

Séance 9

DOM et programmation événementielle

Technologies web

HENALLUX — IG2 — 2016-2017

DOM et progra événementielle

➤ **Le DOM : kézako ?**

- Le lien entre Javascript et HTML

➤ **L'objet document**

- L'arborescence HTML dans le DOM

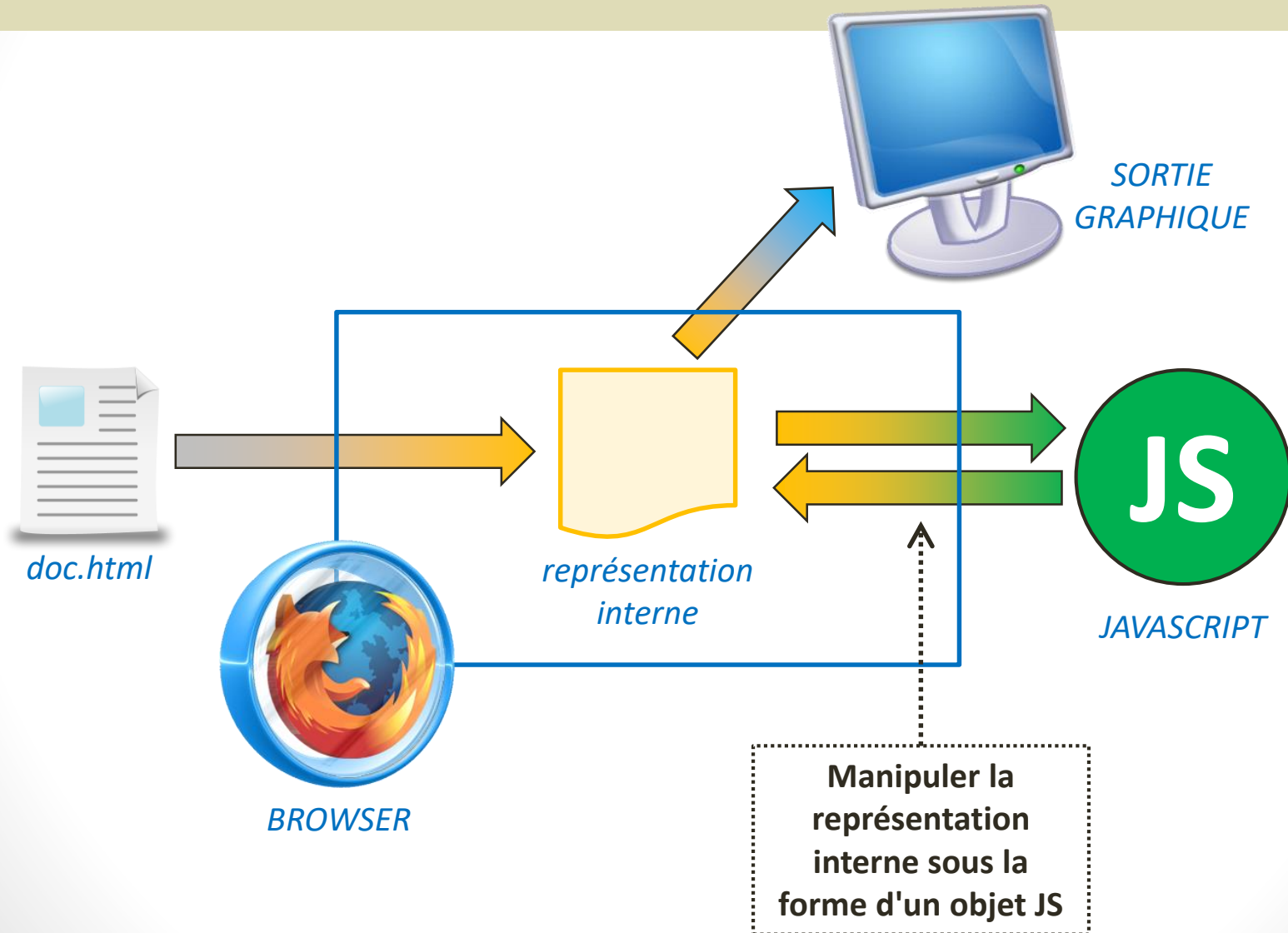
➤ **Utilisation du DOM**

- Parcourir l'arborescence HTML
- Cibler un élément HTML (ou plusieurs)
- Modifier un élément HTML (contenu, attributs, style, classe)
- Modifier l'arborescence HTML

