme de la page web qu'il consulte.

La majorité de ces demandes sont déclenchées par des actions qu'il réalise sur des éléments de la page.

Par exemple, il clique ou double clique sur certains éléments de la page, il déplace, il frappe une ou plusieurs touches du clavier, il déplace des éléments de la page, il valide les informations qu'il vient de saisir, il scrolle sur la page, etc. Certaines actions peuvent être liées à des périphériques donnés comme l'action de .... sur un écran tactile.

# Présentation

Des exemples d'utilisation :

[Zoombox](#)

[Des dessins et des animations](#)

[Galerie 1](#)

[Galerie2](#)

[Appareilphoto](#)

[Autocompletion](#)

[Jeu 2d](#)

# Présentation

La possibilité d'inclure des programmes dans les pages HTML et de les exécuter directement sur le poste client est intéressante, car elle permet une réponse quasi-immédiate à une action utilisateur.

L'exécution du programme est assuré par le navigateur client (qui a en charge la création de l'arbre DOM et sa maintenance).

Au fil des années, javascript est devenu un langage prépondérant, de nouvelles APIS sont mises au point (AJAX, géolocalisation, animation 2D/3D, sauvegarde de données, web socket,...)

Avec le framework NODE.js, il est aussi devenu un langage pouvant être utilisé du coté serveur.



# Présentation

JavaScript est aujourd'hui défini par la norme ECMA-262, aussi connue sous l'appellation **ECMAScript (norme W3C)**

Microsoft a aussi implémenté cette norme sous le nom de **JScript**

La version actuellement supportée par les navigateurs: ECMA-262 version 5.1

<http://www.ecma-international.org/ecma-262/5.1/>

Une nouvelle version ECMA-262 édition 6 communément appelée ES6 a été définie en avril 2015. Elle commence à être mise en place par les navigateurs mais ,à l'heure actuelle, il faut utiliser un **polyfill** ou **transcompilateur**(code javascript permettant de mettre en place des fonctionnalités que les navigateurs ne possèdent pas nativement) .On peut par exemple utilisé le framework Babel

<http://www.ecma-international.org/ecma-262/6.0/index.html>

# La notion de programmation événementielle

# La notion de programmation événementielle

Quand Anne-Marie Puizillout lève sa main gauche  
Tous les étudiants de l'amphi frappent une fois dans leur main.

L'action : Lever la main gauche

Le sujet associé à l'action (objet observé): AM.  
Puizillout

La réaction obtenue quand l'action a lieu : tous les  
étudiants de l'amphi frappent dans leur main

# La notion de programmation événementielle

Donner la consigne : exécuté une fois et une seule

La réaction obtenue quand l'action a lieu : tous les élèves frappent dans leur main

s'exécute à chaque fois que l'action a lieu. Cela peut donc ne jamais s'exécuter (AM. Puizillout ne lève jamais la main gauche) aussi bien que 2000 fois

# La notion de programmation événementielle

## La réaction obtenue quand l'action a lieu

Il faut identifier :

- Les objets qui réagissent : ils peuvent être complètement différents de celui observé (ici le sujet observé est Anne-Marie Puizillout et les sujets qui réagissent sont les étudiants de l'amphi) ou cela peut être le même objet
- La réaction que l'on veut obtenir de ces objets : ici chaque sujet frappe une fois dans ces mains

# La notion de programmation événementielle

## Comment décliner cela sur une page web : exemple1

Quand le bouton "applaudir" est cliqué, l'audio "applaudissement.mp3" se joue.

1. On met en place les éléments sur la page HTML

→ `<input type="button" value="applaudir" id="btnApplaudir"/>`

→ `<audio src="applaudissement.mp3" preload="metadata" >`

2. On définit l'interactivité

L'action : "click"

L'objet à l'origine de l'action : l'élément input

La réaction obtenue quand l'action a lieu :

l'objet qui réagit : l'élément audio

La réaction demandée : l'audio se met en lecture

# La notion de programmation événementielle

La terminologie "programmation événementielle" associée

action → événement

les objets → les éléments du DOM (Document Object Model)

définir la réaction associée → écrire la fonction écouteur

associer l'action et l'objet observé et la fonction écouteur associée → mettre en place un écouteur sur l'élément du DOM observé

# La notion de programmation événementielle

- La mise en place de l'écouteur ne s'exécute qu'une seule fois
- La fonction écouteur associée s'exécutera de 0 à n fois
- Un événement peut se produire sur un objet même si aucun écouteur n'est mis en place. Par contre, cela n'engendre évidemment aucune réaction



# La notion de programmation événementielle

- Quand Anne-Marie Puizillout lève la main gauche  
Tous les élèves de l'amphi frappent une fois dans leur main.
- Quand Anne-Marie Puizillout lève la main droite, les étudiants de la première rangée d'étudiants se lèvent
  - On peut observer plusieurs actions différentes sur un même objet c'est-à-dire qu'on peut mettre en place plusieurs écouteurs sur un objet
  - Lorsqu'un objet est modifié, **il reste en l'état jusqu'à la prochaine modification** → les étudiants du premier rang restent levés jusqu'à ce qu'on leur demande de s'asseoir

# La notion de programmation événementielle

Comment décliner cela sur une page web : [exemple2](#)

Quand le bouton "applaudir" est cliqué, l'audio "applaudissement.mp3" se joue.

Quand le bouton "applaudir" est survolé, le bouton "applaudir" change de couleur (couleur de fond noire et couleur de texte blanche)

# La notion de programmation événementielle

- Quand Anne-Marie Puizillout lève la main gauche

Tous les élèves de l'amphi frappent une fois dans leur main.

- Quand Anne-Marie Puizillout lève la main gauche, les étudiants de la dernière rangée d'étudiants se lèvent

➤ On peut associer plusieurs fonctions écouteurs différentes à une même action sur un même objet (ici le fait qu'AM Puizillout lève la main gauche)

➤ Les fonctions écouteurs se déclenchent alors l'une après l'autre dans l'ordre de leur mise en place. Sur l'exemple donné, les étudiants frappent d'abord dans leur main puis se lèvent

# La notion de programmation événementielle

## Comment décliner cela sur une page web : [exemple3](#)

Quand le bouton "applaudir" est cliqué, l'audio "applaudissement.mp3" se joue.

Quand le bouton "applaudir" est cliqué, l'arrière-plan de body change de couleur (il devient rouge)

L'internaute peut avoir l'impression qu'il y a une simultanéité entre le lancement du son et le changement de couleur de la page. Malgré tout, le navigateur a d'abord lancé le son avant de changer la couleur de la page mais comme cela se passe dans un intervalle très court (micro-seconde), nous avons cette impression de simultanéité.

# La notion de programmation événementielle

Nous aurions pu aussi définir un seul écouteur et modifier les 2 objets (l'audio et body) dans la fonction écouteur associée.

Quand le bouton "applaudir" est cliqué :

- l'audio "applaudissement.mp3" se joue.
- l'arrière-plan de body change de couleur (il devient rouge)

[exemple4](#)

Le résultat pour l'internaute est exactement identique dans les 2 cas

# La notion de programmation événementielle

- Votre code Javascript est entièrement **dépendant** du navigateur qui l'exécute. Nous allons être confronté aux mêmes problèmes de compatibilité et de respect des standards que lors du cours HTML5/CSS
- C'est un **langage interprété**: les instructions sont traduites en langage "binaire" au fur et à mesure de l'exécution (ralentit les performances)
- De plus en plus de navigateur, compile votre code javascript juste avant de l'exécuter (**Just In Time Compiler**). Pour vous cela est transparent (vous fournissez le source du code et non pas sa version exécutable) mais cela augmente de manière sensible les performances de javascript. Google chrome a amélioré les temps de traitement de 25% dans la version V8 de son moteur javascript
- Dans tous les cas, il n'y a pas, pour vous, d'étape de compilation formelle donc vous ne découvrirez vos erreurs de frappe, de syntaxe qu'au moment de l'exécution.

# La notion de programmation événementielle

Tout le code javascript que vous écrirez sera stocké dans un (ou plusieurs fichiers) , ces fichiers ayant **.js** comme extension.

ex: td1Exercice1.js

**Bien évidemment, comme pour les fichiers css , ce fichier devra être associé à la page html dans la partie <head> ou <body>.**

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

Ex : `<script type="text/javascript" src="td1Exercice1.js"></script>`



Souvent, votre code ne fonctionne pas car vous ne l'avez pas ou vous l'avez mal associé à votre page. Les outils de développement vous permettront très rapidement de vérifier que votre fichier javascript a bien été chargé par le navigateur

# La notion de programmation événementielle

Il est aussi possible, si besoin, d'intégrer le code source javascript directement dans la page en utilisant les balises **<script>** Votre code javascript **</script>**

Il est malgré tout nettement préférable de dissocier son code javascript de son code HTML (même principe que pour la css)



# La notion de programmation événementielle

Avant de vous mettre à programmer l'interactivité de votre site, il faut d'abord lister les interactions que vous souhaitez mettre en oeuvre :

1. identifier les objets interactifs sur votre page
2. identifier les événements associés à ces objets
3. pour chaque couple objet-événement, définir la réaction attendue
  - A. lister les objets à modifier
  - B. pour chaque objet, lister les modifications souhaitées

Manipuler les objets de la page

# Manipuler les objets de la page

Les principes de programmation objet vont s'appliquer à ce cours javascript :

## Rappel :

Avant d'afficher la page, le navigateur va d'abord analyser le code HTML et créer en mémoire un **objet** pour chaque élément trouvé dans ce code (balise, texte, commentaire,...). Ces objets sont liés les uns aux autres par une arborescence :

Cette arborescence s'appelle **l'ARBRE DOM**

**Chaque élément de cet arbre est appelé nœud de l'arbre.**

**C'est à partir de cet arbre DOM que la page est ensuite affichée**

Via javascript , généralement on manipule :

- des nœuds éléments
- des nœuds textes

# Manipuler les objets de la page

Donc tout élément visible (ou invisible) sur votre page correspond à un objet. Comme tout objet, chaque objet a des caractéristiques(attributs) et un comportement défini(méthode) .

Par contre, un certain nombre d'étapes nécessaires à la programmation objet sont **prises en charge par le navigateur** :

- **L'écriture des classes** correspondant à chaque objet ("modèle") . Le modèle que doit respecter chaque objet est défini par la documentation du W3C

- **L'instanciation des objets** : les objets seront en grande majorité instanciés par le navigateur lors de l'analyse du flux HTML(il sera malgré tout possible d'instancier de nouveaux objets par code)

- Le **DOM** (Document Object Model) est une API (définie par le W3C) permettant d'obtenir que toutes les applications et notamment les navigateurs accèdent de manière identique à ces objets. Il définit donc les propriétés et les méthodes des différents objets manipulés

# Manipuler les objets de la page

- Tout attribut HTML correspond à une propriété de l'objet HTML  
    <a href="page1.htm"> → l'objet DOM correspondant à cette balise a une propriété href

- Le fait de définir un attribut dans le code HTML de la page permet de définir la valeur de la propriété

La propriété href de l'objet du DOM aura pour valeur "page1.htm"

- Si l'attribut n'est pas défini dans le code HTML, la propriété existe malgré tout dans l'objet du DOM mais elle est initialisée avec une valeur par défaut définie par le W3C

<input type="checkbox" />

L'attribut "checked" n'est pas défini pourtant la propriété checked existe et est initialisée à false (le bouton n'est pas coché par défaut)

# Manipuler les objets de la page

- Il existe aussi des propriétés ne correspondant à aucun attribut
- Ex : la propriété **currentTime** pour un objet audio et video qui permet de connaître à quel endroit en est la lecture du média

Tous les nœuds disposent aussi des propriétés suivantes :

## Le type du nœud

**nœud.nodeType** : type du nœud

1: nœud élément

3: nœud texte

8: nœud commentaire ....

## Le nom du nœud

**nœud.nodeName** : #text pour les nœuds textes, le nom de la balise pour les nœuds éléments (en majuscule. Ex DIV,P,TITLE)

# Manipuler les objets de la page

## La valeur du nœud:

**nœud.nodeValue**: la valeur du texte pour un nœud texte, null pour les nœuds éléments

Les autres attributs d'un nœud sont les attributs html ( ex src pour une image, href pour un lien , title ou id pour tous)

## Le contenu HTML d'un nœud élément

**nœud.innerHTML**: le code HTML ou le texte qui se trouve entre la balise ouvrante et la balise fermante

Ex :

```
<div id="div1"><p id="p1"> paragraphe1 </p></div>
```

innerHTML de div1 a pour valeur :<p> paragraphe 1 </p>

innerHTML de p1 a pour valeur :paragraphe 1

# Les attributs communs aux noeuds

Enfin les objets disposent **de méthodes** qui permettent d'exécuter des actions

Certaines sont communes à tous les nœuds , d'autres spécifiques à des éléments HTML particuliers.

Ex : la méthode `remove()` est commune à tous les nœuds et permet de le supprimer

la méthode `pause()` est propre aux objets audios et vidéos et permet d'arrêter leur lecture.





# Manipuler les objets de la page



Dans la norme du W3C , **les retours chariots** "de présentation" font partie du DOM : **ce sont des nœuds textes** (dit blanc ou vide car il n'y a pas de valeur). Tous les navigateurs ne les gèrent pas de la même façon : Firefox, opera, chrome les intègrent dans leur DOM , certaines versions d'Internet Explorer non.

Ex

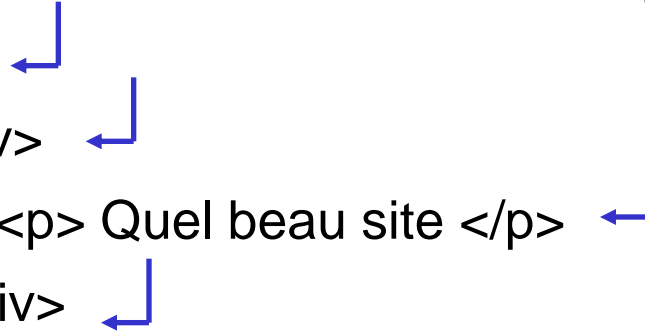
```
<div id="div1">   
       
</div>
```

le premier enfant de la div "div1" est un nœud texte vide et la div1 a 3 enfants : 1 nœud élément et 2 nœuds textes

# Manipuler les objets de la page

**Question1 : Combien d'objets de type nœud élément sont créés dans le DOM suite à l'analyse du flux HTML suivant:**

```
<div>
  <div>
    <p> Quel beau site </p>
  </div>
</div>
```

The diagram illustrates the DOM tree structure for the provided HTML. It shows a root <div> element containing a child <div> element, which in turn contains a <p> element. Blue arrows indicate the parent-child relationships: from the root <div> to its child <div>, from that <div> to the <p> element, and from the <p> element to its closing tag </p>. Another blue arrow points from the inner <div> to its closing tag </div>.

