

Le D.O.M.

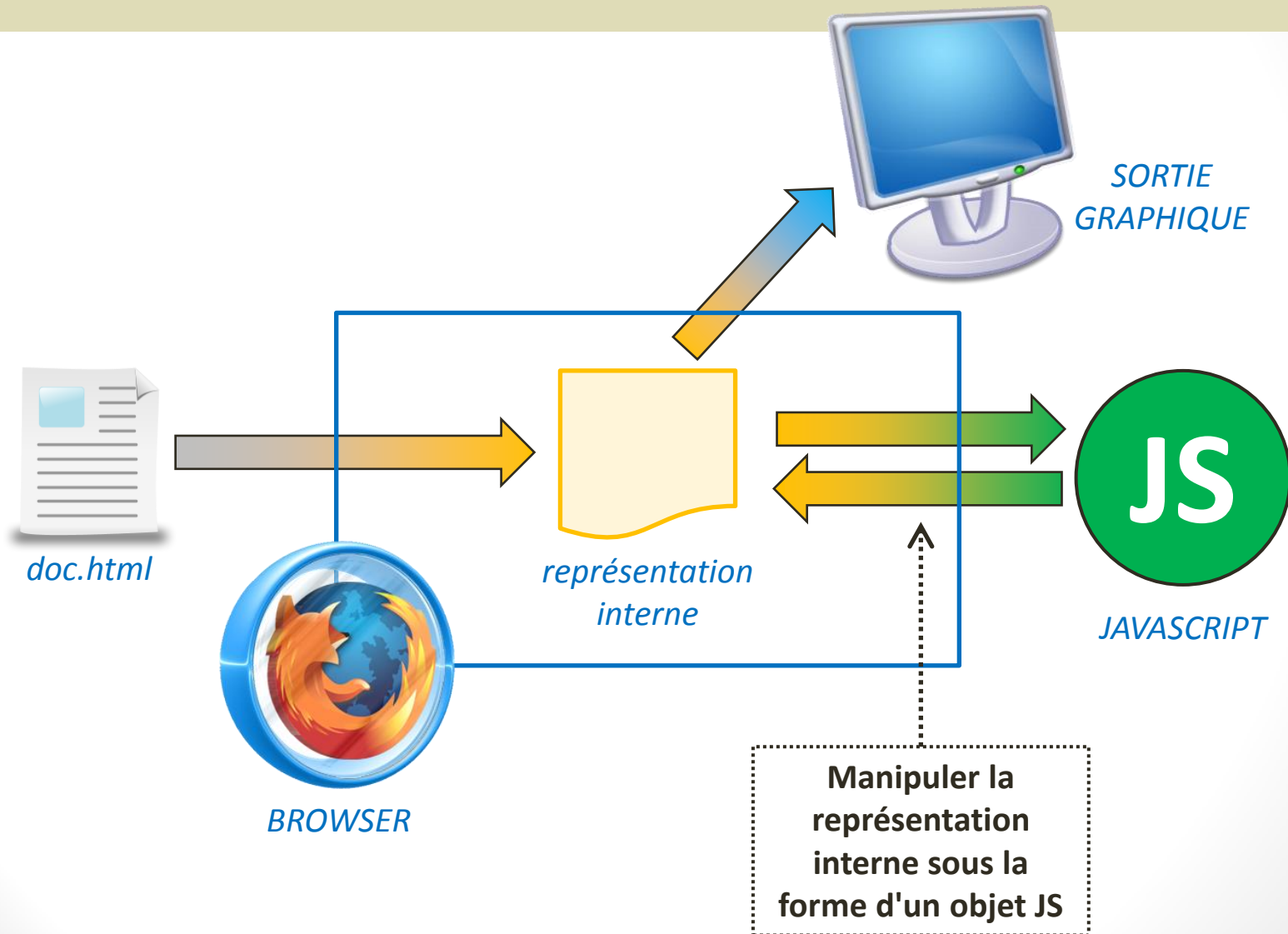
Document Object Model

HENALLUX

Technologies web

IG2 – 2015-2016

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)
- Le DOM ne se limite pas au document mais couvre également d'autres informations (fenêtre d'affichage, navigateur...).

DOM – késako ?


