DICJ

Département d'informatique

JavaScript et DOM

420-JQA-JQ

JavaScript

- JavaScript est un **langage de programmation de scripts** principalement utilisé dans les pages web interactives côté client.
- Depuis 2009, il est aussi utilisé côté serveur grâce à Node.js

Créer des scripts

- Les scripts JavaScript :
  - sont de simples **fichiers "texte" (dont l'extension est .js) à créer** avec un **éditeur de texte.**
  - sont **intégrés au sein des pages web.**
  - sont exécutés **côté client par le navigateur web** ou **côté serveur grâce à l'application Node.js**.

Pourquoi ?

- Dans le navigateur, JavaScript sert généralement:
  - à contrôler les données saisies dans des formulaires HTML
  - à interagir avec le document HTML via l'interface **DOM** (*Document Object Model)* fournie par le navigateur
  - à modifier le contenu des pages web par programmation avec la méthode **Ajax** *(Asynchronous JavaScript And XML)*

Variables

- JavaScript est sensible à la casse employée.
  - **variabletest** et **Variabletest** désignent deux variables différentes.
- Il existe un certain nombre de noms réservés qui ne peuvent pas être utilisés comme nom de variable ou de fonction:
  if, **for**, **while**, **var**, **else**, …

Types de données

- JavaScript comporte 10 types:
  - Object
  - String
  - Number
  - boolean
  - Array
  - Date
  - RegExp
  - Function
  - Undefined
  - Null

Déclaration et affectation

- Une déclaration (sans affectation de valeur) se fait à l'aide du mot-clef **var**.
- **var a**; déclare l'existence d'une variable s'appelant **a**, mais sans lui avoir affecté de valeur. Elle est donc de type **undefined** et a comme valeur **undefined**.
- **var a; //Type et valeur undefined**
- **a="Ceci est une chaîne"** ; **//Type String et valeur « Ceci est une chaîne »**
- **a=b=c=d=e=5; //Toutes des Number ayant pour valeur 5**
- **var b = 6; //Type Number et valeur 6**

Typage faible et dynamique

- En JavaScript, on n'a pas besoin d'indiquer le type d'une variable. La résolution se fait à l'exécution.

```
1   //Un entier
2   var entier = 5;
3
4   //Une chaîne de charactère
5   var chaine = "Bonjour!";
6
7   //Un booléen
8   var bool = true;
9
10  alert(entier);
11  alert(chaine);
12  alert(bool);
```

```
15  bool = entier + bool;
16  entier = entier * bool;
17  entier = entier + chaine;
18  alert(entier);
```

Si **true** équivaut à 1,
qu'affichera le **alert**?

Tableaux

```
//Déclaration d'un tableau de 4 éléments
var tableau1 = new Array(4) ;
//Déclaration d'un tableau dont le nombre d'éléments est a priori inconnu
var tableau2 = new Array() ;

var tableau1 = new Array(4) ;
tableau1[0]="Beurre" ;
tableau1[1]="Confiture" ;
tableau1[2]="Pain" ;
tableau1[3]="Jus de fruit" ;

// Syntaxe alternative
var autre_tableau=["Beurre", "Confiture", "Pain", "Jus de fruit"] ;

var encore_un_autre_tableau= new Array('Beurre', 'Confiture', 'Pain', 'Jus
de fruit') ;
```

Tableaux multidimensionnels

```
var tableau1 = new Array(4) ;
for (i=0;i<tableau1.length;i++)
{
   tableau1[i] = new Array(2) ;
}
```

Portée d'une variable

- La portée d'une variable désigne l'ensemble du code dans lequel elle peut être utilisée.
- Si une variable est déclarée sans le mot-clef **var**, elle peut être utilisée n'importe où dans le script. On l'appelle alors **variable globale**.
- Si une variable est déclarée avec le mot-clef **var**, elle ne peut être utilisée que dans le bloc où elle se trouve. On l'appelle alors **variable locale**.

Exemple de portée

```
var a = 8 ;

function testFonction1()
{
   var pi = 3.14 ;
   (...) ;
}

function testFonction2()
{
   (...) ;
}
```

- La variable **a** peut être utilisée dans les fonctions **testFonction1** et **testFonction2**, mais la variable **pi** ne peut être utilisée que dans la fonction **testFonction1**.