**3** (les 2 éléments div et l'élément p)

**Question1 bis: Combien d'objets de type nœud texte sont créés par le DOM suite à l'analyse du flux HTML ci-dessus:**

**5** (le texte "Quel beau site" et 4 nœuds texte vides liés au retour chariot)

# Manipuler les objets de la page

**Question 2 : Citer des propriétés possibles d'un élément img**

src, alt, title, id , class, style, ...

**Question 3 : Citer des méthodes possibles d'un élément video**

la méthode pour mettre la vidéo en lecture : play()

la méthode pour arrêter la lecture : pause ()

.....

# Manipuler les objets de la page

Le **DOM** (Document Object Model) est une API (définie par le W3C) permettant d'obtenir que toutes les applications et notamment les navigateurs accèdent de manière identique à ces objets. Il définit donc les propriétés et les méthodes des différents objets manipulés

Documentation MDN : <https://developer.mozilla.org/fr/docs/Web/API>

# Les autres objets

- Il existe d'autres objets standards supportés par le langage : String, Date, RegExp,.....

Documentation MDN : <https://developer.mozilla.org/fr/docs/Web/JavaScript>  
et choisir l'onglet "objet natif"

On peut aussi créer par programmation **ses propres objets**

# Les objets du DOM : L'objet window

Au chargement de la page, le premier objet créé par le navigateur est l'objet **window**

Cet objet dispose d'un certain nombre de propriétés identifiant l'environnement de la **fenêtre**. Voici les principaux :

- **window.location** : barre d'adresse
- **window.status** : la barre d'état
- **window.navigator** : le navigateur utilisé
- **window.screen** : l'écran dont dispose votre internaute
- **window.document** : le document chargé dans la fenêtre
- **window.innerWidth/window.innerHeight** : les dimensions de la fenêtre

• L'objet window est toujours sous-entendu, on peut omettre son nom :

`window.screen.width` ↔ `screen.width`

# Les objets du DOM : L'objet window

Les principales méthodes dont il dispose, permettent :

- de mettre en place des **timers** qui permettront d'exécuter des actions répétitives toutes les x milli-secondes
  - window.setTimeout (...)
  - window.setInterval(....)
- d'**imprimer** la page
  - window.print(...)
- d'ouvrir des **pops-ups**
  - window.open(...)
- d'afficher des **boîtes de dialogues**
  - window.alert (...)
  - window.confirm(....)
  - window.prompt(....)

# Les objets du DOM : L'objet document

- Le deuxième objet instancié par le navigateur est l'objet **document**, cet objet est instancié systématiquement, par le navigateur, dès le démarrage du chargement de la page.
- On y accède via la syntaxe : **window.document** ou bien encore **document** (window étant alors sous entendu)
- Il correspond au document chargé dans la fenêtre.
- Il dispose d'un certain nombre de propriétés et de méthodes permettant **d'accéder aux différents éléments de la page**
- Il permet aussi de **créer de manière dynamique (par code) de nouveaux éléments dans la page**



# Les objets du DOM : L'objet document

- `document.documentElement` → permet d'accéder à l'élément du DOM `<html>`
- `document.title` → permet d'accéder à l'élément du DOM `<title>`
- `document.body` → permet d'accéder à l'élément du DOM `<body>`
- `document.anchors` → permet d'avoir une collection de tous les éléments `<a>` du DOM
- `document.forms` → permet d'avoir une collection de tous les éléments `<form>` du DOM
- `document.images` → permet d'avoir une collection de tous les éléments `<img>` du DOM

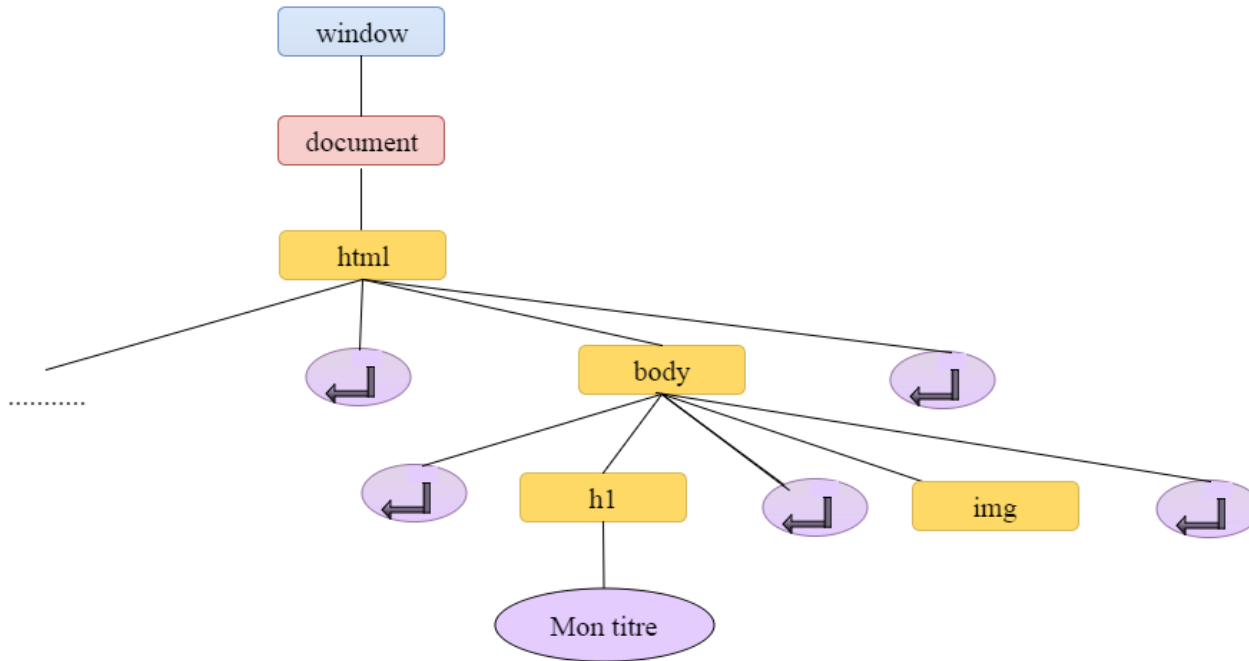
# Les objets noeuds éléments et noeuds textes

Le navigateur va ensuite analyser le code HTML qu'il a reçu du serveur web et va instancier (créer)

- un **noeud élément** à chaque fois qu'il rencontre une balise
- un **noeud texte** à chaque fois qu'il rencontre un texte

# L'arbre DOM

L'organisation hiérarchique de ces différents objets



```
<html>
.....
<body>
  <h1>Mon titre </h1>
  <img ..... />
</body>
</html>
```

**Combien d'objets ont été instanciés par le navigateurs ?**

# L'API DOM : Accéder à des nœuds

Tout code javascript aura pour problématique de « récupérer » dans le DOM, des objets de type nœud élément ou nœuds textes afin mettre en place des écouteurs ou bien afin de les modifier, les supprimer, les rendre invisibles, etc

2 types d'objets ont la capacité d'accéder à des nœuds définis dans le DOM :

- L'objet **document** : sa recherche s'effectue dans tout l'arbre DOM
- Les **nœuds éléments** : la recherche s'effectue parmi leurs descendants (rappel : les enfants font partie des descendants)

# L'API DOM : Accéder à des nœuds

- Pour accéder à **UN nœud-élément précis** on peut utiliser **l'id** de ce nœud : **document.getElementById(id recherché)**

Ex : `document.getElementById("lien1");`

- Pour accéder à **un ENSEMBLE de nœuds-éléments** correspondant à un **tag** (balise) donné :

**document.getElementsByTagName(nom du tag)** : recherche dans le DOM

Ex : `document.getElementsByTagName("div");`

**ou noeudElement.getElementsByTagName(nom du tag)** : La recherche s'effectue parmi les descendants du nœud élément

Ex : `document.getElementById("div1").getElementsByTagName("div");`

# L'API DOM : Accéder à des nœuds

- Pour accéder à **un ENSEMBLE de nœuds-éléments** correspondant à un **nom** (attribut name) donné :

**document.getElementsByName(nom recherché)**

Ex : `document.getElementsByName("btnTransport");`

Rq : Principalement utilisée pour input radio ou checkbox

- Pour accéder à **un ENSEMBLE de nœuds-éléments** correspondant à une classe donnée (attribut class) :

**document.getElementsByClassName(classe recherchée)**: recherche dans tout le dom ceux ayant la class demandée

**nœudElement.getElementsByClassName(classe recherchée)**: recherche parmi les descendants du nœud élément ceux ayant la class demandée

# L'API DOM : Accéder à des nœuds

- Pour accéder à **un ENSEMBLE de nœuds-éléments** correspondant à un **sélecteur css**:

**document.querySelectorAll("sélecteur css")**: recherche dans tout le dom

Ex : `document.querySelectorAll("div>:first-child")`; ramènera le premier enfant de chaque div de la page

**nœudElement.querySelectorAll("sélecteur css")**: recherche dans les descendants du nœud élément

- Pour accéder à **UN nœud élément** correspondant au premier élément vérifiant le sélecteur css.

**document.querySelector("sélecteur css")**: recherche dans tout le dom

Ex : `document.querySelector("div>:first-child")`; ramènera le premier enfant de la première div de la page

**\_nœudElement.querySelector("sélecteur css")**: recherche dans les descendants du nœud élément

# L'API DOM: accéder à des nœuds éléments

- Seul l'objet document disposent des méthodes `getElementById` et `getElementsByName`



- Les méthodes `getElementsByTagName`, `getElementsByName`, `getElementsByClassName` et `querySelectorAll` ramènent toujours leur résultat sous forme de collection (tableau) même s'il n'y a aucun objet ou qu'un seul objet dans celle-ci.

- Pour accéder à un des objets de la collection, il faut utiliser la notion **d'indice entre [ ]** (rang de l'élément dans la collection → 0 : premier élément, 1:deuxième élément, etc )

Ex: `var lesInputs= document.getElementsByTagName("input");`

`var input1= lesInputs[0];`

`var input4= lesInputs[3];`



# L'API DOM: accéder à des nœuds éléments

- Pour accéder à l'ensemble des éléments d'une collection, il faut utiliser une structure répétitive généralement de type `for`. La propriété `length` d'une collection permet d'en connaître le nombre



```
var lesDivs=document.querySelectorAll('div');
```

```
for (var i=0; i<lesDivs.length;i=i+1) {  
    var uneDiv=lesDivs[i];  
    .....  
}
```

# L'API DOM: accéder à des nœuds éléments

- Lorsqu'une méthode a pour résultat un élément (getElementById et querySelector) , elle ramène **null** si aucun élément ne correspond à la recherche.



- Lorsqu'une méthode retourne une collection d'éléments (getElementsByTagName, getElementsByName, getElementsByClassName, querySelectorAll), elle ramène un tableau vide si aucun élément ne correspond à la recherche

• **Null = aucun objet**

• **Tableau vide = 1 objet (le tableau) mais avec 0 case → propriété length = 0**

# L'API DOM: accéder à des nœuds éléments

.....

```
<body> ①
  <div id="div1"> ②
    <img src=img1.jpg" id="img1" class="c1" /> ③
    <img src=img2.jpg" id="img2" class="c2" /> ④
  </div>
  <div id="div2"> ⑤
    <img src=img3.jpg" id="img3" /> ⑥
    <img src=img4.jpg" id="img4" class="c1" /> ⑦
  </div>
</body>
</html>
```

Donnez le n° des éléments sélectionnés

(1) document.getElementById("div1")

(2) document.getElementById("IMG1")

(3) document.getElementsByTagName("img")

(4) document.getElementsByClassName("c1")

(5) document.querySelectorAll("img:last-child")

(6) document.querySelector("div")

# L'API DOM: accéder à des nœuds éléments

.....

```
<body> ①
  <div id="div1"> ②
    <img src=img1.jpg" id="img1" class="c1" /> ③
    <img src=img2.jpg" id="img2" class="c2" /> ④
  </div>
  <div id="div2"> ⑤
    <img src=img3.jpg" id="img3" /> ⑥
    <img src=img4.jpg" id="img4" class="cl1" /> ⑦
  </div>
</body>
</html>
```

Donnez le n° des éléments sélectionnés

(1) `document.body.querySelector(":first-child")`

(2) `document.getElementById("div2").querySelectorAll("img")`

(3) `document.querySelector("div").getElementsByTagName("img")`

# L'API DOM : Naviguer dans le DOM

Les nœuds éléments disposent aussi de propriété permettant **d'accéder aux éléments ayant un lien de parenté avec eux**

Nom de la propriété	Résultat
<code>childNodes</code>	une collection contenant tous ses nœuds enfants (nœuds textes et nœud éléments)
<code>children</code>	Une collection contenant tous les <b>nœuds éléments</b> enfants
<code>firstChild</code>	le 1 <sup>er</sup> nœud enfant (ou null s'il n'en a pas)
<code>firstElementChild</code>	Le 1 <sup>er</sup> <b>nœud élément</b> enfant (ou null s'il n'en a pas)
<code>lastChild</code>	le dernier nœud enfant (ou null s'il n'en a pas)
<code>lastElementChild</code>	le dernier nœud <b>élément</b> enfant (ou null s'il n'en a pas)
<code>nextSibling</code>	nœud "frère" suivant (ou null s'il n'y en a pas)
<code>nextElementSibling</code>	Nœud <b>élément</b> "frère" suivant (ou null s'il n'y en a pas)
<code>previousSibling</code>	nœud "frère" qui le précède (ou null s'il n'y en a pas)
<code>previousElementSibling</code>	nœud <b>élément</b> "frère" qui le précède (ou null s'il n'y en a pas)
<code>parentNode</code>	nœud parent

# L'API DOM : Naviguer dans le DOM

La présence des nœuds textes va souvent modifier le résultat obtenu par les méthodes `firstSibling` et `firstElementSibling` ou `nextSibling` et `nextElementSibling`, etc

Ex :

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

```
  <h1>Mon titre </h1>
```

```
  <img ..... />
```

```
</div>
```

`document.getElementById("div1").firstChild` → nœud texte vide

`document.getElementById("div1").firstElementChild` → nœud `<h1>`

`document.querySelector("h1").nextSibling` → nœud texte vide

`document.querySelector("h1").nextElementSibling` → nœud `<img>`

# L'API DOM: accéder à des nœuds éléments

.....

```
<body> ①
  <div id="div1"> ②
    <img src=img1.jpg" id="img1" class="c1" /> ③
    <img src=img2.jpg" id="img2" class="c2" /> ④
  </div>
  <div id="div2"> ⑤
    <img src=img3.jpg" id="img3" /> ⑥
    <img src=img4.jpg" id="img4" class="cl1" /> ⑦
  </div>
</body>
</html>
```

Donnez le n° des éléments sélectionnés

(1)

`document.getElementById("div2").lastChild`

(2) `document.getElementById("div2").lastElementChild`

(2) `document.getElementById("div1").nextElementSibling`

Mettre en place un  
écouteur



# Mettre en place un écouteur

## Comment gérer les actions utilisateurs ?

Lorsqu'une action utilisateur détectable par le navigateur se produit (manipulation de la souris, du clavier, ...), le navigateur va envoyer aux éléments html concernés un **message** indiquant que l'événement (par ex un **click**) vient de se produire.