- Chaque **élément de l'arborescence HTML** peut être ciblé par une commande Javascript. On peut ainsi...
 - le supprimer ;
 - modifier son contenu ;
 - modifier son apparence (via les propriétés CSS) ;
 - ou encore lui ajouter de nouveaux sous-éléments.
- Le tout se présente sous la forme d'**objets**.
- Exemple déjà vu :

```
var para = document.getElementById("idParagraphe");  
para.innerHTML = "Nouveau texte à afficher !";
```

Le D.O.M.

- 
- **Le DOM et les éléments HTML**
 - Cibler un élément HTML, naviguer le DOM
 - **Actions sur les éléments HTML**
 - Modifier un élément HTML
 - Ajouter/supprimer des éléments HTML
 - Associer des actions à certains événements
 - **À la racine du DOM : window**
 - L'objet window

DOM et éléments HTML

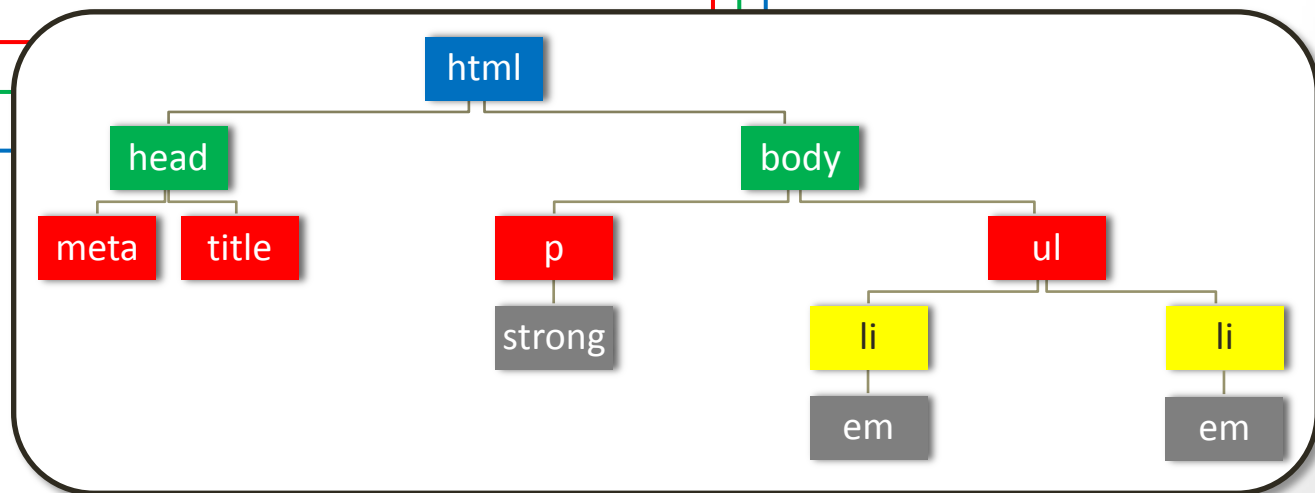
- 
- **Utiliser l'arborescence HTML en Javascript**
 - L'arborescence dans le DOM
 - Naviguer l'arborescence
 - **Autres méthodes pour cibler un élément HTML**

Ensuite : *Actions sur les éléments HTML*

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>
```

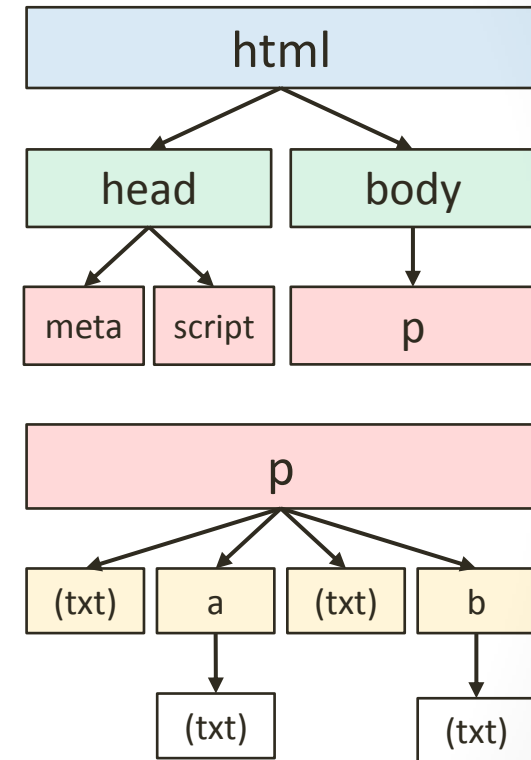


L'objet **document**

- 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>
```