If … else if … else

- Les conditions s'utilisent de la même façon en JavaScript qu'en Java ou en C#

```
if(age < 18){
  alert("Pas le droit d'entrer");
} else if(age === 18){
  alert("Salut le nouveau");
} else{
  alert("Bienvenue");
}
```

Opérateurs

- + : addition ou concaténation
- -, /, *, %
- <, <=, >, >=, ==, !=, ===, !==
- &&, | |
- !, ++, --

- Privilégiez
  === et !==
  à
  == et !=

Switch

```
switch (choix)
{
case 1:
  alert("Vous avez fait le premier choix") ;
  break ;
case 2:
  alert("Vous avez fait le deuxième choix") ;
  break ;
case 3:
  alert("Vous avez fait le troisième choix") ;
  break ;
default :
  alert("Vous avez fait un choix au moins égal à 4") ;
}
```

Les boucles

- JavaScript possède quatre types de boucle, soit **for**, **for…in**, **while** et **do…while**.
- **for:** boucle utilisée lorsque l'on connait le nombre d'iterations que l'on veut faire. S'écrit sous la forme:
  - for(**variable numeric**; **condition d'arrêt**; **bond**)

    {

    code à exécuter

    }

```
1  for(var cptFor = 0; cptFor < 10; cptFor ++)
2  {
3      //affiche les nombres de 0 à 9
4      alert(cptFor);
5  }
```

Les boucles

- **for…in**: boucle permettant de parcourir chacun des éléments d'un tableau ou chacune des clés d'un objet. S'écrit sous la forme:
  - for (**variable** in **objet ou tableau**)
    {
    code à exécuter
    }

```
for(var elem in tableau){
  console.log(elem);
}
for(var key in objet){
  console.log(key + ": " + objet[key]);
}
```

ATTENTION, LORSQU'UTILISÉ AVEC UN TABLEAU, ÇA VA RETOURNER LES INDICES NON-NULL ET NON LES VALEURS!

Les boucles

- **while:** boucle utilisée lorsque l'on ne connait pas le nombre d'iterations que l'on veut faire. S'écrit sous la forme:
  - while(**condition d'arrêt**)
    {
        code à exécuter
    }
- **do…while:** boucle utilisée lorsque l'on ne connait pas le nombre d'iterations que l'on veut faire, mais qu'on sait qu'elle doit se faire au moins une fois. S'écrit sous la forme:
  - do
    {
        code à executer
    }while(**condition d'arrêt**)

Exemple While et do…while

```
var chaine;

do {
        chaine=prompt("Entrez une chaîne de caractères contenant le
        caractère \"");
}while (chaine.indexOf("\"")==-1);

alert("La chaîne entrée est \""+chaine+"\"");

var chaine2;

while ( chaine2.indexOf(("\"")==-1)
{
        chaine2=prompt("Entrez une chaîne de caractères contenant le
        caractère \"");
}

alert("La chaîne entrée est \""+chaine2+"\"");
```

Fonctions

- En JavaScript, les fonctions sont un type de données. Elles peuvent être passées en argument à d'autres fonctions et être mises dans des variables.
- Il existe deux catégories de fonctions:
  - Fonction nommée;
  - Fonction anonyme;

Fonction nommées

- S'écrit sous la forme:
  function ma_fonction(arg1, arg2){ … }

```
function surfaceCercle()
{
    var entree=prompt("Entrez le rayon du cercle :  ");
    var rayon = parseFloat(entree);

    return 3.14*rayon*rayon;
}
```

Fonctions anonymes

- Fonction anonyme
  function(arg1, arg2) { … }

```
var uneFonction = function(x, y)
                  {
                    return x + y;
                  };
```

```
function(x, y)
{
    return x + y;
};
```

CAUSE UNE ERREUR!