A charge pour l'élément HTML de réagir ou non à cet événement.

Notre travail dans cette phase va donc consister à indiquer les éléments qui doivent réagir à un événement donné.

Certains événements ne pourront être traités que par certains types d'éléments :

Ex : l'événement submit ne peut être géré que par les éléments `<form>`

# Les évènements utilisateurs

Evènement “utilisateur”	Description
<b>click</b>	Se produit lorsque l'utilisateur clique sur l'élément associé à l'évènement
<b>change</b>	Survient lorsque l'élément perd le focus et que la valeur de l'élément a changé par rapport à la valeur initiale
<b>blur</b>	Survient lorsque l'élément perd le focus
<b>focus</b>	Survient lorsque l'élément prend le focus (pour les champs saisissables , le curseur se trouve dans l'élément)
<b>keypress</b>	Survient lorsque l'utilisateur appuie sur une touche du clavier
<b>keydown</b>	Survient lorsque une touche du clavier est enfoncée
<b>keyup</b>	Survient lorsque une touche du clavier est relâchée
<b>mousedown</b>	survient lorsque l'utilisateur appuie sur un bouton de la souris
<b>mousemove</b>	survient lorsque l'utilisateur bouge la souris
<b>mouseover</b>	survient lorsque le pointeur de souris entre sur un élément du DOM ou sur un de ces descendants
<b>mouseenter</b>	survient lorsque le pointeur de souris entre sur un élément du DOM
<b>mouseout</b>	survient lorsque le pointeur de souris quitte l'élément du DOM ou un de ses descendants
<b>mouseleave</b>	survient lorsque le pointeur de souris quitte l'élément du DOM (mais ne se produit pas lorsqu'il reste au dessus d'un de ses descendants)
<b>mouseup</b>	survient lorsque l'utilisateur relache un bouton de la souris
<b>reset</b>	survient lorsque l'utilisateur clique sur un bouton reset
<b>select</b>	survient lorsque l'utilisateur sélectionne tout ou partie d'un texte
<b>submit</b>	survient lorsque l'utilisateur clique sur un bouton submit
<b>dblclick</b>	survient lorsque l'utilisateur double-clique sur l'élément associé à l'évènement
<b>resize</b>	survient lorsque l'utilisateur redimensionne un élément (généralement la fenêtre)
<b>touchstart</b>	Ecran tactile : Se produit lorsque le doigt est placé sur un élément du DOM.
<b>touchend</b>	Ecran tactile : Se produit lorsque le doigt est retiré de la surface tactile
<b>touchmove</b>	Ecran tactile : Se produit lorsque le doigt est déplacé sur un élément du DOM.
<b>touchleave</b>	Ecran tactile : Se produit lorsque le doigt est retiré d'un élément du DOM.
<b>touchcancel</b>	Ecran tactile : Se produit lorsque le toucher est interrompu d'un élément du DOM.

# Les évènements utilisateurs

Evènement	Supporté par quel élément html
<b>click,dblclick</b>	<a>, <address>, <area>, <b>, <bdo>, <big>, <blockquote>, <body>, <button>, <caption>, <cite>, <code>, <dd>, <dfn>, <div>, <dl>, <dt>, <em>, <fieldset>, <form>, <h1> to <h6>, <hr>, <i>, <img>, <input>, <kbd>, <label>, <legend>, <li>, <map>, <object>, <ol>, <p>, <pre>, <samp>, <select>, <small>, <span>, <strong>, <sub>, <sup>, <table>, <tbody>, <td>, <textarea>, <tfoot>, <th>, <thead>, <tr>, <tt>, <ul>, <var>
<b>change</b>	<input type="text">, <select>, <textarea>
<b>blur,focus</b>	<a>, <acronym>, <address>, <area>, <b>, <bdo>, <big>, <blockquote>, <button>, <caption>, <cite>, <dd>, <del>, <dfn>, <div>, <dl>, <dt>, <em>, <fieldset>, <form>, <frame>, <frameset>, <h1> to <h6>, <hr>, <i>, <iframe>, <img>, <input>, <ins>, <kbd>, <label>, <legend>, <li>, <object>, <ol>, <p>, <pre>, <q>, <samp>, <select>, <small>, <span>, <strong>, <sub>, <sup>, <table>, <tbody>, <td>, <textarea>, <tfoot>, <th>, <thead>, <tr>, <tt>, <ul>, <var>
<b>keypress</b>	<a>, <acronym>, <address>, <area>, <b>, <bdo>, <big>, <blockquote>, <body>, <button>, <caption>, <cite>, <code>, <dd>, <del>, <dfn>, <div>, <dt>, <em>, <fieldset>, <form>, <h1> to <h6>, <hr>, <i>, <input>, <kbd>, <label>, <legend>, <li>, <map>, <object>, <ol>, <p>, <pre>, <q>, <samp>, <select>, <small>, <span>, <strong>, <sub>, <sup>, <table>, <tbody>, <td>, <textarea>, <tfoot>, <th>, <thead>, <tr>, <tt>, <ul>, <var>
<b>mousedown,mouse move, mouseout,mouseover</b>	<a>, <address>, <area>, <b>, <bdo>, <big>, <blockquote>, <body>, <button>, <caption>, <cite>, <code>, <dd>, <dfn>, <div>, <dl>, <dt>, <em>, <fieldset>, <form>, <h1> to <h6>, <hr>, <i>, <img>, <input>, <kbd>, <label>, <legend>, <li>, <map>, <ol>, <p>, <pre>, <samp>, <select>, <small>, <span>, <strong>, <sub>, <sup>, <table>, <tbody>, <td>, <textarea>, <tfoot>, <th>, <thead>, <tr>, <tt>, <ul>, <var>
<b>reset,submit</b>	<form>
<b>select</b>	<input type="text">, <textarea>
<b>resize</b>	<a>, <address>, <b>, <big>, <blockquote>, <body>, <button>, <cite>, <code>, <dd>, <dfn>, <div>, <dl>, <dt>, <em>, <fieldset>, <form>, <frame>, <h1> to <h6>, <hr>, <i>, <img>, <input>, <kbd>, <label>, <legend>, <li>, <object>, <ol>, <p>, <pre>, <samp>, <select>, <small>, <span>, <strong>, <sub>, <sup>, <table>, <textarea>, <tt>, <ul>, <var>

# interactivité native

Le navigateur a déjà programmé l'interactivité de certains éléments du DOM → **interactivité native**

**Citer des éléments naturellement interactif et indiquer à quelle action utilisateur ils répondent :**

a + click → on se débranche à l'adresse fournie via l'attribut href

area + click → on se débranche à l'adresse fournie via l'attribut href

form + submit → on exécute l'action définie par l'attribut action

élément ayant un attribut title + hover → affiche la valeur de title dans une zone au-dessus de l'élément

zone de saisie (input text, textarea, + keypress → concatène la touche pressée avec le texte déjà affiché

input à cocher (checkbox, radio) + click → coche ou décoche le bouton

etc

# Mettre en place un écouteur : via la CSS

On peut utiliser des sélecteurs CSS3 (pseudo-classes) pour détecter une action utilisateur et modifier la mise en forme des éléments en conséquence.

`:hover` → survol de la souris

`:checked` → un élément est coché

`:active` → la touche de souris est enfoncé au dessus de l'élément (disparaît dès que l'on relâche la touche)

`:focus` → un élément vient de recevoir le focus (principalement pour les zones de saisie lorsqu'on place le curseur à l'intérieur de la zone)

`:target` → l'élément cible d'un lien lorsque le lien vient d'être activé

# Evenements "Système"

Ils existent aussi beaucoup d'autres événements non directement liés à des actions utilisateurs mais plus à des **messages "systèmes"** transmis par le navigateur pour informer du chargement du DOM , d'un fichier, du début ou de la fin d'une animation ,de la lecture ou fin de lecture d'un flux, etc

Evènement	Description
<b>load</b>	Survient lorsque le navigateur a fini un chargement (page, image ,audio...)
<b>unload</b>	Survient lorsque le navigateur décharge la page
<b>transitionstart</b>	Une transition CSS3 commence
<b>transitionend</b>	Une transition CSS3 se termine
<b>animationstart</b>	Une animation CSS3 commence
<b>animationend</b>	Une animation CSS3 se termine
<b>animationiteration</b>	Une nouvelle itération d'une animation CSS3 commence
<b>play</b>	Un flux audio ou vidéo est mis en lecture
<b>pause</b>	Un flux audio ou vidéo est mis en pause
<b>ended</b>	Un flux audio ou vidéo est terminé
<b>etc</b>	Il existe beaucoup d'autres évènements systèmes

# Mettre en place un écouteur : via la CSS

Comment mettre en place un écouteur dans notre code?

On a longtemps intégré le code directement dans la page HTML (API DOM 0) :

**<balise `onevent`= "fonction javascript">**

`event` faisant référence à un des événements définis précédemment.

**ex**

```
<body onload="chargement()" >
```

```
<form id="form" action="page.php" onsubmit="verifier()">
```

```

```



Comme tous les attributs , l'attribut `onevent` s'écrit entièrement en minuscule en XHTML.

```
<body onload="chargement()" >
```

**et non**

```
<body onLoad="chargement()" >
```

# Mettre en place la gestion d'évènement

**Mais ça c'était AVANT**, maintenant on dissocie l'interactivité (donc le javascript) de la page html (même principe que pour la mise en forme)

La mise en place de la gestion des évènements se fera donc OBLIGATOIREMENT dans le fichier javascript

2 possibilités pour gérer cette interactivité :

- en utilisant les attributs `on*` des éléments (API DOM 0)
- en utilisant la méthode `addEventListener` de l' API DOM 2



# Mettre en place la gestion d'évènement

**1 ) En utilisant l'API DOM 0 et les attributs onevent (onclick, onmouseover ,.....)**

**objet.onevent=nomFonctionAExecuter;**

*Ex :pour mettre une bordure rouge sur l'image 1 lorsqu'on clique dessus*

```
document.getElementById("img1").onclick=clickImage1;
```

```
function clickImage1() {  
    document.getElementById("img1"). style.border="1px solid red";  
}
```

# Mettre en place un écouteur

## 2 )En utilisant les spécifications du DOM 2

**objet.addEventListener("event",nomFonctionEcouteur)**

Ex

```
document.getElementById("img1").addEventListener("click", clickImage1);
```

### Avantage:

- C'est la norme donc moins de risque que cela ne fonctionne plus un jour .
- Dans certains cas, on veut mettre en place plusieurs écouteurs pour un même événement. Seule cette solution nous permet cela.

Ex : on veut gérer 2 écouteurs associés à l'événement click : un qui permet de modifier la bordure et l'autre qui permet de changer l'image

```
document.getElementById("img1"). addEventListener("click", borderRouge);
```

```
document.getElementById("img1"). addEventListener("click", changerImage);
```

# Mettre en place un écouteur

- On peut facilement supprimer une gestion d'évènements

**`nœud.removeEventListener("event", nomFonction)`**

Ex : `document.getElementById("img1").removeEventListener("click", clickImage1);`

- La mise en place d'un écouteur à l'aide de **`addEventListener`** est possible sur tous les objets du DOM
- l'objet window (par exemple l'évènement 'resize' n'existe que pour window)
- l'objet document
- tous les nœud éléments du DOM
- c'est aussi possible sur les nœuds textes (mais cela a peu d'intérêt car ils ne réagissent qu'à très peu d'événements)

**Seul Inconvénient: si besoin d'une compatibilité IE8**

**Non compatible Internet Explorer <9**

# Mettre en place un écouteur



Pour mettre en place un écouteur pour une **collection d'éléments**, il faut mettre en place l'écouteur pour **CHAQUE** élément de la collection  
→ il faut OBLIGATOIREMENT utiliser une structure répétitive

exemple

```
var lesDivs=document.querySelectorAll("div") ;  
for (var i=0;i<lesDivs.length;i++) {  
    var uneDiv=lesDiv[i];  
    uneDiv.addEventListener("click", mettreEnCouleur);  
}
```

# Mettre en place un écouteur

Si l'instruction "Quand Anne-Marie Puizillout lève sa main gauche, tous les étudiants de l'amphi frappent une fois dans leur main." est donnée alors que l'amphi est vide

**Que se passera-t-il lorsque, plus tard ,une fois les étudiants installés dans l'amphi, AM Puizillout lèvera la main gauche ?**

→ aucun étudiant ne réagira car la mise en place de l'écouteur ne s'est pas faite

De la même manière, **il faut mettre en place les écouteurs sur les éléments du DOM uniquement lorsque ces éléments existent !!**

**→ Il faut attendre la diffusion de l'événement "load" sur l'objet window pour mettre en place les écouteurs sur les autres éléments du dom**

# A retenir

Toute instruction se trouvant en dehors de toute fonction va s'exécuter au moment où le navigateur charge le fichier javascript (c'est à dire lorsqu'il analyse le fichier script suite à son chargement à l'aide de la balise script.)

Toute instruction se trouvant dans une fonction est exécutée lors de l'appel de la fonction. Elle peut donc ne jamais être exécutée.