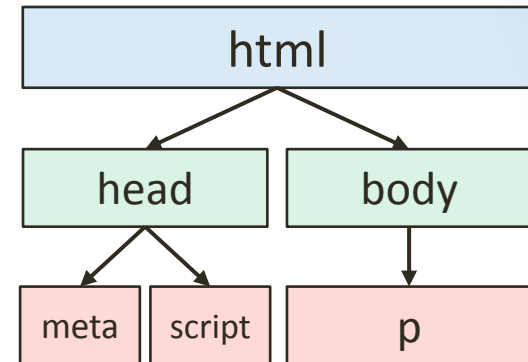
Note : version simplifiée, car on ignore les blancs (voir plus loin).

L'arborescence HTML

`document.childNodes[0]`

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

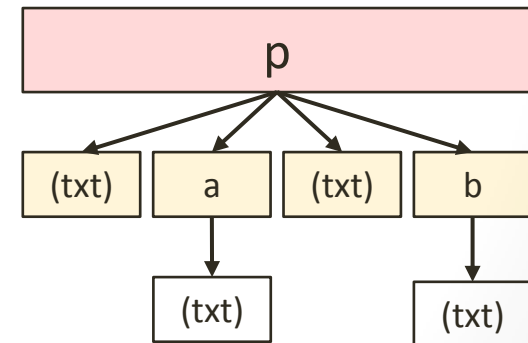
`document.childNodes[0].childNodes[1].childNodes[0]`



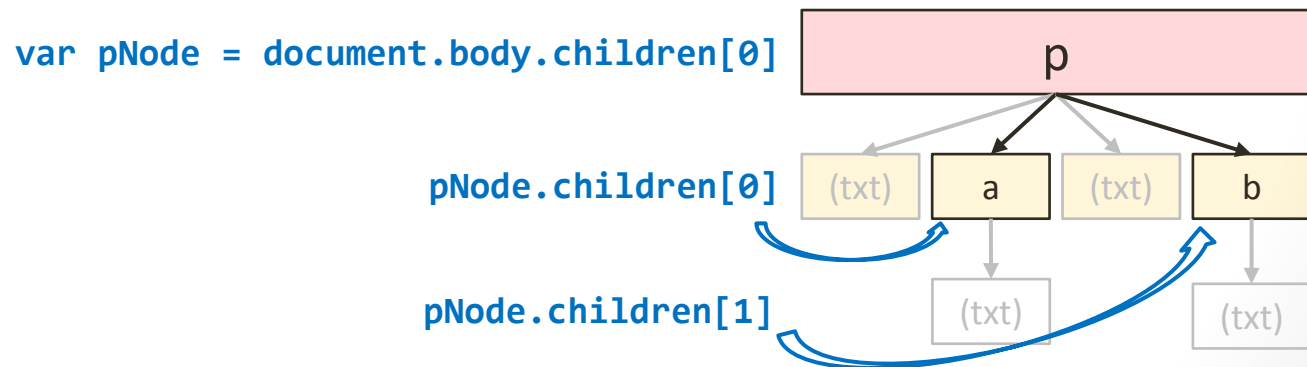
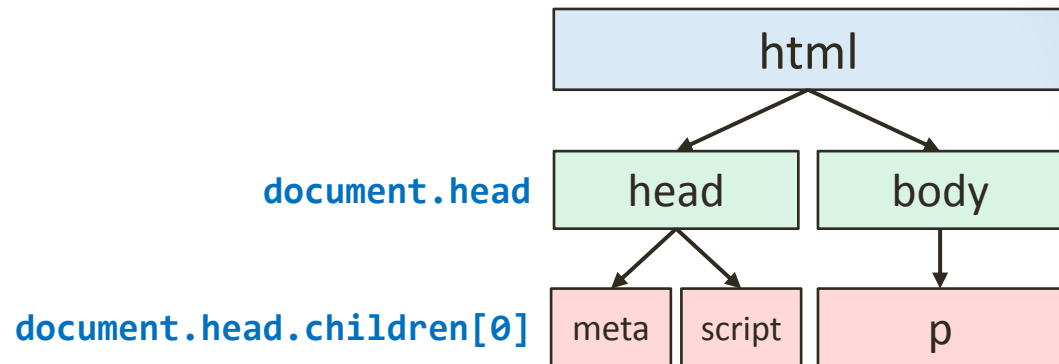
`var pNode = document.childNodes[0].childNodes[1].childNodes[0]`