➤ **Programmation événementielle**

➤ **L'objet global window**

DOM – késako ?



DOM – késako ?

- **Dom = Document Object Model**
= modèle représentant le document sous forme d'objet
- But : permettre à Javascript de
 - consulter (en **lecture**) le contenu du document, et de
 - modifier (en **écriture**) le contenu du document.
- Standardisé par W3C (avant, chaque browser faisait à sa sauce)

DOM – késako ?

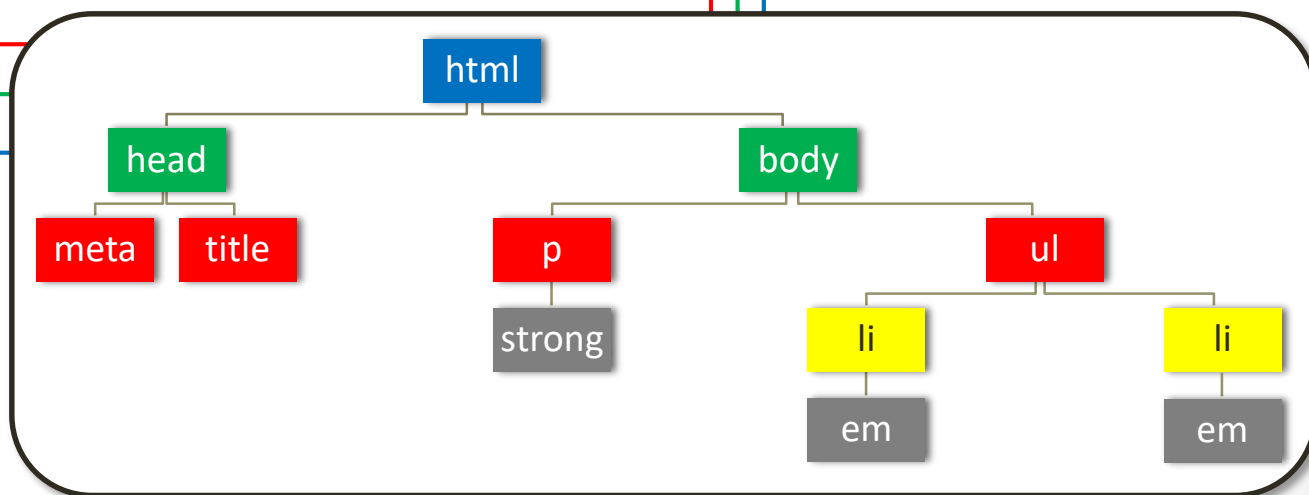
- Le DOM indique comment chacun des éléments de l'arborescence HTML est représenté en mémoire.
 - Quels **objets** ? Avec quels **attributs** et quelles **méthodes** ?
 - Comment utiliser ces objets pour rendre le **HTML dynamique**.
 - Recueillir des informations sur les éléments HTML
 - Modifier ces informations (contenu, attributs, style, classe)
 - Manipuler l'arborescence HTML (ajouter/déplacer/supprimer)
- Exemple déjà vu :

```
let para = document.getElementById("idParagraphe");  
para.innerHTML = "Nouveau texte à afficher !";
```
- Le DOM ne se limite pas au document mais couvre également d'autres informations (fenêtre d'affichage, navigateur...).