On ne peut accéder aux éléments du DOM que lorsqu'ils sont créés i.e lorsque le dom est chargé i.e après la diffusion de l'événement load auprès de l'objet window. Tout script commencera donc par l'instruction :

```
window.addEventListener("load",nomFonction)
```

# Mettre en place un écouteur

```
<html>
<head>
<script src="code.js"></script>
</head>
<body>
  <div id="div1"></div>
  <div id="div2"></div>
</body>
</html>
```

**// Seul les éléments du DOM : window ,document , html, head et script existent → ceux présents à l'endroit du chargement du script**

**window.addEventListener("load" , initialiser);**

```
function initialiser(evt) {
```

**// Tous les éléments du DOM ont été créés**

```
  document.getElementById("div1").addEventListener("click", doA1);
```

```
  document.getElementById("div2").addEventListener("click", doA2);
```

```
}
```

```
function doA1(evt) {
```

```
  .....
```

```
}
```

```
function doA2(evt) {
```

```
  .....
```

```
}
```

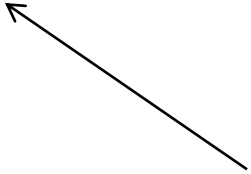
# Modifier les éléments du DOM



# Modifier les éléments du DOM

Toutes les transformations à apporter à la page lors de la survenue d'un événement utilisateur ou d'un événement système seront programmés dans une **fonction écouteur**

```
function nomFonctionEcouteur(evt) {  
    .....  
}
```



Nous reviendrons ultérieurement sur ce paramètre transmis par le navigateur à toute fonction écouteur. C'est le seul paramètre possible !

# L'API DOM: Modifier le contenu du DOM

## Modifier le contenu de la page

L'api dom nous permet d'accéder et/ou de modifier les propriétés d'un nœud

var variableX= nœud.propriété ; → pour récupérer la valeur d'une propriété  
nœud.propriété = valeur ; → pour mettre à jour la propriété d'un élément

Ex : Lorsqu'on clique sur le bouton changer l'image, l'image affichée est modifiée

```

```

```
<input id="btn" type="button" value="changer l'image" />
```

```
window.addEventListener("load", initialiser) ;
```

```
function initialiser (evt) {
```

```
    var bouton=document.getElementById("btn");
```

```
    bouton.addEventListener("click", changer);
```

```
}
```

```
function changer (evt) {
```

```
    var img=document.getElementById("img1");
```

```
    img.src="photo2.jpg";
```

```
}
```

Bien sur, avant de modifier un élément, il faut d'abord y accéder !

# L'API DOM: Modifier le contenu du DOM

## Modifier le contenu de la page

L'api dom nous permet d'accéder et/ou de modifier les attributs d'un nœud

`var variableX= noeud.getAttribute("nom attribut")` → pour récupérer la valeur d'un attribut

`noeud.setAttribute("nom attribut", " nouvelle valeur"` ; → pour mettre à jour un attribut d'un élément → si on modifie la valeur d'un attribut,

Lorsqu'on modifier une attribut, automatiquement le navigateur **modifie la propriété associée**.

Ex : Même exemple que sur la diapo précédente mais en modifiant l'attribut src et non plus la propriété src

```
function changer (evt) {  
    var img=document.getElementById("img1");  
    img.setAttribute("src","photo2.jpg");  
}
```

Cela ne change rien au résultat visualisé par l'internaute , dans les 2 cas , l'image est changée.

# L'API DOM : Modifier le DOM



Attention pour certains d'attributs, la valeur obtenue n'est pas la même si on utilise la méthode `getAttribute()` ou la propriété de l'objet

- **Pour les attributs dont la valeur est un chemin**

Ex: ``

`document.getElementById("img1").src` retournera le **chemin absolu** du fichier image soit `http://www.monsite.com/images/toto.jpg`

`document.getElementById("img1").getAttribute("src")` retournera le **chemin indiqué dans l'attribut src du code html** soit le chemin relatif `"images/toto.jpg"`

# L'API DOM : Modifier le DOM



## Pour les attributs d'état (ex checked, selected, ..)

Ex: `<input id="i1" type="checkbox" checked="checked" />`

`document.getElementById("i1").getAttribute("checked")` retournera toujours la valeur de l'attribut dans votre code HTML soit "checked" et cela même si la case a été décochée par l'internaute entre temps

`document.getElementById("i1").checked` retournera une valeur booléenne indiquant si la case est cochée ou non. Si l'internaute coche la case, la propriété `checked` a pour valeur `true`, si il décoche la case elle a pour valeur `false`.

# L'API DOM : Modifier le DOM

## Des attributs particuliers : les attributs data-

Souvent, nous avons besoin de définir des informations utiles à notre code javascript, dans la page HTML. La liste des attributs possibles étant fixés par le W3C, cela ne semblait pas possible. Depuis HTML5, le W3C a permis de définir **des attributs sur mesure** à condition qu'il commence par **data-**. La page reste valide W3C.

Ex `<input type="button" data-video="video1.mp4" id="i1"/>`

## Comment manipuler ces attributs data- en javascript ?

on utilise la propriété **dataset** de l'élément suivi du nom de l'attribut donné après le data

Ex : `var urlVideo=document.getElementById("i1").dataset.video;`

# L'API DOM : Modifier le DOM



Pour modifier les propriétés ou la mise en forme d'une **collection d'éléments**, il faut modifier **CHAQUE** élément de la collection → il faut OBLIGATOIREMENT utiliser une structure répétitive

exemple

```
var lesDivs=document.querySelectorAll("div") ;  
for (var i=0;i<lesDivs.length;i++) {  
    var uneDiv=lesDiv[i];  
    uneDiv.classList.add("rouge");  
    uneDiv.title="Vous survolez actuellement une div ....";  
}
```

# L'API DOM : Modifier le DOM

## Ajouter des nœuds dans le DOM

L'API DOM dispose de méthodes permettant d'ajouter des nœuds dans le dom.

*document.createElement('nom balise')* : création d' un nœud élément

*document.createTextNode('texte')* : crée un nœud texte

*noeudParent.appendChild(nœud à ajouter)* : permet de le ranger dans l'arbre comme dernier nœud enfant du nœud parent

*noeudParent.insertBefore(nœud à ajouter, noeud1)* : permet de le ranger comme noeud enfant du nœud parent, juste avant le frère noeud1

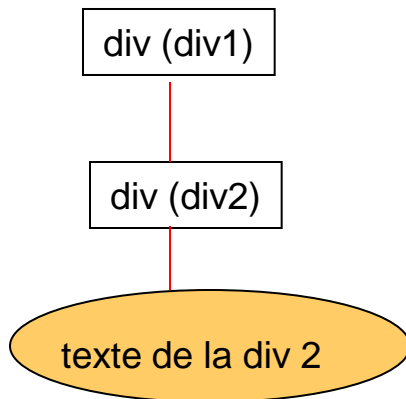


# L'API DOM : Modifier le DOM

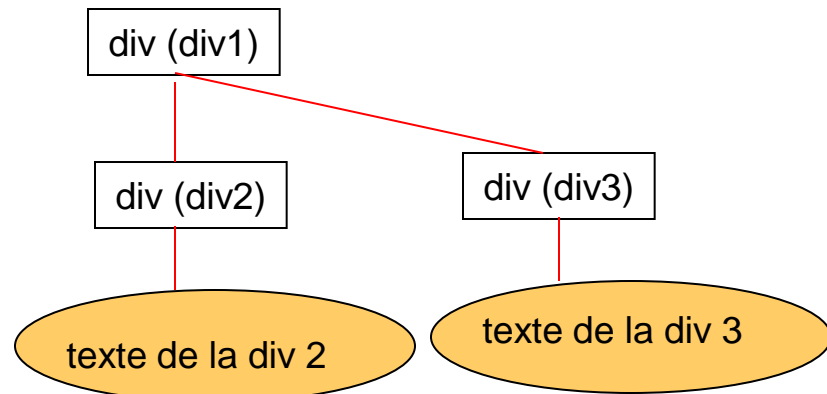
## Exemple :

```
var nouvDiv= document.createElement("div");  
nouvDiv.id="div3";  
var texte= document.createTextNode("texte de la div 3");  
nouvDiv.appendChild(texte)  
document.getElementById("div1").appendChild(nouvDiv)
```

### Avant



### Après



# L'API DOM : Modifier le DOM

Ajouter du code HTML : la navigateur transforme ce code html en éléments du DOM

*noeudParent*.innerHTML=code HTML : on remplace le contenu d'un élément par le code HTML fourni.

```
var div= document.getElementById("div1")
```

```
div.innerHTML="<p> Voici un nouveau  paragraphe </p>";
```

*noeud.insertAdjacentHTML*("où l'ajouter", code HTML) : permet d'insérer le code HTML à l'intérieur ou autour du nœud choisi.

"où l'ajouter" peut prendre 4 valeurs :

**"beforeend"** : le code HTML est ajouté après le dernier enfant

**"afterbegin"** : le code HTML est ajouté avant le premier enfant

**"afterend"** : le code HTML est ajouté après l'élément choisi

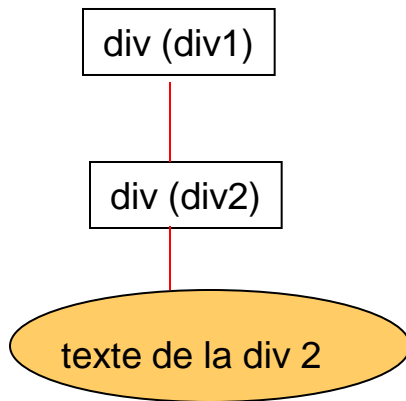
**"beforebegin"** : le code HTML est ajouté avant l'élément choisi

# L'API DOM : Modifier le DOM

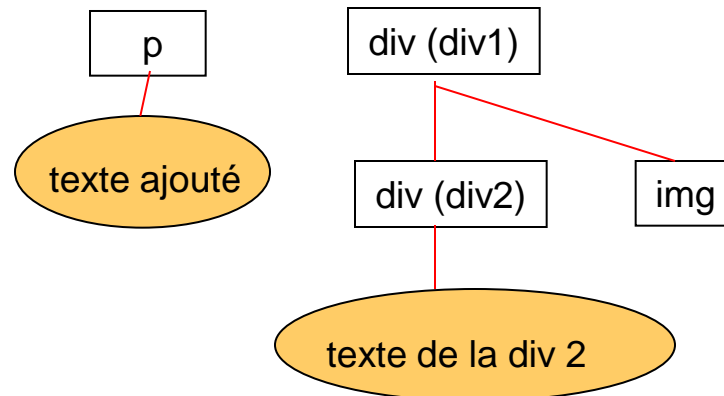
## Exemple :

```
var div= document.getElementById("div1");  
div.insertAdjacentHTML("beforeend","<img src='img1' />");  
div.insertAdjacentHTML("beforebegin","<p>Texte ajouté </p>");
```

### Avant



### Après



# L'API DOM : Modifier le DOM

On peut aussi **cloner** un nœud élément (avec ou non ses descendants).

`noeudACloner.cloneNode(avecDescendant)` : retourne un nœud identique au nœud cloné (avec tous ses descendants si avecDescendant = true, juste le nœud sinon).

On peut **supprimer** un nœud

`noeudParent.removeChild(noeudASupprimer)`

Ou dans les dernières versions:

`nœud.remove()`

On peut le **déplacer** dans la hiérarchie:

`noeudParent.appendChild(noeudADéplacer)`

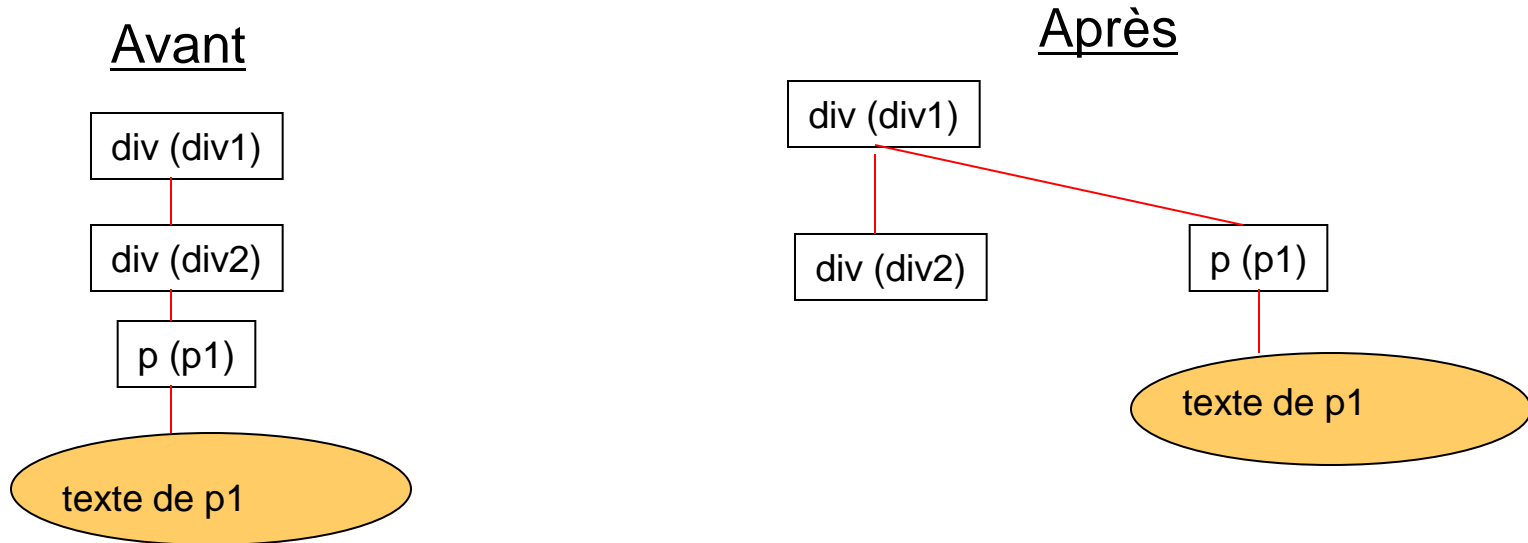
ou `noeudParent.insertBefore(noeudADeplacer,noeud1,)`

Un nœud ne pouvant avoir qu'un seul parent, le fait de le rattacher à un nouveau nœud parent, élimine sa relation à l'ancien nœud parent

# L'API DOM : Modifier le DOM

## Exemple :

```
var eltP1= document.getElementById("p1");  
document.getElementById("div1").appendChild(eltP1) ;
```



On déplace le nœud avec toute sa descendance. De même si on supprime un nœud , on supprime le nœud et toute sa descendance.