`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>
```

Naviguer l'arborescence HTML

- Se déplacer 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`
- Se déplacer dans les autres directions
 - Vers le haut : `elem.parentNode`
 - Horizontalement : `elem.nextSibling`, `elem.nextElementSibling`, `elem.previousSibling`, `elem.previousElementSibling`

Comment cibler un élément ?

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

Comment cibler un élément ?

Méthode 1 : accès direct

- Voir slides précédentes
- L'objet `document` possède des propriétés qui permettent d'accéder directement à certains éléments :
 - `document.head`
 - `document.body`
 - `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`

Comment cibler un élément ?

Méthode #3 : recherche

- Selon **l'identificateur** : `document.getElementById(id)`
- Selon la **balise** ou la **classe** :
 - `document.getElementsByTagName(balise)`
 - `document.getElementsByClassName(classe)`

Ces deux méthodes renvoient une collection d'éléments qui peut être utilisée comme un tableau à index numérique ou comme un tableau associatif (basé sur l'id des éléments).

Ex : `var titres = document.getElementsByTagName("h1")`
... `titres[0]` ...
... `titres["main"]` ... si un titre h1 a pour id "main"
`document.getElementsByClassName("important")`

Comment 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 `document` mais également sur chacun des éléments.

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

Actions sur les éléments HTML

- 
- **Modifier un élément HTML**
 - Modifier son contenu
 - Modifier son apparence (propriétés CSS)
 - **Modifier l'arborescence**
 - Ajouter / supprimer / déplacer un élément
 - **Programmation événementielle**
 - Associer des actions à certains événements

Ensuite : *À la racine du DOM, window*

Modifier un élément HTML

- Pour le **contenu textuel** : `elem.innerHTML`
 - en lecture ou en écriture
 - *Note : `elem.textContent` renvoie le contenu sans les balises HTML.*
- Un **attribut** : 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 :
`var cible = lien.getAttribute("href");`
`lien.setAttribute("href", "http://www.henallux.be");`

Modifier le style d'un élément

- **Option #1 : via les classes CSS**, grâce à `elem.classList`
 - `elem.classList` : collection des noms des classes de l'élément (fonctionne en lecture comme un tableau)
 - `elem.classList.length`
 - `elem.classList[0] ...`
 - `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)

Modifier le style d'un élément

- **Option #2 : modifier directement le style**

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

- Remarque. Pour examiner le style d'affichage actuel :

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

Note : `elem.style.color` reflète uniquement le style inline ; si `elem` est écrit en rouge parce que sa classe le dicte, `elem.style.color` sera `undefined`.

- Liste des propriétés de style : voir par exemple

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

Ajouter un élément HTML

- **Étape #1** : création du nouveau nœud
 - `var nvNoeud = document.createElement(balise);`
`nvNoeud.innerHTML = ... ; nvNoeud.classList.add(...)`
 - `var nvNoeud = noeud.cloneNode()` pour copie superficielle
`var nvNoeud = noeud.cloneNode(true)` pour copie profonde
 - **Étape #2** : ajout du nœud dans l'arborescence en tant que fils d'un nœud existant
 - `noeud.appendChild(nvNoeud)` : ajoute comme dernier fils
 - `noeud.insertBefore(nvNoeud, fils)` : ajoute juste avant le fils
 - `noeud.replaceChild(nvNoeud, fils)` : remplace le fils cité
- Note : ces méthodes permettent aussi de déplacer un nœud existant.*

Supprimer un élément HTML

- Supprimer tous les fils d'un élément :

```
noeud.innerHTML = "";
```

- Supprimer un fils :

```
noeud.removeChild(fils);
```

Gestion des événements

- On peut associer des événements aux nœuds HTML :
 - 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`

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, false);
```

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.