L'arborescence HTML (rappel)

Note : il s'agit d'une version simplifiée, car on ignore les blancs.

```
<html>  
  <head>  
    <meta charset="ISO-8859-1"/>  
    <title>Exemple en HTML</title>  
  </head>  
  <body>  
    <p>Un <strong>arbre</strong> possède :</p>  
    <ul>  
      <li>Une <em>racine</em> et</li>  
      <li>des <em>feuilles</em></li>  
    </ul>  
  </body>  
</html>
```

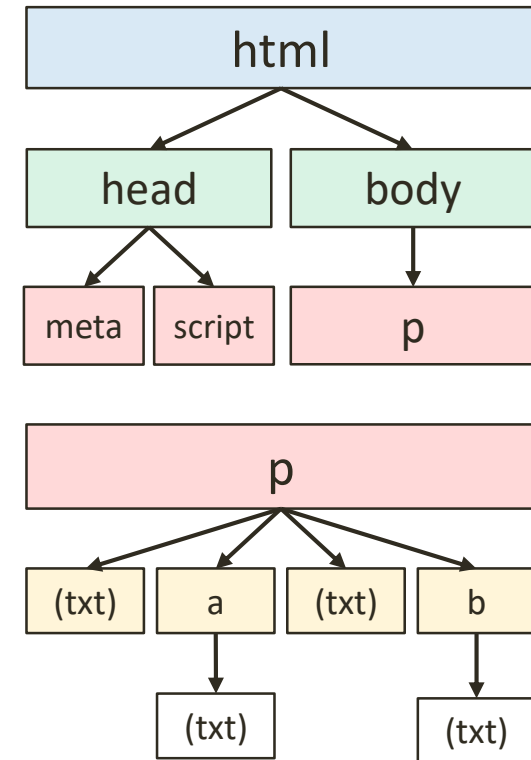


L'objet **document**

- L'objet **document** représente le document HTML.
- Il permet à Javascript d'accéder au contenu HTML (tant en lecture qu'en écriture).
- Structure en arbre, où chaque élément HTML correspond à un nœud et est décrit par un objet.
 - **noeud.childNodes** : tableau reprenant tous les fils du nœud
 - **noeud.children** : tableau ne reprenant que les fils qui sont eux-mêmes des éléments.
 - Un nœud peut être
 - soit un élément (**noeud.nodeName** = le nom de la balise HTML)
 - soit une donnée textuelle (**noeud.nodeName** = "#text" ; son contenu est dans **noeud.textContent**)

L'arborescence HTML

```
<html>  
  <head>  
    <meta charset="ISO-8859-1"/>  
    <script>  
      function crie () {alert("Bouh!")}  
    </script>  
  </head>  
  <body>  
    <p>Si tu  
      <a href="javascript:crie();">cliques</a>  
      alors je  
      <b>crie</b>  
    </p>  
  </body>  
</html>
```



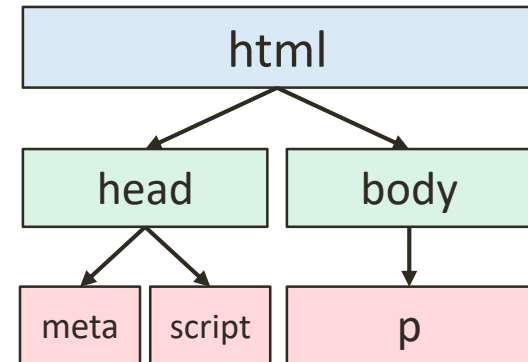
Note : version simplifiée, car on ignore les blancs (voir plus loin).