# L'API DOM: Modifier la mise en forme

## Modifier la mise en forme d'éléments dans la page

1 ) Des classes associées à des mises en forme précises ont été définies dans la css

### On peut utiliser la notion de class

En HTML5 , on utilise la propriété `classList` d'un élément

`noeud.classList.add('classe');` → ajouter une classe à l'élément

`noeud.classList.remove('classe');` → supprime une classe à l'élément

`noeud.classList.toggle('classe');` → ajoute la classe si l'élément ne l'a pas, la supprime sinon

`noeud.classList.contains('classe');` → renvoie true si l'élément a la classe demandée, false sinon

ex :

```
var img1 = document.getElementById("img1");
```

```
img1.classList.add("rouge")
```



```
<img id="img1" .. class="rouge" />
```

dans la css : **.rouge** { ..... }

# L'API DOM: les propriétés de style (css)

**pour récupérer / modifier les propriétés de style d'un élément → on utilise **la propriété (ou l'attribut) style** dont dispose tout élément**

`var valeurProp = nœud.style.nomProprieteCss` → récupère la valeur d'une propriété css définie via la propriété style

`nœud.style.nomProprieteCss= valeur;` → met à jour la valeur d'une propriété css de style



le nom de la propriété css n'est pas identique à celui de la feuille de style : **le tiret est remplacé par une majuscule.**

Ex background-color devient backgroundColor

font-size devient fontSize

Ex: voir code écouteur "mouseover" de exemple2.htm

# L'API DOM: les propriétés de style (css)

## Rappel

Lorsqu'on utilise **la propriété (ou l'attribut) style** les mises en forme ainsi définies ont alors **une priorité de 1000** → le style d'un élément modifié via javascript est toujours prioritaire aux sélecteurs définis pour cet élément dans la css **SAUF** si la règle css contient la valeur **!important;**

## Exemple :

```

```

```
img { ← priorité 1      #img1 { ← priorité 100  
    width:50px;          width:200px;  
}                        }
```

```
document.getElementById("img1").style.width="400px" ← priorité 1000
```

L'image aura donc une largeur de **400px**



# L'API DOM: les propriétés de style (css)



L'attribut style fait référence à la valeur de l'attribut style défini dans l'élément HTML (ou via javascript) , **il ne tient pas compte des propriétés de style mise en place par la feuille de style.**

Si l'on veut pouvoir obtenir la propriété de style "globale" (html +css) il faut utiliser :

**`window.getComputedStyle(element, null);`**

Exemple : `recuperationStyle.htm`

Pour IE < 9 :

**`element.currentStyle //IE`**

# Le mot clé this

Dans un certain nombre de cas, notamment lorsqu'on souhaite associer un comportement identique à plusieurs éléments html, il est souvent nécessaire de pouvoir faire référence **à l'élément répondant à l'événement** .

C'est le but du mot clé **this**.

Ex :

```
document.getElementById("para1").addEventListener("click",quiEstClique);  
document.getElementById("para2").addEventListener("click",quiEstClique);
```

```
.....  
function quiEstClique () {  
  alert (this.id);  
  alert(this.innerHTML);  
}
```

```
page html      <p id="para1"> paragraphe 1 </p>  
                  <p id="para2">paragraphe 2</p>
```

Si on clique sur

- le premier élément p , il s'affichera para1 puis paragraphe1
- sur le 2eme il s'affichera para2 puis paragraphe2

# L'appel de fonction écouteur

Ecrire :

`window.addEventListener("load",initialiser)`

ou `window.addEventListener("load",initialiser())` ne veut pas du tout dire la même chose.

**cas 1:** `window.addEventListener("load",initialiser)`

On donne le nom (et donc le code) de la fonction à exécuter lorsque l'événement load sera transmis à l'objet window.

**C'est donc le moteur javascript du navigateur qui se chargera de l'exécution effective de la fonction** (de manière schématique, c'est lui qui ajoute les parenthèses pour demander l'exécution)

# L'appel de fonction écouteur

**cas 2** : `window.addEventListener("load",initialiser())`

La fonction initialiser est donc **exécutée immédiatement(on a défini les parenthèses)** et le résultat de cette fonction correspondra à la fonction écouteur associée à load. C'est possible, mais dans ce cas, le résultat de initialiser doit être de type Function.

**Concrètement, vous ne serez pas amené à utiliser cette syntaxe cette année, cela fait partie des techniques de javascript avancé.**



Dans une méthode **addEventListener**, on ne met **jamais de parenthèses** après le nom de la fonction écouteur

# L'appel de fonction

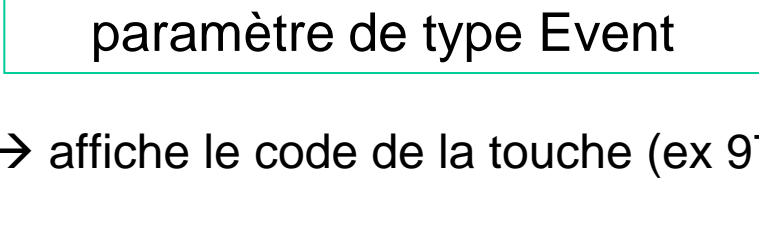
## Quels sont les paramètres transmis aux fonctions écouteur ?

Il y a un seul paramètre transmis aux fonctions "ecouteurs", c'est un objet de type **Event**.

Cet objet contient un certain nombre d'informations sur l'événement. Par exemple, la position de la souris lorsque le click a eu lieu, l'élément html cible de l'événement, etc.

```
document.getElementById("input1").addEventListener("keypress",taperTouche);  
window.addEventListener("click",cliquer);
```

```
function taperTouche(evt) {  
    console.log(evt.which) ; → affiche le code de la touche (ex 97 pour 'a')  
}  
function cliquer(evt) {  
    console.log(evt.clientX) ; → donne l'abscisse de la position de la souris au  
    moment du click  
}
```



paramètre de type Event

# L'appel de fonction

## Puis-je transmettre des paramètres à une fonction écouteur?

C'est possible mais cela nécessite l'utilisation de techniques avancées de javascript (**closure**)

Ce n'est pas au programme de la première année.

Donc pour cette année, le seul paramètre reçu par les fonctions écouteurs est le paramètre [evt](#). Ce paramètre est transmis par la navigateur lorsque l'action a lieu.

# L'objet Event

Nom de la propriété	description	I.E<9
bubbles	indique si l'événement bouillonne (est diffusé à toute l'arborescence)	inexistant
cancelable	indique si l'événement est annulable	inexistant
target	indique le nœud cible de l'évènement	srcElement
currentTarget	indique le nœud actuel réagissant à l'évènement	inexistant
type	indique le type d'événement (click, mouseover,...)	inexistant
timestamp	indique la date et l'heure de l'évènement	timestamp
which	Code de la touche frappée	keyCode
clientX, clientY	Position de la souris	clientX,clientY
Nom de la méthode		
stopPropagation()	arrête la propagation	cancelBubble=true
preventDefault()	annule l'événement. Cela empêche le navigateur de procéder à l'action par défaut pour l'événement (par ex, activer l'url d'un lien, soumettre un formulaire)	returnValue=false

# Ordre d'exécution

## Exemple :

```
<html>
  <head>
    <script type="text/javascript" src="ex.js"></script>
  </head>
  <body>
    <div>
      
    </div>
  </body>
</html>
```

```
window.addEventListener("load",initialiser);
```

1

```
function initialiser() {
```

2

```
  document.getElementById("img1").addEventListener("mouseover",afficherBord);
```

3

```
  document.getElementById("img1").addEventListener("mouseout", cacherBord);
}
```

```
function afficherBord(evt) {
```

4

```
  document.getElementById("img1").style.border="1px solid red";
}
```

```
function cacherBord(evt) {
```

5

```
  document.getElementById("img1").style.border="none";
}
```



# Ordre d'exécution

## Déroulement :

**Au temps t0 :** l'internaute tape l'url de la page, le navigateur reçoit la page et commence à l'analyser

**au temps t1 :** il analyse `<script....src="ex.js">` , il charge le fichier et exécute les instructions hors fonctions donc ici l'instruction **1** . Il enregistre un écouteur permettant d' exécuter la fonction *initialiser* lorsque l'événement load sera diffusé

**au temps t2 :** la page est entièrement analysée et chargée, le DOM est créé. Le navigateur diffuse l'événement load qu'intercepte la fenêtre et donc elle exécute la fonction *initialiser* ( **2** et **3** ). Dans cette fonction, on met en place l'interactivité sur l'image 1 : on enregistre un ecouteur permettant d'exécuter la fonction *afficherBord* lorsqu'un événement mouseover sera diffusé sur l'image1 (lorsque l'internaute passe sa souris). Un autre écouteur est défini sur l'émission de l'événement qui permettra l'exécution de la fonction *cacherBord*.

**au temps t3 :** L'internaute passe la souris sur l'image1 , l'événement mouseover est envoyé à l'image 1 et la fonction *afficherBord* ( **4** ) est exécutée. On modifie la propriété de style border de l'image 1. Les ascendants de image1 sont aussi avertis de l'événement mais ils ne l'écoutent pas donc rien ne se produit.

# Ordre d'exécution

**au temps t4 :** L'internaute enlève la souris de l'image1, l'événement mouseout est diffusé à l'image 1 et , la fonction *cacheBord* ( 5 ) est exécutée. On modifie la propriété de style border de l'image 1. Les ascendants de image1 sont aussi avertis de l'événement mais ils ne l'écoutent pas donc rien ne se produit.

**au temps t5 :** L'internaute passe la souris sur l'image1 , l'événement mouseover est diffusé à l'image 1 et la fonction *afficherBord* ( 4 ) est exécutée. On modifie la propriété de style border de l'image 1. Les ascendants de image1 sont aussi avertis de l'événement mais ils ne l'écoutent pas donc rien ne se produit.

**au temps t6 :** L'internaute tape une autre url, la page est déchargée. L'événement "unload" est diffusée à la fenêtre mais elle ne l'écoute pas donc rien ne se produit.

En programmation événementielle, le code s'exécute à des moments différents.

Lorsqu'on associe une fonction à la diffusion possible d'un événement pour un élément donné, la fonction n'est pas exécutée lors de l'association mais lorsque l'événement est réellement diffusé.

# La notion de propagation d'événement

La gestion des événements est un phénomène plus compliqué qu'il en a l'air au premier abord

Soit le code suivant associé à la css suivante

...

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

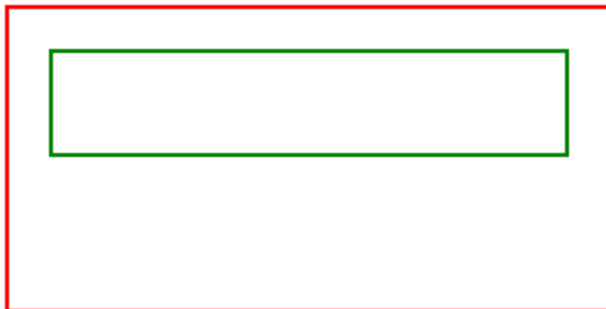
```
  <div id="div2">
```

```
    </div>
```

```
</div>
```

```
#div1 { height:150px; border:2px solid red;}
```

```
#div2 { height:50px; border:2px solid green; margin:20px;}
```



Lorsque je clique à l'intérieur de la boîte verte est-ce que je clique aussi à l'intérieur de la boîte rouge ?

# La notion de propagation

```
<html>
```

```
.....
```

```
<body>
```

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

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

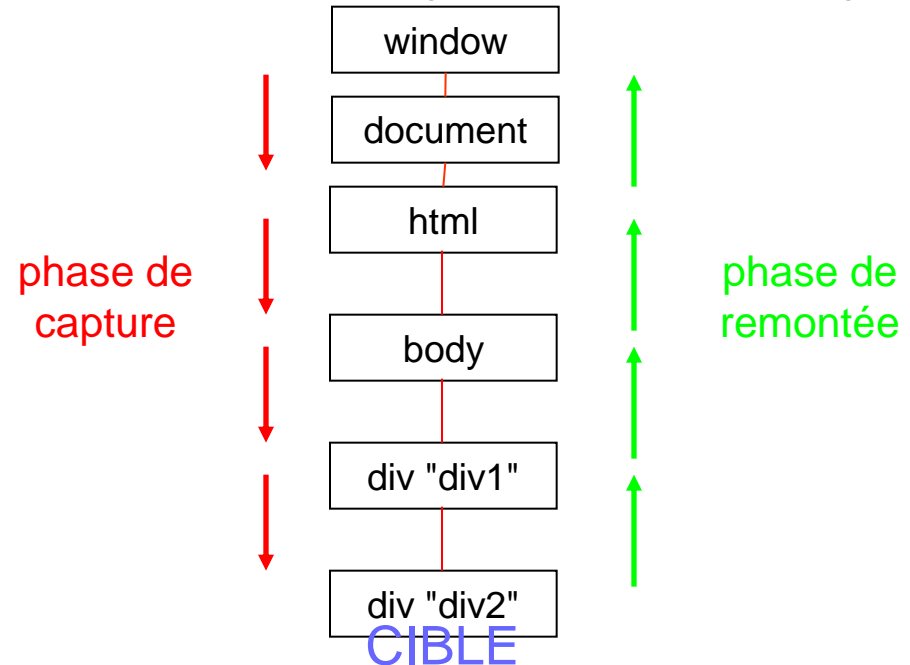
```
      </div>
```

```
    </div>
```

```
  </body>
```

```
</html>
```

Si on clique sur l'image, l'événement se propage ainsi



La **cible** est toujours l'élément en avant-plan (donc ici div 2)

Il va donc y avoir 2 phases distinctes lorsqu'un événement se produit :

**phase de capture** : tous les éléments en lien avec la cible (de window jusqu'à la cible) sont prévenus qu'un événement de type click vient d'avoir lieu

**phase de remontée** : Dans cette phase on repart de la cible et on remonte les ancêtres ayant un lien avec la cible. A nouveau chacun est averti qu'un événement a eu lieu

# La notion de propagation

Cette propagation existe systématiquement, malgré tout :

- Généralement, on intervient pendant une seule de ces phases
- Bien souvent, seule la cible réagira à l'événement

Par défaut, lorsqu'on utilise `addEventListener` ou les attributs `"onaction"`, on se place dans la phase de remontée

Pour intercepter un événement pendant la phase de capture, il faut fournir **un troisième paramètre** à `addEventListener`  
`objet.addEventListener("action",fonction,true)`

# La notion de propagation

Si on veut gérer l'évènement à la fois pendant sa phase de capture ET pendant sa phase de remontée → il faut mettre en place 2 écouteurs.