Exemple de fonctions

```
var a=3;
var b=-2;

function multiplie(x)
{
        return 3*x;
}

function affiche()
{
        alert(multiplie(a));
        alert(multiplie(b));
}
```

Exemples de fonctions

```
function times(n, action){
  for(var i = 0; i < n; i++)
  {
    action(i);
  }
 }

times(10, function(iteration)
          {
              console.log("Hello world!"  + iteration);
          });
```

```
var array = ['a','b','c'];

array.forEach(function(element)
              {
                  alert(element);
              });
```

Les objets

- Les objets sont comme des tableaux, sauf que les index sont des chaînes de charactères au lieu d'être des chiffres. Ils permettent donc d'associer une chaîne de caractères à une valeur et d'organiser le code.
- S'écrit sous la forme:
  - var nomDeLObjet =

```
{
    clé: valeur,
    clé2: valeur,
    clé3: fonction anonyme,
    etc.
}
```

Exemple d'objet

```
var john =
{
    first_name: "John",
    last_name: "Doe",
    age: 40,
    pets: ["Mickey", "Donald", "Donatello"]
    say: function(message){
        console.log(this.first_name + " says " +
message)
    }
}
```

Un attribut

Une méthode

Exemple d'objet (suite)

- Ces deux lignes de code sont équivalentes. Il est donc possible d'utiliser un objet comme un tableau, ou via le « . »

```
john.say(john.age);

john["say"](john["age"]);
```

Utiliser un objet ou un tableau?

- Un tableau devrait être utilisé quand:
  - Il faut stocké plusieurs données liées ensemble;
  - La clarté de l'index n'a pas d'importance;
  - Les données sont toutes du même type.
- Un objet devrait être utilisé quand:
  - Les données misent ensemble représente une entité;
  - Chaque donnée stocké doit être bien identifiée.
- Feriez-vous un tableau ou un objet?
  - Une classe de plusieurs élèves;
  - Un personnage de jeux vidéo;
  - Un film;
  - Votre bibliothèque de film.

Objets usuels

- String
  - indexOf, lastIndexOf, charAt , toUpperCase, toLowerCase, substring, substr, split, concat, …
- Math
  - PI, LN2, LN10, SQRT2, …
  - abs, cos, sin, tan, ceil, floor, round, log, max, min, pow, sqrt, random, etc…
- Date
  - getFullYear(), getDay(), getMonth(), getMinute() etc...

For…in

- La boucle **For…In** permet d'itérer à travers toutes les clés d'un objet. Il n'est donc pas nécessaire de connaître toutes les clés pour afficher l'objet au complet.

- Exemple:

```
for(var key in john)
{
    alert(key + " is " + john[key]);
}
```

L'outil de développement

- Habituellement atteignable via la touche "F12" ou le menu avancé ou le clic droit de la souris
- Très pratique pour:
  - Visualiser la page web
  - Déboguer
  - Voir les erreurs
  - Tester du code JS, HTML ou CSS

Web dynamique…

- Utiliser JavaScript pour modifier le document côté utilisateur
- Le **Modèle Objet de Document**, ou **DOM** est un outil permettant l'accès aux documents HTML et XML.
  - Il fournit une représentation structurée du document ;
  - Il codifie la manière dont un script peut accéder à cette structure.
- Il s'agit donc, essentiellement, d'un moyen de lier une page Web, par exemple, à un langage de programmation ou de script.