L'arborescence HTML

`document.childNodes[0]`

`document.childNodes[0].childNodes[1]`

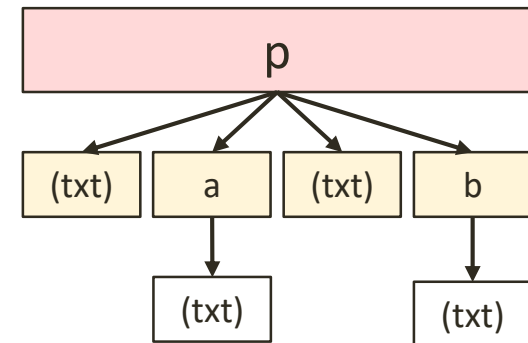
`document.childNodes[0].childNodes[1].childNodes[0]`



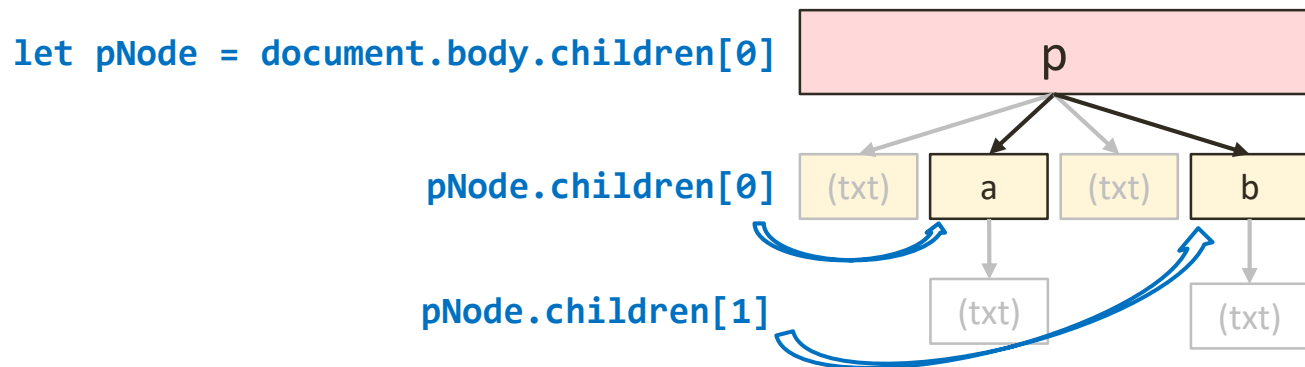
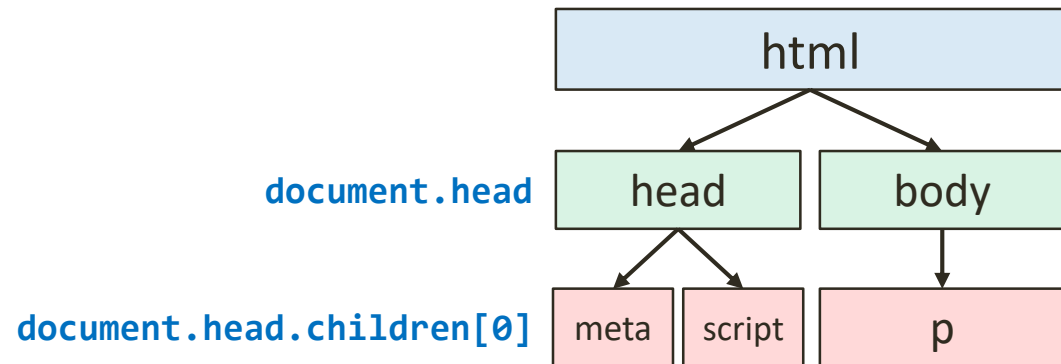
`var pNode = document.childNodes[0].childNodes[1].childNodes[0]`

`pNode.childNodes[0]`

`pNode.childNodes[1].childNodes[0]`



L'arborescence HTML




L'arborescence HTML

Tous les
espacements sont
comptés comme
des `childNodes` de
type "texte".