- **un pour la phase de capture**

```
objet.addEventListener("evenement",fctEcouteur1,true);
```

- **un pour la phase de remontée**

```
objet.addEventListener("evenement",fctEcouteur2,false);
```

# La notion de propagation



Si vous souhaitez supprimer un écouteur via la méthode **removeEventListener**, les paramètres de cette méthode doivent être à l'identique de ceux fournis pour le addEventListener **y compris la phase de capture**

Exemple :

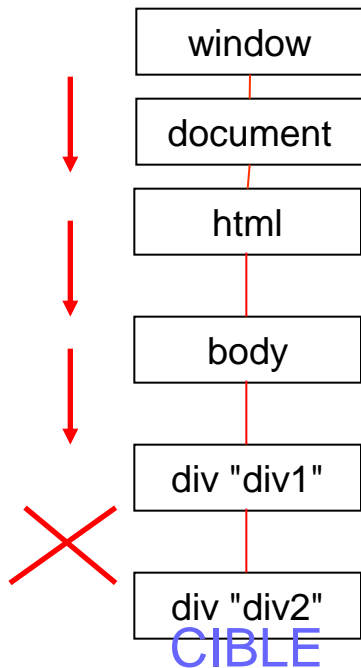
```
var div1=document.getElementById("div1");  
div1.addEventListener("click",fctEcouteur,true);
```

Pour le supprimer, il faudra écrire :

```
div1.removeEventListener("click",fctEcouteur,true);
```

# La notion de propagation

Par l'intermédiaire de l'objet Event, on peut par contre empêcher cette propagation :



On arrête la propagation lors de la phase de capture sur l'objet div1 :

```
document.getElementById("div1").addEventListener ("click",stop,true)
```

```
function stop(evt) {  
    evt.stopPropagation();  
}
```

→ l'élément div2 bien que cible du click ne reçoit jamais l'événement



# La notion de propagation

**Il existe un autre type de propagation, ce que j'appelle la propagation "native"**

Certains éléments sont naturellement interactifs :

les liens, les boutons radios, les cases à cocher , l'élément form des formulaires, etc.

Lorsqu'on met en place une écoute de cet évènement (ex click) , notre code s'exécute **avant** le code exécuté par le navigateur.

Il est alors possible d'empêcher son exécution en faisant appel à la méthode **preventDefault() de l'objet Event**

# La notion de propagation

## Exemple :

```
<input type="text" />
```

```
document.querySelector("textarea").addEventListener("keypress",  
lipogramme)
```

```
function lipogramme(evt) {  
    if (evt.which==101) { //code de la touche "e"  
        evt.preventDefault();  
    }  
}
```

→ le caractère "e" aura beau être tapé par l'internaute, il n'apparaîtra jamais dans la zone de texte.

# Les bases du langage

## Les commentaires :

// ou /\* \*/

## Les variables

- En javascript sont reconnus 5 types de données :

- **nombre** (entier ou réel) (number)
- **chaîne de caractère** (string)
- **booléen** (boolean)
- **objet** (object)
- **undefined**: lorsque la variable n'est pas initialisée

- **Pas de déclaration de type en javascript.**

Le type est directement attribué par le langage en fonction du contenu de la variable. La variable n'a pas de type fixe, il évolue en fonction du contenu.

- Les variables doivent malgré tout être déclarées à l'aide de **var nomVariable;**

# Les bases du langage

Même règle de nommage qu'en java (par contre pas de caractère accentué).

JavaScript est aussi **sensible à la casse** : var1 et Var1 sont 2 variables différentes.



La **déclaration de variable** en javascript peut être **implicite**, mais dans ce cas vous prenez des risques : si vous orthographiez mal les variables, à chaque nouvelle orthographe, une nouvelle variable sera créée. → commencer ces scripts par l'instruction "use script";

# Les bases du langage

Initialisation de variable :

variable =valeur;

Exemples :

var nom="Puizillout";  
var nom='Puizillout';

" ou ' indique tous deux une chaîne de caractères

var nb=0;

var nbReel=0.0;

var coursJavaScript=true;

# Les bases du langage

**Rque :** Le type de la variable étant défini de manière automatique au cours de l'exécution il peut parfois être intéressant de le connaître :

Pour cela vous pouvez utiliser la fonction `typeof` :

Ex

- `var chaine='moi' ;`  
`typeof(chaine)` a pour résultat `String`
- `chaine=0 ;`  
`typeof(chaine)` a pour résultat `Number`
- `chaine=0.0;`  
`typeof(chaine)` a pour résultat `Number`
- `chaine=false;`  
`typeof(chaine)` a pour résultat `Boolean`

# Les bases du langage

JavaScript se charge lui-même des conversions de type.

Malgré tout, 2 fonctions permettent de lui indiquer explicitement cette conversion :  
`parseInt(String,base)` convertit une chaîne en entier (système hexadécimal).

*base indique dans quelle base est exprimée la chaîne à transformer (si elle est omise, il considère que la valeur est en base 10)*

ex `parseInt('12abc',10)` a pour résultat 12

```
var chaine1 = "123" ;
```

```
var chaine2 = "456" ;
```

```
var chaine3 = chaine1 + chaine2;
```



```
chaine3="123456" →
```

```
var chaine4 = parseInt(chaine1,10) + parseInt(chaine2,10)
```

```
chaine4=579
```

```
var chaine5= parseInt("111",2)
```

```
chaine5=7
```

`parseFloat(String,base)` converti une chaîne en float dans une base.

ex `parseFloat('12.8abc',10)` a pour résultat 12.8

`parseFloat("abc",10)` a pour résultat NaN (not a number)

# Les bases du langage

A l'inverse, on peut aussi transformer un **nombre en String** à l'aide de la méthode **toString()**:

Ex:

```
var nb1 =123 ;
```

```
var nb2 = 456 ;
```

```
var chaine3 = nb1 + nb2;           ↔           chaine3=579; (typeof Number)
```

```
var chaine4 = nb1.toString() +nb2 ↔ chaine4="123456"
```



# Les bases du langage

## La portée des variables

- Une variable déclarée dans une fonction est utilisable uniquement dans la fonction et inconnue hors de la fonction.  
Elle existe uniquement le temps de l'exécution de la fonction et est supprimée à la fin de la fonction
- Une variable déclarée en-dehors d'une fonction est accessible par toutes les fonctions javascript liées à la page → c'est une **variable globale**.  
Elle est créée lors du chargement du script et disparaît lors du déchargement de la page  
Toute variable globale devient une propriété de l'objet window



une variable non déclarée est systématiquement considérée comme étant une variable globale.

# Les bases du langage

## Exemple

```
var nb1;
```

nb1 est déclarée en dehors de toute fonction , nb1 est une variable globale

```
function f1 () {
```

```
  var nb2=3;
```

f1 a accès aux variables nb1 et nb2

```
}
```

```
function f2(nb3) {
```

```
  var nb4;
```

f2 a accès aux variables nb1,nb3 et nb4

```
}
```



Afin de rapidement identifier les variables globales, il est préférable de les déclarer en tout début du fichier de script. Il faut limiter au maximum, l'utilisation de variables globales.

# Les bases du langage

## Les opérateurs : idem java

- arithmétique : + , - , \* , / , %
- affectation : = , += , -= , \*= , /= , % =
- incrémentement : ++ , -- (idem java : ++nb1 ou nb1++ : se reporter au cours java pour voir la différence)
- comparaison : == , != , === , !== , > , < , <= , >=
  - == : compare les 2 variables en essayant de convertir le type  
Ex 0=="0" a pour résultat vrai
  - === : compare la valeur **et** le type  
Ex 0=== "0" a pour résultat faux
- logiques:
  - non : ! ,
  - et : && ,
  - ou : ||
- ternaire : condition ? val1 : val2

# Les bases du langage

## Les instructions conditionnelles :

- **si.. sinon :**

```
if (condition) {  
    bloc instructions;  
}  
else {  
    bloc instructions;  
}
```

- **instruction selon :**

```
switch (variable) {  
    case valeur1 :  
        bloc 1;  
        break;  
    case valeur2 : case valeur3:  
        bloc2;  
        break;  
    default : bloc default;  
}
```

# Les bases du langage

## Les instructions répétitives :

- **tant que .... fin tant que:**

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

- **répéter.... tant que**

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

- **pour compteur allant de debut à fin par pas de 1**

```
for (var compteur=debut; compteur<= fin; compteur++) {  
    instructions;  
}
```

**Comme en java :**

- break pour arrêter la boucle
- continue pour passer un tour

# Les bases du langage

## Déclarations de fonctions :

```
function nomDeLaFonction (param1,param2,...) {  
    instructions;  
}
```

**Si la fonction calcule un résultat :**

```
function nomFonction (param1,param2) {  
    instructions;  
    return valeur ou variable  
}
```

## Appel d'une fonction

```
nomFonction(valeur param1,valeur param2);  
ou  
variable=nomFonction(valeur param1,valeur param2);
```

# Les bases du langage



Un paramètre ne doit jamais être déclaré à l'aide de var dans l'entête de la fonction.

```
function fonction1(var param1) {  
}
```

# Les bases du langage

## Exemple

```
function calculerPuissance (nombre,puissance) {  
    var resultat=1;  
  
    for (var i=1;i<=puissance;i++) {  
        resultat=resultat*nombre;  
    }  
    return resultat;  
}
```

Pour l'appeler :

```
var nb1=calculerPuissance (2,3); // calcule 23
```

↑  
↓  
nb1 = 8;



# Les bases du langage

## La notion de paramètres optionnels

En javascript , on peut , lors de l'appel d'une fonction, donner moins de valeurs que le nombre de paramètres demandé par la signature de la fonction. C'est généralement ce qu'on appelle des paramètres optionnels.

Ex :

```
function calculerPuissance(nombre,puissance) {  
    if (arguments.length==1) { // permet de savoir combien il y a de valeurs en paramètres  
        puissance=1;  
    }  
    var resultat=1;  
  
    for (var i=1;i<=puissance;i++) {  
        resultat=resultat*nombre;  
    }  
    return resultat;  
}
```

calculerPuissance(3) aura pour résultat 3 , calculerPuissance(3,2) aura pour résultat 9



**arguments** est un tableau où sont stockées toutes les valeurs passées en paramètre

# Les bases du langage

## Les tableaux:

on utilise un objet intégré du langage : Array.

### 3 possibilités pour l'initialiser :

- créer un tableau sans spécifier de taille (s'agrandira au fur et à mesure)  
`var tableau = new Array();`
- créer un tableau en spécifiant sa taille i.e le nombre de cases(s'agrandira si besoin)  
`var tableau = new Array(taille);`
- créer un tableau en indiquant le contenu de chacune des cases  
`var tableau = new Array(valeur1,valeur2,valeur3,..);`  
ou `var tableau = [valeur1,valeur2,valeur3];`

Une fois défini, vous pouvez mettre n'importe quel type de données dans les cases et vous pouvez même mélanger les types de données.

### Ex

```
var tableau = new Array(true,10,"a","cd");
```

# Les bases du langage

## Pour obtenir la taille du tableau

nomDuTableau.length

ex: var tableau=new Array(1,2);

tableau.length vaut 2

## Pour accéder à une case :

nomDuTableau[noCase] avec noCase variant de 0 à taille-1;

ex :

var tableau=new Array(1,2);

tableau[0] = 3;

var nb1=tableau[1];

tableau



nb1



## Pour parcourir toutes les cases du tableau :

var taille=tableau.length;

for (var i=0;i<taille; i++ ) {

    traitement sur tableau[i];

}

# Les bases du langage

Raccourci pour passer en revue tous les n° de case d'un tableau

```
for (variable in tableau) {  
    instructions  
}
```

Ex :

```
var nb;  
var chaine="";  
var tableau = new Array("a","b","c",);  
for (nb in tableau) {  
    chaine=chaine+tableau[nb];  
}
```

nb

2

chaine

abc

tableau

a	b	c
---	---	---

# Les objets prédéfinis de javascript

Il existe un certain nombre d'objets prédéfinis dans le langage disposant de propriétés et de méthodes facilitant la tâche du programmeur :

En voici quelques uns:

**Screen** : représente l'écran de l'internaute et nous permet d'avoir des informations sur les caractéristiques de l'écran de l'internaute

**Date** : permet de manipuler plus facilement les dates (notamment pour ajouter ou soustraire 2 dates)

**RegExp** : permet notamment de vérifier le format d'une saisie

**Navigator** : correspond au navigateur qui tourne

**String** : permet de manipuler les chaînes de caractère

**Array** : permet de manipuler les tableaux

# Les objets prédéfinis de javascript

**Image** : permet de manipuler des images notamment pour les pré-chargés.

**Math** : contient les fonctions mathématiques (cos, sin, random)...

**Global** : contient des fonctions générales (parseInt, eval,...)

Vous retrouverez à la fin de ce cours , un récapitulatif des propriétés et méthodes de chacun de ces objets.

# Les objets prédéfinis : Exemple Date

Cet objet permet de travailler avec les dates et les heures dans vos scripts.

Les objets Date ont une précision de l'ordre de la milliseconde et sont calculés à partir du 1er Janvier 1970.

Cet objet dispose d'un grand nombre de méthode permettant

- de lire
- d'écrire
- de manipuler des dates.

# Les objets prédéfinis : Exemple Date

Pour créer un objet Date : 4 façons

- `var uneDate=new Date();` // initialise la variable uneDate avec la date et l'heure de //votre ordinateur (au moment de la création)
- `var uneDate=new Date(nombre);` // nombre est numérique et représente le nb de // millisecondes s'étant écoulé depuis le 1er Janvier 1970.