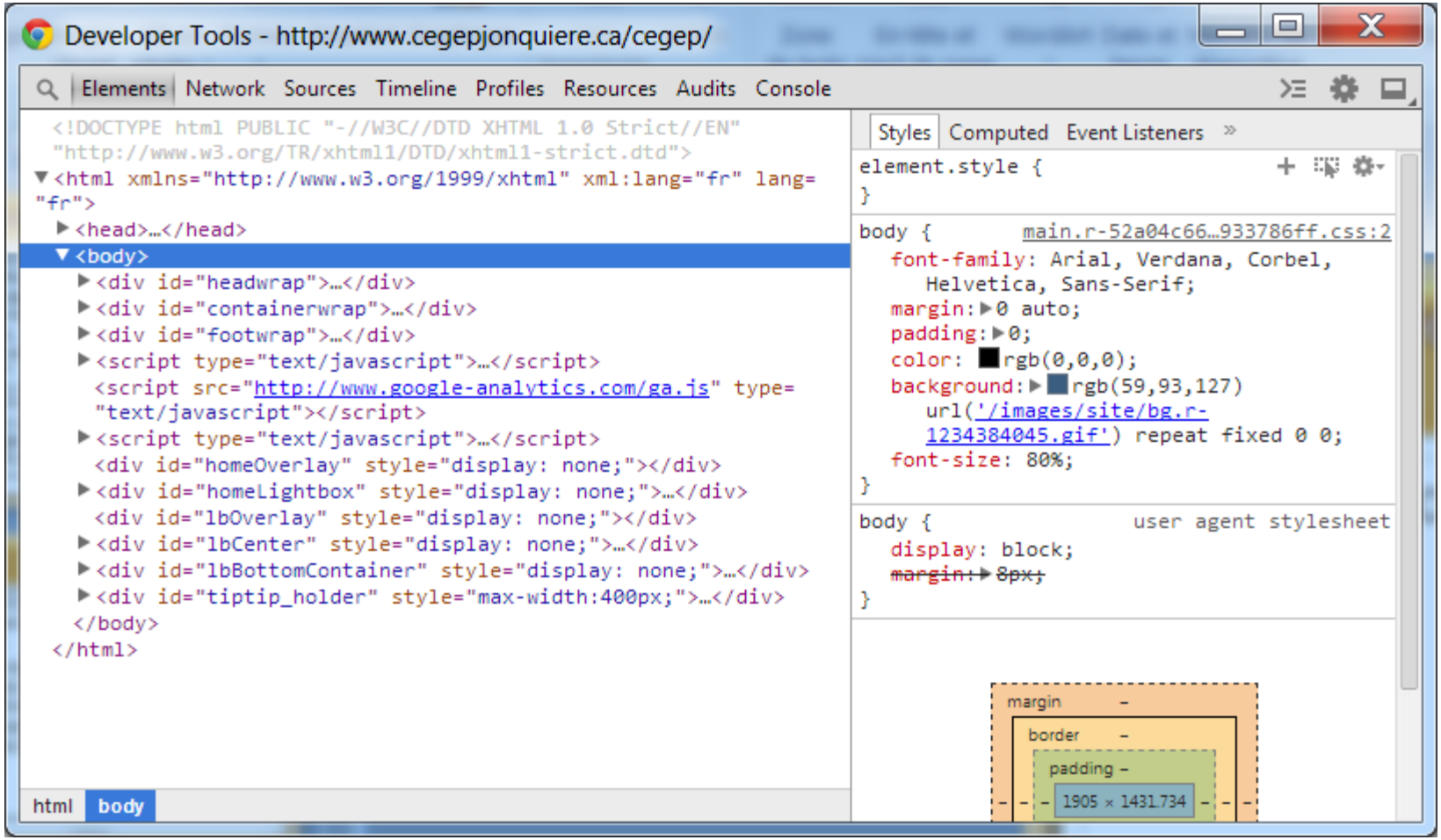
DOM

- DOM : structure d'arbre représentant le document et présente dans la mémoire du navigateur.
- DOM : API (*Application Programming Interface*) qui nous fournit les moyens d'interagir avec l'arbre-document. C'est une recommandation du W3C pour gérer le contenu de documents XML, HTML en particulier.

Le DOM pourquoi faire ?

- Rendre visible/invisible une partie du document,
- Modifier un élément de style de la page,
- Changer une image,
- Remplissage automatique de formulaires ou test de validité des données saisies dans un formulaire avant envoi au serveur, etc.
- **Notons que, dans tous les cas, les identifiants HTML (attributs ID) vont jouer un rôle crucial pour repérer les nœuds de l'arbre.**

Une arborescence ou un ensemble de nœuds



Accès aux éléments

- L'objet **document** est un objet qui représente l'ensemble de l'arborescence du document
  - **getElementById** permet de sélectionner un élément d'identifiant donné dans une page.
    - Par exemple, si on a dans la page (...) , document.getElementById("intro") permettra de sélectionner précisément l'élément p en question.
  - **getElementsByName** permet de sélectionner les éléments portant un nom donné dans une page.
  - **getElementsByTagName** permet de sélectionner les éléments portant un nom de balise donné dans une page.
  - **getElementsByClassName** permet de sélectionner les éléments portant un classe CSS donnée dans une page.