*Note : cela ne
change rien aux
children !*

```
<html>  
  <head>  
    <meta charset="ISO-8859-1"/>  
    <script>  
      function crie () {alert("Bouh!")}  
    </script>  
  </head>  
  <body>  
    <p>Si tu  
    <a href="javascript:crie();">cliques</a>  
    alors je  
    <b>crie</b>  
  </p>  
</body>  
</html>
```

Utilisation du DOM

- 
- **Cibler un élément HTML (ou plusieurs)**
 - **Modifier un élément HTML**
 - Contenu, attributs, style, classe
 - **Modifier l'arborescence HTML**

Ensuite : *Programmation événementielle*

Cibler un élément

- Plusieurs méthodes :
 - **#1 Accès direct** : `document.head`, `document.images[2]`, ...
 - **#2 Accès indirect** : en se déplaçant dans l'arborescence
 - `unElem.nextElementSibling`, `unElem.parentElement`
 - Voir slides précédents
 - **#3 Recherche** selon un critère
 - Identificateur : `document.getElementById("monID")`
 - Aussi : par balise, par classe (CSS), en fonction d'un sélecteur

Cibler un élément

Méthode #1 : accès direct

- L'objet `document` permet d'accéder directement à certains éléments HTML :
 - `document.head`
 - `document.body`
- Il possède également des propriétés donnant des collections d'éléments semblables :
 - `document.anchors` : toutes les ancrs ``
 - `document.forms` : tous les formulaires `<form>`
 - `document.images` : toutes les images ``
 - `document.links` : tous les liens ``
 - `document.applets`, `document.embeds`

Cibler un élément

Méthode #2 : accès indirect (se déplacer autour d'un élément)

- Vers le bas dans l'arborescence HTML :
 - `elem.childNodes` : collection des fils de elem, qui sont
 - soit des éléments eux-mêmes
 - soit des données textuelles (contenu : `.textContent`)
 - `elem.children` : collection des éléments fils de elem
 - ignore les données textuelles
 - Aussi : `elem.firstChild`, `elem.firstElementChild`, `elem.lastChild`, `elem.lastElementChild`
- Dans les autres directions
 - Vers le haut : `elem.parentNode`
 - Horizontalement : `elem.nextSibling`, `elem.nextElementSibling`, `elem.previousSibling`, `elem.previousElementSibling`

Cibler un élément

Méthode #3 : recherche

- Selon l'**identificateur** : `document.getElementById(id)`
 - Une seule réponse !
- Selon la **balise** ou la **classe** :
 - `document.getElementsByTagName(balise)`
 - `document.getElementsByClassName(classe)`
 - Plusieurs réponses !

Résultat = collection d'éléments (peut être utilisée comme un tableau dans la plupart des cas).

Ex : `let titres = document.getElementsByTagName("h1")`
... `titres[0]` ...
... `for (let titre of titres)` ...

Conversion en tableau : `Array.of(...titres)`

Cibler un élément

Méthode #3 : recherche (suite)

- Via un **sélecteur CSS** :
 - `document.querySelector(sel)` renvoie le premier élément
 - `document.querySelectorAll(sel)` renvoie tous les éléments
- Ex : `document.querySelector("#cadre")`
`document.querySelectorAll("p.rouge")`

Note. Toutes ces méthodes de recherche (sauf `getElementById`) sont disponibles sur tous les éléments HTML (pas seulement `document`).

`elem.getElement...` : recherche à l'intérieur de l'élément

Note. Les méthodes de recherche qui renvoient plusieurs éléments produisent des objets itérables (for-of).

Modifier un élément HTML

- On peut modifier :
 - **#1 le contenu**
 - `innerHTML`, `textContent`
 - **#2 la valeur des attributs**
 - Généralement nom de propriété = nom d'attribut
 - **#3 le style**
 - Via la propriété `style` (correspondant au style "inline")
 - **#4 les classes**
 - Via la propriété `classList` (collection de classes)

Modifier un élément HTML

#1 Modifier le contenu d'un élément HTML

- Le contenu tel que présenté dans le code HTML : `elem.innerHTML`
 - en lecture ou en écriture
- Le contenu tel qu'affiché dans le navigateur : `elem.textContent`
 - en lecture : donne le texte affiché
 - en écriture : convertit les caractères pour qu'ils soient tous affichés