Ex `var date1= new Date(30000510255);`

date1 correspond au 14/12/1970 6h28mn30s

- `var uneDate= new Date(année,mois,jour [,heures [,minutes [,seconde [,millisecondes] ] ] ] );` **ATTENTION: mois varie de 0 à 11 et non de 1 à 12**

Ex `date1=new Date(2008,3,3)` correspond au 03 Avril 2008

Rque : les variables écrites entre [ ] sont des paramètres optionnels



# Les objets prédéfinis : Exemple Date

Et enfin

- `var uneDate=new Date(aaaa/mm/jj);` // initialise la variable uneDate avec la date passée en paramètre
- Ex : `var uneDate= new Date("2004/11/02");` correspond au 02 Novembre 2004



**Il** que soit la manière de créer la date, la date créée est toujours valide :

Ex : `uneDate=new Date("2004/12/53")` donne la date du 22 Janvier 2005

`uneDate = new Date(2004,25,3)` donne la date du 3 Février 2006

`uneDate = new Date (2004,12,1)` donne la date du 1 Janvier 2005

[voir date.htm](#)

# Les objets prédéfinis : l'objet Date

```
var uneDate= new Date();
```

Méthode	Description
uneDate.getFullYear()	retourne l'année stockée dans uneDate sous la forme d'un nombre sur 4 chiffres
uneDate.getMonth()	retourne un entier compris entre 0 (janvier) et 11 (décembre)
uneDate.getDate()	retourne un entier compris entre 1 et 31 correspondant au jour
uneDate.getDay()	retourne un entier correspondant au jour de la semaine : 0 pour Dimanche, 1 pour Lundi, .... 6 pour Samedi.
uneDate.getHours()	retourne un entier compris entre 0 et 23 correspondant aux heures
uneDate.getMinutes()	retourne un entier compris entre 0 et 59 correspondant aux minutes
uneDate.getSeconds()	retourne un entier compris entre 0 et 59 correspondant aux secondes
uneDate.setFullYear(an[,mois [,jour]])	permet de mettre à jour la variable uneDate
uneDate.setMonth(mois[,jour])	permet de mettre à jour le mois et éventuellement le jour de uneDate
uneDate.setDate(jour)	permet de mettre à jour le jour de la variable uneDate
idem pour les heures,minutes,secondes	
uneDate.toLocaleString()	retourne une chaîne de caractère contenant la date formatée selon la valeur par défaut des paramètres régionaux de la machine.
uneDate.toUTCString()	retourne une chaîne de caractère contenant la date formatée selon le format UTC (temps universel)

# Comment afficher une boîte de dialogue

## Qu'est-ce qu'une boîte de dialogue?

Une boîte de dialogue est une fenêtre qui s'affiche au premier plan et qui permet

- d'avertir l'utilisateur
- de le confronter à un choix (oui, non)
- de lui poser une question et de récupérer sa réponse

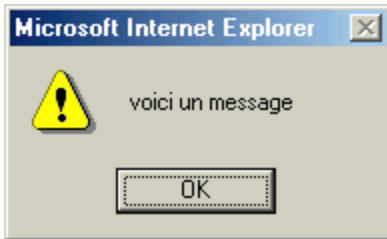
Ces 3 possibilités correspondent à 3 boîtes différentes en JavaScript.

# Comment afficher une boîte de dialogue

## Avertir l'utilisateur

La fonction: `window.alert(texte)` texte étant de type String

ex : `window.alert("voici un message");`



Pour fermer la boîte, l'utilisateur doit cliquer sur OK

# Comment afficher une boîte de dialogue

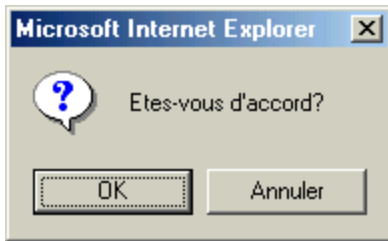
Poser une question à l'utilisateur

Ne peut répondre que par oui ou non

La fonction `window.confirm(texte)` avec texte de type String

Ex :

```
var choix;  
choix=window.confirm("Etes-vous d'accord ?");
```



Ok correspond à oui, Annuler correspond à non

Lorsque l'utilisateur appuie sur "OK" la fonction a pour résultat la valeur *true*, *false* si l'utilisateur appuie sur *annuler*.

# Comment afficher une boîte de dialogue

## Avec une réponse libre

La fonction `window.prompt(question, preRenseignerReponse)` avec question et preRenseignerReponse de type String

Ex :  
var reponse:  
reponse=window.prompt ("Voici la question" , "Saisissez votre réponse");



La variable réponse contiendra ce qu'a saisi l'utilisateur.

Si l'utilisateur a cliqué sur `Annuler`, reponse aura pour valeur `null`

# Comment ouvrir une pop-up

**window.open ("url de la fenetre", "nom de la fenetre", "paramètres optionnels")**

paramètres optionnels :

**width**=valeur largeur de la nouvelle fenêtre

**height** =valeur hauteur de la nouvelle fenêtre

**left** =valeur permet de positionner la fenêtre :abscisse du point supérieur gauche

**top** =valeur ordonné du coin supérieur gauche de la fenêtre

**toolbar**=yes/no spécifie si la fenêtre doit contenir la barre d'outils

**location**=yes/no spécifie si la fenêtre doit contenir la barre d'adresse

**status**=yes/no spécifie si la fenêtre doit contenir la barre d'état

**menubar**=yes/no spécifie si la fenêtre doit contenir la barre de menu

**scrollbar** =yes/no spécifie si la fenêtre doit contenir les barres de défilement

**resizable**=yes/no spécifie si la fenêtre peut être redimensionnée

**fullscreen**=yes/no spécifie si la fenêtre est affiché en plein écran

# Comment ouvrir une pop-up

## Exemple :

```
window.addEventListener("load",init,false);
```

```
function init(evt) {
```

```
    // la fonction ouvrirPopUp se déclenchera lorsque l'internaute cliquera sur le bouton  
    //ayant pour id btnOuvrir
```

```
        document.getElementById("btnOuvrir"). addEventListener ("click",ouvrirPopUp,false);  
    }
```

```
function ouvrirPopUp() {
```

```
    // on ouvre dans une nouvelle fenêtre la page page2.htm
```

```
    var maPopUp=window.open("page2.htm","page 2",{width:300px,height:200px});  
}
```



# Comment ouvrir une pop-up

La fenêtre principale et la pop up sont ensuite capable de dialoguer :

`window.opener` retourne la référence de la fenêtre ayant ouvert celle-ci.

voir exemple [creerPopup.htm](http://creerPopup.htm)

Attention: Lorsqu'on clique la 1ere fois sur un des boutons , il se peut que la fenêtre ne s'affiche pas ou sans la couleur de fond. Cela est dû au fait que lorsqu'on donne le focus ou que l'on change la couleur, la fenêtre n'est pas encore finie de charger.

# Comment mettre en place un timer

Il arrive régulièrement que l'on soit amené à exécuter une action de manière répétitive toutes les x millisecondes (par exemple mettre à jour l'heure ) ou à effectuer une action au bout d'un certain temps (par exemple dans un quizz, on lui laisse 1 minute pour répondre puis on passe à la question suivante).

Dans ces 2 cas là, on va utiliser un timer.

Pour exécuter UNE fonction au bout de x millisecondes :

identifiant= window.setTimeout(nomFonction, x millisecondes)

L'identifiant est unique pour chaque timer et permet de l'identifier de manière unique

ex timer1=window.setTimeout(passerQuestion,60000);

Pour arrêter le timer avant l'exécution de la fonction

window.clearTimeout(identifiant);

ex : window.clearTimeout(timer1);

# Comment vérifier un formulaire

La saisie d'un formulaire s'effectue via la création d'une balise form qui elle-même contient un certain nombre de champs de saisie ET un bouton submit.

```
<form action="script.php" method="post">
```

```
<input type="text"../>
```

```
<input type="checkbox"../>
```

...

```
<input type="submit" value="enregistrer" />
```

```
</form>
```

# Comment vérifier un formulaire

Les éléments d'un formulaire sont déjà nativement interactif (cette interactivité a été mise en place par le navigateur)

- Les champs de saisies réagissent aux frappes du clavier (le caractère tapé apparaît dans la zone)
- Les boutons radios ou les case à cocher réagissent au click (ou sélection via le clavier) une coche ou un rond noir apparaît dans l'élément sélectionné et l'élément dispose alors d'un attribut checked à true
- les input de type submit ou image réagissent au click (ou idem sélection clavier). Il y a automatiquement diffusion d'un événement submit lorsqu'on clique sur ces boutons
- Le formulaire (balise form) réagit à l'événement submit . Automatiquement ce que vous avez indiqué dans l'attribut action (généralement un script serveur) est exécuté et les données saisies sont transmises.

# Comment vérifier un formulaire

La données saisies doivent toujours être vérifiées avant d'être traitées. Cette vérification doit toujours avoir lieu sur le serveur. On va généralement la doubler d'une vérification en javascript :

- permet d'avertir immédiatement l'internaute des erreurs qu'il a commises
- évite qu'il perde sa saisie
- évite de trop surcharger le serveur :

**Etape1** : Bien écrire votre formulaire html : balise form et input submit ou image obligatoires.

**Etape 2** : l'objet **formulaire** doit écouter l'événement submit.

```
addEventListener(document.getElementById("form1"), "submit", verifier);
```

**Rque** : Vous ne **devez surtout pas** mettre en place la vérification du formulaire sur l'événement click du bouton submit.

# Comment vérifier un formulaire

**Etape 3 :** Programmer la fonction verifier. Si la saisie contient des erreurs vous devez **obligatoirement arrêter l'interaction native** (l'exécution du script serveur et l'envoi des données) en appelant la methode preventDefault() de l'objet Event.

```
function verifier(objetEvent) {  
  if (document.getElementById("nom").value=="") {  
    alert("veuillez remplir le nom");  
    objetEvent.preventDefault();  
  }  
}
```

exemple : [verifierFormulaire.htm](#)

# l'objet RegExp

On se sert de RegExp pour effectuer des recherches dans des chaînes de caractères ou pour remplacer les occurrences par d'autres. On s'en sert aussi principalement pour valider le format d'une saisie dans un formulaire. Par exemple, pour vérifier qu'une date est bien sous la forme jj/mm/aaaa, qu'un email est valide, qu'un numéro de téléphone est valide, etc...

Pour créer une instance de cet objet :

```
var exp=new RegExp("modèle",[option]) ;
```

[option](#) permet de préciser le type de recherche à effectuer :

- **g**: Recherche globale, c'est à dire ne s'arrête pas à la première occurrence trouvée, fait une recherche sur toute la chaîne.
- **i**: Ignore la casse.
- **m**: Fait une recherche sur plusieurs lignes.

[modèle](#) est composé d'une position(optionnelle) et d'un ensemble de classes de caractères

# L'objet RegExp

Classe de caractère peut être construit à l'aide des caractéristiques suivantes

[xyz]	caractères présents dans les crochets	[abc] correspond au caractère a ou b ou c
[^xy]	interdit les caractères présents dans les crochets	[^cp] correspond à un caractère différent de c et de p
\d	Correspond à un caractère représentant un chiffre	
\D	Correspond à un caractère ne représentant pas un chiffre	
\w	Correspond à un caractère représentant une lettre, un chiffre ou un souligné	
\W	tout ce qui n'est pas lettre, chiffre, souligné	
\s	espace, retour à la ligne, retour chariot, tabulation	
\S	tout ce qui n'est pas espace, retour à la ligne, retour chariot, tabulation	
*	le caractère peut apparaître 0 ou n fois	ex : m* : m peut être présent 0 ou n fois
+	le caractère doit apparaître au moins une fois	ex : m+ : m doit au moins être présent une fois
?	le caractère doit apparaître 0 ou 1 fois	ex : m? : m doit être présent 0 ou 1 fois
{n}	le car. doit apparaître n fois	ex m{2} m doit être présent 2 fois
{n,}	il doit apparaître au moins n fois	ex: m {2,} m doit au minimum être présent 2 fois
{n,m}	au moins n fois et au plus m fois.	ex m {2,3} m doit être présent 2 fois minimum et 3 fois maximum
a   b	a ou b	



# L'objet RegExp

## Position

^	correspondance en début de chaîne
\$	correspondance en fin de chaîne
\b	correspondance en début de mot
\B	correspondance en fin de mot

## Méthode de RegExp

Méthode	Description
exec(chaîne)	Exécute la recherche de l'expression régulière dans chaîne. Si aucune valeur n'est trouvée retourne null, sinon retourne un tableau contenant la valeur de la première occurrence trouvée. si on rappelle une deuxième fois exec retourne la deuxième, etc
test(chaîne)	Retourne un booléen indiquant si la chaîne respecte le modèle. retourne vrai si c'est le cas, faux sinon.

Rque : les caractères spéciaux doivent être précédés de \\

- ♦ \\ correspond au caractère '\'
- ♦ \\n correspond à un saut de ligne
- ♦ \\f correspond à un saut de page
- ♦ \\r correspond à un retour chariot
- ♦ \\t correspond à une tabulation
- ♦ \\.. correspond à tout caractère

Exemple : [verifierAvecRegExp.htm](#)

# Comment stocker une information :cookie

Javascript ne peut en aucun cas manipuler des fichiers présents sur le poste de l'utilisateur, que ce soit en lecture ou en écriture.

Il peut, malgré tout, être utile de conserver des informations sur le poste de l'utilisateur pendant une durée déterminée, et ce, même après la fermeture du navigateur.

Javascript met à notre disposition ce que l'on appelle des cookies pour effectuer cela.

## Qu'est-ce qu'un cookie :

C'est une information qui est stockée sur le disque dur de l'utilisateur et qui donc persiste d'une session à une autre, y compris lorsque l'ordinateur est éteint.

Ils sont disponibles dans le fichier cookies.txt pour netscape et mozilla et dans le répertoire Cookies pour internet Explorer. Si vous en ouvrez un, vous trouverez une longue liste de site chacun accompagné d'une chaîne de texte généralement incompréhensible.

# Comment stocker une information :cookie

- La taille d'un cookie ne peut dépasser 4 kilo-octet (environ 4000 caractères)
  - Un site web ne peut définir et lire que ses propres cookies.
- Vous n'avez droit qu'à 20 cookies par domaine (netscape) (adresse URL principale)
  - Tous les navigateurs n'acceptent pas les cookies
  - Tous les utilisateurs n'acceptent pas les cookies.

# Comment stocker une information :cookie

Un cookie est formé de :


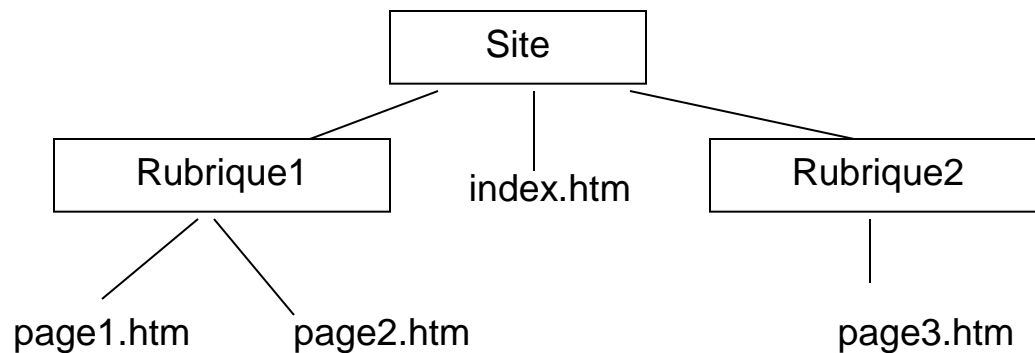
- **Un nom** : comme vous pouvez en avoir plusieurs, c'est ce qui les différencient. Même règle de nommage que les autres variables.
- **Une valeur** : Elle composée d'une chaîne de caractère dont les conventions sont celles d'une URL, c'est à dire que les caractères spéciaux sont remplacés leur correspondance Hexadécimale (espace = %20...). Pour cela on utilise escape et unescape (voir objet Global)
- **Une date d'expiration** : Les cookies peuvent exister uniquement le temps de la session (dans ce cas ils sont supprimés lorsque l'utilisateur quitte le navigateur) ou bien ils servent à stocker des informations devant être disponible hors session et on indique combien de temps le cookie doit exister sur le poste client.