3^e argument = quand accomplir l'action (voir plus loin)

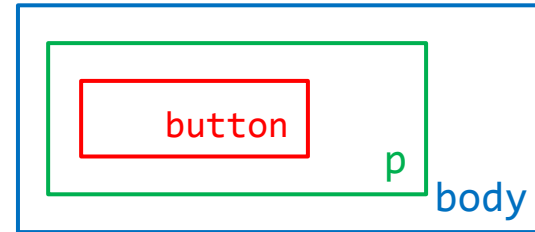
- (Dangereux) Si on ne lie qu'une seule action à un événement :

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

Gestion des événements

- **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

Gestion des événements

- Enlever une action

```
but.removeEventListener("click", fonction, false);
```

Il faut pouvoir faire référence à la fonction !

Exemple :

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

Gestion des événements

- Quelques propriétés et méthodes de l'objet événement
 - `evt.clientX`, `.clientY` : coordonnées de la souris dans la fenêtre
 - `evt.offsetX`, `.offsetY` : coordonnées de la souris dans l'élément
 - `event.keyCode` : code ASCII de la touche pressée
 - `event.target` : élément qui est à l'origine de l'événement
 - `event.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

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

À la racine du DOM : window

- 
- **L'objet window**
 - Modifier son contenu

L'objet window

- L'objet prédéfini `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`.
- Pour être plus précis, l'objet `window` est le contexte global. Tout ce qui est défini en Javascript est un attribut de `window`.
Ex :

```
var x = 3;           function go () {alert("go")};  
    alert(window.x);    window.go();
```
- On peut toujours omettre le préfixe `window`.
 - Exemple : `document` au lieu de `window.document`

L'objet 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...

L'objet window

- Quelques propriétés de l'objet `window`
 - `window.opener` : 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 !)

L'objet 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
 - `.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");
```

L'objet 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!") };  
var 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!") };  
var id = setInterval(crie, 3000);
```

jQuery

(pour information)

HENALLUX

Technologies web

IG3 – 2015-2016

jQuery en quelques mots

- **jQuery** est une bibliothèque définissant plusieurs fonctions utilitaires en Javascript permettant de :
 - manipuler le DOM (éléments HTML, événements...)
 - réaliser certains effets et animations standards
 - effectuer des échanges Ajax (voir plus loin),
 - le tout de manière indépendante des navigateurs !
- Rappel Javascript :

Les identificateurs

 - commencent par une lettre, **\$** ou **_** et
 - sont composés de lettres, de chiffres, de **\$** et de **_**
- jQuery tire parti du fait que **\$** est un identificateur acceptable.

jQuery en quelques mots

- Où trouver jQuery ?
 - <http://www.jquery.com>
 - Disponible en deux versions :
 - version de développement (code lisible)
 - version minimaliste (code illisible mais beaucoup plus court)
- Inclure jQuery (version locale)
`<script src="jquery-chez-moi.js"></script>`
- Inclure jQuery (version partagée)
`<script
 src="http://code.jquery.com/jquery-latest.pack.js">
</script>`
Pour tirer parti du cache (d'autres adresses standards existent)

Sélecteurs jQuery

- Version sans jQuery

```
document.getElementById("ident")
```

```
document.getElementsByTagName("h2")
```

```
document.getElementsByClassName("rouge")
```

- Version avec jQuery (sélecteurs utilisant la syntaxe CSS)

```
$("#ident")
```

```
$("h2")
```

```
$(".rouge")
```

```
$("p strong:first")
```

 : tous les premiers éléments "gras" des paragraphes

... et de nombreux autres ...

Actions jQuery

- Un sélecteur jQuery peut retourner un ou plusieurs objets, sur le(s)quel(s) on peut accomplir diverses actions.

Ex : `$("#cadre").hide(); $("h2").hide();`

- La plupart des actions jQuery peuvent être effectuées sur un élément ou sur une liste d'éléments.
- jQuery permet également d'enchaîner les actions (chaque action est une fonction qui renvoie l'objet sur lequel elle porte).