- Exemples

`elem.innerHTML = "mot"` → *mot*

`elem.textContent = "mot"` → `mot`

Modifier un élément HTML

#2 Modifier la valeur des attributs

- Utiliser le nom de l'attribut HTML comme propriété
 - HTML : `Vers le site`
``
 - Javascript : `lien.href = "http://www.henallux.be";`
`image.src = "monchat.jpg";`
 - Autre méthode :
`let cible = lien.getAttribute("href");`
`lien.setAttribute("href", "http://www.henallux.be");`

Modifier le style d'un élément

#3 Modifier le style d'affichage

- Via la propriété `style` de l'élément (transformer les noms de propriétés CSS en camelCase) :

```
elem.style.backgroundColor = "black";  
elem.style.color = "yellow";
```

- **Note.** Pour observer le style d'affichage actuel, il faut utiliser

```
actuel = window.getComputedStyle(elem);  
... actuel.backgroundColor, actuel.color ...
```

- Liste des propriétés de style : voir par exemple

http://www.w3schools.com/jsref/dom_obj_style.asp

Modifier le style d'un élément

#4 Modifier la (les) classe(s)

- Utiliser la propriété `classList` de l'élément :
 - `elem.classList.add(nom)` : ajoute une classe
 - `elem.classList.remove(nom)` : enlève une classe
 - `elem.classList.toggle(nom)` : ajoute la classe si elle n'est pas présente, la supprime si elle est présente
 - `elem.classList.contains(nom)` : indique si une classe est présente ou pas (renvoie un booléen)
- **Note.** Javascript ne permet pas de modifier les règles CSS (ni, donc, la définition des classes). Par contre, on peut définir à l'avance des classes inutilisées dans le document HTML et les attribuer via un script Javascript.

Modifier l'arborescence HTML

- On peut modifier l'arborescence pour
 - **#1 supprimer un nœud existant**
 - `parent.removeChild(noeud)`
 - **#2 déplacer un nœud existant**
 - L'ajouter au contenu d'un élément : `elem.appendChild(noeud)`
 - Le placer dans un élément, avant un autre nœud :
`elem.insertBefore(noeud, autrenoeud)`
 - L'utiliser pour remplacer un autre nœud dans un élément :
`elem.replaceChild(noeud, autrenoeud)`
 - **#3 ajouter un nouveau noeud**
 - Créer un nœud via `document.createElement(balise)`
 - Puis le déplacer

Modifier l'arborescence HTML

#1 Supprimer un nœud existant

- Supprimer le fils d'un élément :
`elem.removeChild(fils);`
 - Attention : message à envoyer au parentNode de l'élément à supprimer !
- Supprimer tous les fils d'un élément :
`elem.innerHTML = "";`

Modifier l'arborescence HTML

#2 Déplacer un nœud existant


- On place l'élément existant comme fils d'un nœud parent.
- Trois positions possibles :
 - `noeudParent.appendChild(elem)`
ajoute elem après le contenu de noeudParent
 - `noeudParent.insertBefore(elem,filsParent)`
ajoute elem dans noeudParent, juste avant filsParent
 - `noeudParent.replaceChild(elem,filsParent)`
remplace le fils filsParent de noeudParent par elem

Modifier l'arborescence HTML

#3 Ajouter un nouveau nœud

- Créer un nouveau nœud
 - Via `document.createElement(balise)`
 - Exemple : `let nvNoeud = document.createElement("a");`
- Remplir le nouveau nœud
 - Définir le contenu (`innerHTML`) et/ou les attributs
 - Exemple : `nvNoeud.innerHTML = "lien";`
`nvNoeud.href = "www.google.com";`
- Placer le nœud en utilisant une des trois méthodes de déplacement (voir slide précédent)
 - Exemple : `parentNode.appendChild(nvNoeud)`