En JavaScript, la date doit être au format **UTC** (pour cela utiliser la fonction `toUTCString()` de `Date`)

Si vous n'indiquez pas de date d'expiration, il n'est valable que le temps de la session.

# Comment stocker une information :cookie

• **Un chemin** : Par défaut **le chemin d'un cookie est le répertoire courant de la page ayant créé le cookie**. Cela a une importance pour les autorisations d'accès à ce cookie. En effet seuls les pages appartenant à ce même répertoire ou bien à des sous-répertoires pourront consulter ou modifier le cookie.



page1.htm crée le cookie "cookie1". Seule y aura accès : page2.htm. index.htm et page3.htm n'auront ni le droit de lecture, ni le droit d'écriture. Si c'est index.htm qui crée ce cookie toutes les pages y auront accès

Si on modifie le chemin, c'est à partir de celui-ci que seront définies les autorisations d'accès. La valeur **/** correspond à la racine du site et dans ce cas toutes les pages du site ont accès au cookie

# Comment stocker une information :cookie

**Le domaine** : Afin que vous ne puissiez pas utiliser les cookies d'autres domaines, celui-ci est rattaché explicitement au domaine l'ayant créé. Cela est fait par défaut par javascript.

**secure** : C'est un booléen qui permet de demander au cookie de n'être accessible que par le SSL (cryptage des données).

En javascript un cookie est donc construit selon le modèle suivant

nom=**valeur**;expires=**date**;path=**chemin**;domain=**domaine**;**[secure]** .

Pour créer un cookie , il faut écrire cette chaîne de caractère dans la propriété cookie de document.

**window.document.cookie**="nom= escape( **valeur**) ;expires=**date**;  
path=**chemin**;domain=**domaine**;**[secure]**";

# Comment stocker une information :cookie

## 2eme exemple avec enregistrement de plusieurs valeurs

### Comment modifier un cookie ?

Il suffit de recréer le cookie en lui donnant le même nom, cela efface la précédente valeur.

Si on veut ajouter des informations au cookie :

- il faut récupérer la valeur stockée
- y ajouter la nouvelle valeur
- recréer le cookie

# Comment stocker une information:localStorage

Avec l'arrivée de html5, une autre manière de sauvegarder des données sur le poste client : l'utilisation d'un objet localStorage (ou sessionStorage suivant la durée de vie demandée) propriété de l'objet window.

Comme pour les cookies les données stockées sont obligatoirement de type string et on stocke des couples clé/valeur

**localStorage** : les données sont stockées sans limitation de durée

**sessionStorage** : les données sont stockées aussi longtemps que la fenêtre n'est pas fermée

Exemple de sauvegarde : `window.localStorage.setItem("nom","Smith");`

Exemple de récupération :

`var nomSauvegarde = window.localStorage.getItem("nom");`



# Comment stocker une information:localStorage

API de localStorage ou sessionStorage :localStorage[cle]=valeur permet de stocker

localStorage.setItem(clé,valeur) ou localStorage[cle]=valeur permet de stocker une information

localStorage.getItem(clé) ou localStorage[cle] permet de lire une information stockée

localStorage.removeItem(clé) permet de supprimer une information

localStorage.length permet de connaître le nombre de valeurs stockées

localStorage.key(indice) permet de lire la ieme valeur stockée

localStorage.clear() permet de vider les valeurs stockées

Vous disposez aussi d'un évènement:

**storage** : cet événement est diffusé lorsqu'on dépose, supprime ou efface l'espace. L'object event associé dispose des propriétés suivantes:

- key : la clé
- oldValue :l'ancienne valeur
- newValue :la nouvelle valeur (ou null si il est supprimé)
- url : la page a l'origine du changement

# Comment stocker une information:localStorage

	cookie	localStorage	sessionStorage
Type des données stockées	String	String	String
Autorisation d'accès	Celles indiquées via le domaine et le chemin. Un site peut donc laisser des informations à destination d'un autre domaine	Le domaine ayant déposé	Le domaine ayant déposé
Navigateur y ayant accès	Celui ayant déposé	Celui ayant déposé	Celui ayant déposé
Restriction de stockage	4 KO – 20 cookies par domaine	5 MB par défaut. Eventuellement certains navigateurs permettent à l'utilisateur de le gérer	Idem localStorage
Durée de vie	Paramétrable par l'internaute: date d'expiration du cookie, session ou cookie non autorisé	Aussi longtemps qu'il n'est pas détruit	Lorsqu'on ferme la fenêtre
Compatibilité	Tous navigateurs	Tous navigateurs récents (IE8, firefox 3.5, Chrome 4.0, opera 10.5)	Idem localStorage
Suppression	Suppression facile pour les internautes	Suppression possible mais pas simple à trouver à l'heure actuelle	Idem
Transmission serveur	Le contenu de tous les cookies du domaine est envoyé automatiquement lors de toute connexion au serveur (http)	Jamais envoyé	Idem

# L'objet Array

Comme nous l'avons vu précédemment, cet objet permet de manipuler des variables de type tableau.

```
var unTableau=new Array()
```

Méthode ou propriété	Description
<code>unTableau.concat(tab1, tab2[,tab3, ...])</code>	permet de concaténer plusieurs tableaux i.e crée un tableau à partir des différents tableaux passés en paramètre
<code>unTableau.join(separateur)</code>	renvoie une chaîne de caractère constituée de la concaténation de tous les éléments du tableau et séparé par le caractère que vous avez indiqué en tant que séparateur <a href="#">Voir 1_Tableau.htm</a>
<code>unTableau.reverse()</code>	inverse l'ordre des éléments du tableau
<code>unTableau.slice(indicedebut , indicefin)</code>	retourne un tableau constitué de la plage d'éléments compris entre indexedebut et indicefin
<code>unTableau.sort()</code>	trie les éléments du tableau selon l'ordre ASCII <a href="#">Voir 1_tableauTri.htm</a>

# L'objet Math

C'est l'objet qui contient un certain nombre de fonctions mathématiques. il ne peut pas s'instancier (var math=new Math()) est impossible) mais le moteur de javaScript en crée automatiquement une instance lors de son initialisation.

Méthode ou propriété	Description
Math.abs(nombre)	valeur absolue
Math.acos (nombre),Math.asin(nombre),Math.atan(nombre)	arc cosinus et arc sinus , arc tangente
Math.ceil(nombre)	entier supérieur ou égal à nombre
Math.round(nombre)	arrondit à l'entier le plus proche (inférieur si partie décimale de nombre <0.5, supérieur sinon)
Math. cos (nombre),Math. sin(nombre),Math. tan(nombre)	cosinus , sinus , tangente
Math.floor(nombre)	entier inférieur ou égal à nombre
Math.log(nombre)	logarithme naturel
Math.max(valeur1,valeur2)      Math.min(valeur1,valeur2)	calcul maximum et minimum de 2 valeurs
Math.pow(nombre1,nombre2)	calcul nombre1 puissance nombre2
Math.random()	retourne un nombre compris entre 0 et 1
Math.sqrt(nombre)	calcule la racine carrée

# L'objet String

JavaScript crée automatiquement des objets String lorsque vous déclarez des variables avec de valeurs de type chaîne de caractère . Pour connaître la taille d'une chaîne de caractère vous avez la propriété **length** (maChaine.length)

Méthode	Description
<code>maChaine.anchor("nom")</code>	Entoure la chaine avec les balises d'ancrage. <code>&lt;a name="nom"&gt; maChaine &lt;/a&gt;</code>
<code>maChaine.charAt(indice)</code>	retourne le caractère présent dans la chaîne à l'indice indiqué. 1er caractère correspond à l'indice 0.
<code>maChaine.charCodeAt(indice)</code>	ramène le code unicode du caractère présent à l'indice indiqué
<code>maChaine.concat("chaîne2")</code>	concatène 2 chaînes
<code>maChaine.indexOf("sous-chaîne" [,indice départ])</code>	ramène l'indice de la première occurrence trouvée,-1 si elle n'existe pas. <code>indicedepart</code> permet d'indiquer à partir de quel indice s'effectue la recherche.
<code>maChaine.substring(pos1, pos)</code>	retourne la sous-chaîne comprise entre <code>pos1</code> et <code>pos2</code>
<code>maChaine.toUpperCase()</code>	retourne la chaîne en majuscule
<code>maChaine.toLowerCase()</code>	retourne la chaîne en minuscule
<code>maChaine.split("caractère de séparation")</code>	La méthode <code>split</code> utilise un caractère ou un groupe de caractères pour diviser une chaîne en ensemble de sous-chaînes. Retourne un tableau contenant chacune des sous-chaînes. <a href="#">voir 1_split.htm</a>

# L'objet Global

C'est un objet un peu particulier car on ne peut pas créer d'instance de cet objet, il est automatiquement créé par le moteur javascript. On accède à ces méthodes directement (sans préciser global)

Méthode	Description
<code>parseInt(String,base)</code>	Transforme une chaîne en nombre entier dans la base indiquée. Si la chaîne ne contient pas de nombre retourne NaN
<code>parseFloat(String,base)</code>	Transforme une chaîne en nombre réel. Si la chaîne ne contient pas de nombre retourne NaN
<code>eval(String)</code>	évalue une expression et la calcule : Ex: <code>eval("2+3")</code> a pour résultat 5 ; <code>eval ("2" +"+" "3")</code> a pour résultat 5
<code>isNaN(valeur)</code>	Retourne true si valeur est égal à NaN(not a number), false sinon
<code>escape(String)</code>	Encode les chaînes pour les rendre lisibles sur tous les ordinateurs. Elle convertit les caractères ASCII en caractères UNICODE. Tous les espaces, la ponctuation, les caractères accentués, les caractères non ASCII sont remplacés par un pourcentage (%) suivi de la valeur hexadécimale du caractère.
<code>unescape(String)</code>	Décode les chaînes encodées à l'aide de la fonction <code>escape(String)</code> .

# L'objet window

Cet objet référence la fenêtre (celle-ci pouvant être une fenêtre classique ou une frame).

S'utilise pour accéder aux informations concernant l'état de la fenêtre, le navigateur utilisé pour l'afficher, le document html,....

Quelques propriétés (liste non exhaustive)

propriété	Description
<code>window.closed</code>	retourne un booléen indiquant si la fenêtre est fermée ou non
<code>window.status</code>	permet d'accéder au message affiché dans la barre d'état
<code>window.defaultStatus</code>	valeur par défaut de la barre d'état
<code>window.event</code>	retourne une référence vers l'objet Event qui permet d'accéder aux paramètres de l'évènement qui représente l'objet HTML
<code>window.frames</code>	retourne un HTMLCollections des frames correspondant aux frames créés à l'aide de la balise frame dans un <frameset>
<code>window.name</code>	définit ou retourne le nom de la fenêtre
<code>window.navigator</code>	retourne une référence vers l'objet Navigator qui permet de collecter des informations sur le navigateur de l'utilisateur
<code>window.screen</code>	retourne une référence vers l'objet Screen qui permet de collecter des informations sur l'écran de l'utilisateur

# L'objet window

## Quelques méthodes (liste non exhaustive)

Méthodes	Description
<code>window.alert("message")</code>	affiche une boîte de message
<code>window. blur()</code>	fait perdre le focus à la fenêtre et déclenche son événement onblur
<code>window. focus()</code>	donne le focus à la fenêtre et déclenche son événement onfocus
<code>window.setTimeout(fonction à exécuter, millisecondes)</code>	Indique une fonction à exécuter au bout de x millisecondes. Retourne un identifiant.
<code>window.clearTimeout(identifiant)</code>	détruit le décompte de temps créé par la méthode au-dessus, la fonction ne sera donc pas exécutée
<code>window.setInterval(fonction à exécuter, millisecondes)</code>	Indique une fonction à exécuter toutes les x millisecondes. Retourne un identifiant. <a href="#">Voir 2_majHeure.htm</a>
<code>window.clearInterval(identifiant)</code>	détruit le décompte de temps créé par la méthode au-dessus, la fonction ne sera donc pas exécutée
<code>window.navigate(url)</code>	affiche le document spécifié par l'url dans la fenêtre courante
<code>window.close()</code>	ferme la fenêtre du navigateur (ne s'utilise que pour les fenêtres ouvertes par script)
<code>window.prompt("message", valeur par défaut)</code>	affiche une boîte de dialogue et retourne la réponse
<code>window.confirm("message")</code>	affiche une boîte de dialogue avec le bouton Ok et annuler
<code>window.print()</code>	permet d'imprimer une fenêtre
<code>window.resizeTo(largeur, hauteur)</code>	permet de spécifier la taille d'une fenêtre



# L'objet Navigator

Il permet de collecter des informations sur le navigateur utilisé

propriété	Description
appCodeName	Retourne le nom de code du navigateur
appName	retourne le nom du navigateur
appVersion	retourne le numéro de version
browserLanguage	retourne la langue du browser
cookieEnabled	retourne un booléen indiquant si l'utilisateur accepte les cookies

# L'objet Screen

Il permet de collecter des informations sur l'écran de l'utilisateur

propriété	Description
height	Hauteur en pixel de l'écran
width	Largeur en pixel de l'écran
availHeight	Retourne la hauteur en pixel de l'espace écran disponible
availWidth	Retourne la largeur en pixel de l'espace écran disponible

# L'objet location

Il permet de collecter des informations sur l'URL de la page (window.location)

propriété	Description
hash	Nom de l'ancre à l'intérieur de l'URL
host	Nom de domaine à l'intérieur de l'url
href	URL/ lien à une URL
pathname	Nom du chemin à l'intérieur de l'url

méthodes	Description
reload()	Permet de recharger la page actuelle (idem à actualiser)
replace()	Charge une autre adresse URI sur l'élément actuel dans la liste des pages visitées (historique). L'élément actuel n'apparaît plus alors dans les pages visitées.