getElementById

- Accéder au contenu grâce à **`innerHTML`**

```
<!DOCTYPE html>
<html>
<head>
<meta content="text/html; charset=utf-8" >
<title>getElementById</title>
</head>
<body>
<p id=para1 > Mon premier paragraphe!<p>
</body>
</html>

<script>
var p = document.getElementById("para1");
alert(p.innerHTML); // Mon premier paragraphe
</script>
```

Accès à un élément via sélecteur

- **`document.querySelectorAll(…)`**
  - Tableau d'éléments de DOM avec le sélecteur
  - Possibilité de restreindre une partie de l'arborescence

```
<!DOCTYPE html>
<html>
<head>
<meta content="text/html; charset=utf-8" >
<title>Selecteur</title>
</head>
<body>
<p id=para1 >
<em> Mon premier paragraphe ! </em>
<em> et des élémnets em </em>
<p>

<p id=para2 >
<em> Mon deuxième paragraphe ! </em>
<em> et des élémnets em </em>
<p>
</body>
</html>
```

document.querySlectorAll(…)

```
<script>
var p = document.getElementById("para2");
var ems = p.querySelectorAll("em:first-child");
alert(ems[0].innerHTML); // Mon deuxième
paragraphe!
</script>
```

- Pour retourner tous les éléments

```
var ems = document.querySelectorAll("em:first-child");
```

- `:nth-child, :last-child, …`

Évènement de Javascript

- **load**, **unload** : événements déclenchés à l'arrivée et au départ de la page,
- **click**, **mousedown**, **mouseup**, **mousemove**, **mouseover**, **mouseout** : événements associés aux clics et déplacements de la souris,
- **keypress**, **keydown**, **keyup** : événements provoqués par l'appui d'une touche au clavier,
- **submit**, **change** : événements associés à la manipulation d'un formulaire par l'utilisateur.
- **Abort**, **Error**, **Move**, **Resize**, **KeyPress**, **KeyUp**, **DblClick**, **MouseDown**, **MouseUp**, **MouseMove**, **Reset**, ...

Évènement

- Pour gérer un évènement en JavaScript, il faut installer un **gestionnaire d'événement :**
  - Un gestionnaire d'événement sera l'action déclenchée automatiquement lorsque l'évènement associé se produit.
  - La syntaxe courante est la suivante : **`onEvenement=fonction()`** où **`Evenement`** est le nom de l'évènement géré.

Exemple Évènement

```
<!DOCTYPE html>
<html>
<head>
<meta content="text/html; charset=utf-8" >
<title>Bouton</title>
</head>
<body>

<button onclick="alert('Vous avez bien cliqué ici !')">Cliquez ici</button>

</body>
</html>
```

Suppression élément

```
<!DOCTYPE html>
<html>
<head>
<meta content="text/html; charset=utf-8" >
<title>Suppression Élément</title>
</head>
<body>
<p> Cliquer ici pour supprimer 1</p>
<p> Cliquer ici pour supprimer 2</p>
<p> Cliquer ici pour supprimer 3</p>
<p> Cliquer ici pour supprimer 4</p>
<p> Cliquer ici pour supprimer 5</p>
<p> Cliquer ici pour supprimer 6</p>
<p> Cliquer ici pour supprimer 7</p>
<p> Cliquer ici pour supprimer 8</p>
<p> Cliquer ici pour supprimer 9</p>
<p> Cliquer ici pour supprimer 10</p>

</body>
```

Exemple Suppression (suite)

```
<script>
var ps = document.querySelectorAll('p');

for(var i in ps)
{
        i.onclick = function()
        {
                this.parentNode.removeChild(this);
        }
}
</script>
```