Programmation événementielle

- 
- Gestion des événements
 - Associer une action à un événement
 - Événements et phases d'exécution

Ensuite : *L'objet global window*

Gestion des événements

- En **programmation événementielle**, on associe
 - des scripts à exécuter
 - à des événements-déclencheurs.
- Quelques exemples standards d'événements-déclencheurs :
 - clic sur un bouton
 - passage de la souris au-dessus d'une image
 - fin du chargement de la page web
- Le **DOM** permet d'associer des scripts à une série d'événements prédéfinis pour chacun des éléments HTML.

Gestion des événements

- Quelques-uns des événements définis par le DOM :
 - Événements de type "souris"
`click`, `dblclick`, `mousedown`, `mouseup`, `mouseover`, `mouseout` ...
 - Événements de type "clavier"
`keypress`, `keydown`, `keyup`
 - Événements généraux
`load` (quand le document / une image est entièrement chargé),
`error`, `resize`, `scroll`, `unload` ...
 - Événements de type "formulaire"
`blur`, `focus` (perte et gain de focus),
`select` (sélection d'un bout de texte dans un champ),
`change` (modification),
`submit` (bouton "soumettre"), `reset`

Gestion des événements

- L'**action à exécuter** lors d'un événement-déclencheur
 - se présente sous la forme d'une fonction Javascript
 - qui peut recevoir un argument décrivant l'événement
- L'**événement** est décrit par un objet `evt` possédant diverses propriétés.
 - `evt.clientX`, `evt.clientY` : coordonnées de la souris dans la fenêtre
 - `evt.offsetX`, `evt.offsetY` : coordonnées de la souris dans l'élément
 - `evt.keyCode` : code ASCII de la touche pressée
 - `evt.target` : élément qui est à l'origine de l'événement
 - `evt.currentTarget` : élément actuellement en train de répondre à l'événement (varie au cours des phases)
 - `evt.type` : type de l'événement ("click", "keypress"...)
 - `evt.preventDefault()` : pour empêcher l'action par défaut associée à l'événement en question

Lier une action à un événement

- Méthode #1 : **dans le code HTML**

```
<button onclick="action();">Click</button>
```

- Méthode #2 : dynamiquement, **via Javascript**

```
but.addEventListener("click", action);
```

1^{er} argument = type d'événement

2^e argument = action à accomplir (fonction)

Dans la fonction, "this" fait référence au nœud déclencheur.

La fonction peut recevoir un paramètre décrivant l'événement.

- (Dangereux) Si on ne lie qu'une seule action à un événement :

```
but.onclick = action;
```

Lier une action à un événement

- Exemple

```
function clicBouton (evt) {
    alert("Vous avez cliqué sur le bouton !");
}
let bouton = document.getElementById("monBouton");
bouton.addEventListener("click", clicBouton);
```

- Exemple (utilisation de preventDefault)

```
function confirmerLien (evt) {
    let reponse = confirm("Voulez-vous vraiment suivre ce lien ?");
    if (!reponse) event.preventDefault();
}
lien.addEventListener("click", confirmerLien);
```


Lier une action à un événement

- Pour enlever une action associée :
`but.removeEventListener("click", fonction);`
 Il faut pouvoir faire référence à la fonction !

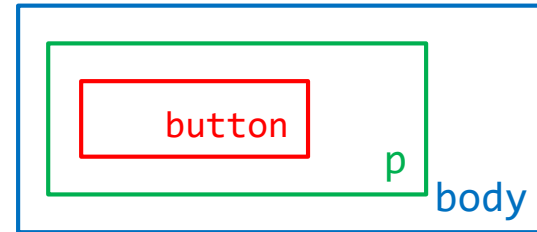
Exemple :

```
function uneFois (evt) {
    alert("Click réussi !");
    bouton.removeEventListener("click", uneFois);
}
bouton.addEventListener("click", uneFois);
```

Phases d'exécution

- **Phases d'exécution**

- Si plusieurs éléments superposés ont un événement "onmouseover", que se passe-t-il ?