Ex : `$("#cadre").hide().show(); // cache puis affiche`

jQuery et DOM

- Obtenir des informations sur un élément
 - `.text()` : contenu textuel
 - `.html()` : contenu au format HTML (bases y compris)
 - `.val()` : valeur (d'un élément `<input>` par exemple)
 - `.attr("src")` : valeur de l'attribut "src"
- Donner une valeur / modifier un élément
 - `.text(contenu)`
 - `.html(contenu)`
 - `.val(valeur)`
 - `.attr(nom, valeur)`

jQuery et DOM

- Modifier plusieurs éléments

- Si la valeur est la même

- `$(".rouge").text("Nouveau contenu");`

- Si la valeur n'est pas forcément la même : on utilise comme argument une fonction qui

- reçoit le numéro d'un élément et son ancienne valeur
 - renvoie la nouvelle valeur.

- `$(".rouge").html(function (i,old) { return old.toUpperCase(); });`

jQuery et DOM

- Ajout d'éléments à l'arbre HTML
 - `.append(html)` ajoute l'argument à la fin du contenu de l'élément

```
$("p").append(" texte ajouté à la fin de chaque para");
$("#maliste").append("<li>Élément oublié</li>");
```
 - `.prepend(html)` ajoute au début du contenu de l'élément
 - `.before(html)` ajoute avant l'élément

```
$("img").before("<p>para ajouté avant chaque image</p>");
```
 - `.after(html)` ajoute après l'élément

Note : ces quatre méthodes peuvent prendre plusieurs arguments (plusieurs éléments à ajouter).

jQuery et DOM

- Suppression d'éléments de l'arbre HTML
 - `.remove()` enlève l'élément (et ses enfants)
 - `.empty()` enlève les enfants de l'élément

jQuery et CSS

- Manipulation des classes
 - `.addClass(classe)`
 - `.removeClass(classe)`
 - `.toggleClass(classe)`
 - `.hasClass(classe)`

Note : l'argument est une chaîne de caractères avec un nom de classe (ou plusieurs noms séparés par des espaces).

jQuery et CSS

- Manipulation des propriétés CSS (style inline)
 - `.css("nom")` donne la valeur d'une propriété CSS
 - `.css("nom", "valeur")` donne une valeur à une propriété CSS
- Ex : `$(".rouge").css("font-weight", "bold");`

Note : cela modifie les éléments (propriétés CSS inline), pas les classes !

Effets jQuery

- Afficher / cacher des éléments
 - `.hide()` et `.show()` : cacher/afficher les éléments
 - 1^{er} argument : "slow", "fast" ou nombre de millisecondes
 - 2^e argument : callback à exécuter à la fin
 - `.toggle()` : cache ou affiche selon l'état actuel
- Réaliser des fondus
 - `.fadeIn()`, `.fadeOut()`
 - 1^{er} argument : "slow", "fast" ou nombre de millisecondes
 - 2^e argument : callback à exécuter à la fin
 - `.fadeToggle()`
 - `.fadeTo(vitesse, opacité-cible, callback)`

Effets jQuery

- Modification progressive du CSS
 - `.animate({paramètres CSS}, vitesse, callback)`
- Ex : `$("#cadre").animate({height:'150px', opacity:0.5});`
`$("#cadre").animate({height:'+=50px'});`
- *Note : on peut enchaîner plusieurs animations (file d'attente)*

jQuery et les événements

- Fonctions pour associer un événement à un (ou plusieurs) élément(s) :
 - `.click()`, `.dblclick()` : clic et double clic
 - `.mouseenter()`, `.mouseleave()` : passage de la souris
 - `.load()` : après le chargement (d'une image, du document)
 - ... et bien d'autres ...
- Ces fonctions prennent pour argument la fonction à appeler.
`$("#monBouton").click(function () { alert("Clic !"); });`
- La fonction peut faire référence à l'élément via `$(this)`.
`$("p").mouseenter(function () {
 $(this).css("color", "red");
});`

jQuery et les événements

- Fonction combinant `onmouseenter` et `onmouseleave` :
 - `.hover(fonction1,fonction2)`
- Fonction à exécuter dès que toute la structure du DOM est en place :
 - `$(document).ready(fonction);`
 - Version abrégée : `$(fonction);`

```
$(function () { $("#bouton").click( ... ); });
```

- *Note : `.load(...)` s'exécute lorsque toutes les ressources (images par exemple) ont été chargées.*