- On donne la main

1. à button

2. à p

3. à body

4. à p

5. à button

} phase ascendante (bubbling up)

} phase descendante

- `but.addEventListener("click", action, false);`
3^e argument : false/true pour phase ascendante/descendante

À la racine du DOM : `window`

- L'objet `window` représente la fenêtre où le document HTML est affiché.
 - C'est le plus gros objet du DOM : il "contient" tous les autres.
 - Exemple : `document` est une des propriétés de l'objet `window`.
 - On peut toujours omettre le préfixe `window`.
 - Exemple : `document` au lieu de `window.document`
- En fait, l'objet `window` est le contexte global. Toutes les fonctions globales et toutes les variables globales déclarées via `var` sont des propriétés de `window`.

```
Ex : var x = 3;           function go () {alert("go")};  
      alert(window.x);    window.go();
```

À la racine du DOM : window

- Quelques **sous-objets / propriétés** de l'objet **window**

- **window.document** : le document affiché **CONTENU**
- **window.location** : adresse de la page
.href, .protocol, .hostname, .pathname, .port, .search,
.reload()...
- **window.history** : historique de navigation pour cette fenêtre
.length, .back(), .forward()...
- **window.frames** : tableau des frames
aussi **window.parent**, **window.top**... pour la gestion des frames
- **window.navigator** : navigateur utilisé
.appName, .appVersion, .appName...
- **window.screen** : écran où l'affichage se produit
.width, .height...

À la racine du DOM : `window`

- Quelques **propriétés** de l'objet `window`
 - `window.open` : la fenêtre responsable de son ouverture
Restriction (sous Firefox) : le code Javascript ne peut modifier la position ou la taille d'une fenêtre ou encore la fermer que s'il s'agit d'une fenêtre que le code a créée lui-même ("popup").
 - `window.status` : texte d'état (barre de statut)
 - `window.screenX`, `window.screenY` : position sur l'écran
 - `window.innerHeight`, `window.outerHeight`,
`window.innerWidth`, `window.outerWidth` : taille interne/externe
 - `window.onload` : action à exécuter dès que le contenu de la page est entièrement chargé (très pratique pour les initialisations !)

À la racine du DOM : window

- Quelques **méthodes** de l'objet `window`
 - `window.alert()`, `.prompt()`, `.confirm()`
 - `.moveTo(x,y)`, `.moveBy(dx,dy)` : repositionne la fenêtre
 - `.resizeTo(x,y)`, `.resizeBy(dx,dy)` : ajuste la taille
 - `.scrollTo(x,y)`, `.scrollby(dx,dy)` : déroulement

Note : ces méthodes ne peuvent être utilisées que sur des fenêtres créées via Javascript (pour éviter les changements indésirables).
- `.close()` : ferme la fenêtre
- `.open(url,nom,options)` : ouvre une nouvelle fenêtre (popup)

```
var fen2 = open("http://site.be", "Mon site",  
"resizable=no, location=no, width=200, height=100,  
menubar=no, status=no, scrollbars=no");
```

À la racine du DOM : window

- Quelques **méthodes** de l'objet **window** (suite)
 - **.setTimeout(f,t[,param,param])** : exécute la fonction f dans t milliseconde et renvoie l'id de la tâche
 - **.clearTimeout(id)** : suspend l'exécution d'une tâche timeout

Ex:

```
function crie () { alert("Bouh!") };  
let id = setTimeout(crie, 3000);
```

- **.setInterval(f,t[,param,param])** : exécute la fonction f toutes les t milliseconde et renvoie l'id de la tâche
- **.clearInterval(id)** : suspend l'exécution d'une tâche interval

Ex:

```
function crie () { alert("Bouh!") };  
let id = setInterval(crie, 3000);
```