Gérer des listeners

- Intercepter des évènements
- Intercepter un **`onclick`** sur un paragraphe

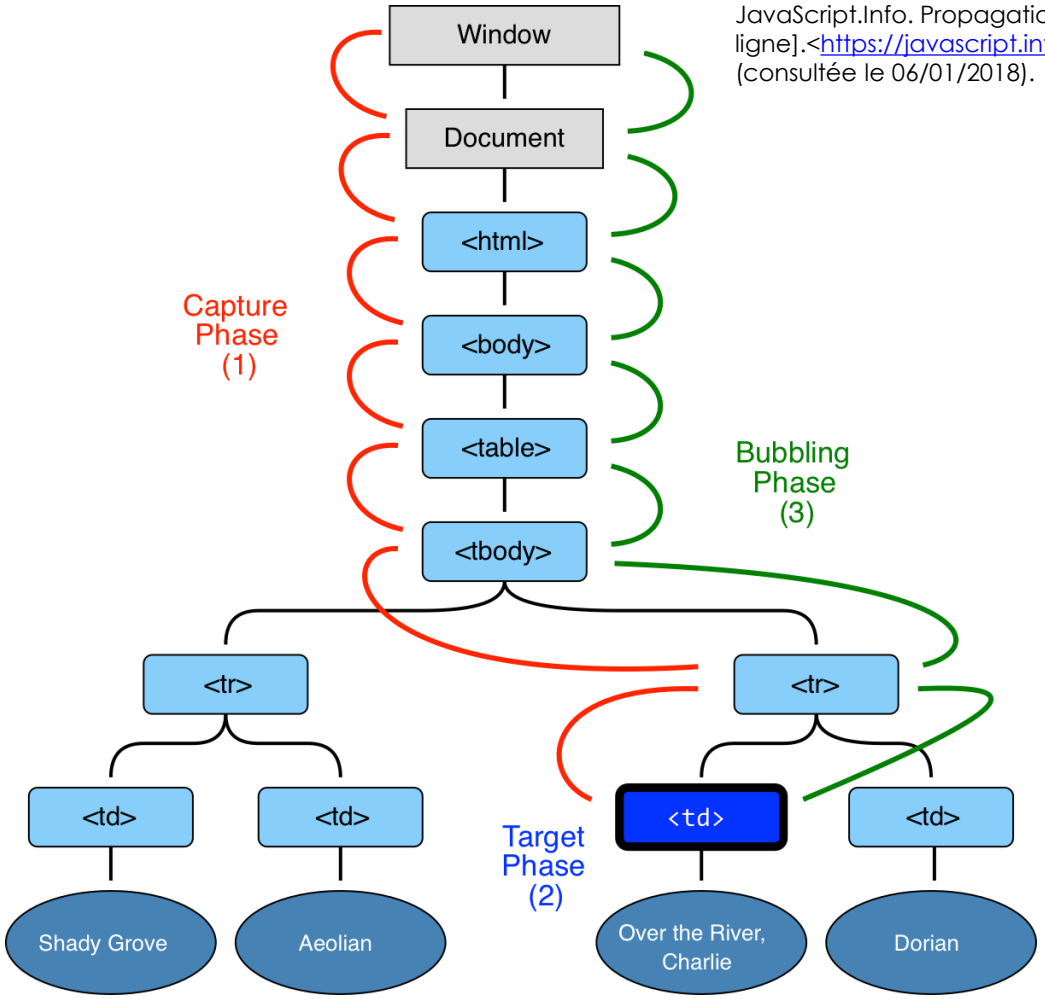
```
<p onclick = "this.parentNode.removeChild(this);"> Cliquer ici pour supprimer 10</p>
```

- Mais si je veux ajouter un autre traitement ?
- Utiliser la fonction : **`addEventListener()`**

addEventListener

- addEventListener(**évènement, fonction, utiliserLaCapture (optionnel)**);
  - Évènement: une « string » représentant le nom de l'évènement à ajouter;
  - Fonction: spécifie la fonction à exécuter quand l'évènement se produit
  - utiliserLaCapture: Booléen optionnel déterminant si la fonction s'exécute lors de la phase de « bubbling » (**false**, valeur par défaut) ou lors de la phase de « capturing » (**true**).
- Pour arrêter la propagation, utiliser: **event.stopPropagation()**

« Bubbling phase » vs « capturing phase »



JavaScript.Info. Propagation d'un évènement. [mage en ligne].<https://javascript.info/bubbling-and-capturing> . (consultée le 06/01/2018).

Exemple

```
<!DOCTYPE html>
<html>
<head>
<meta content="text/html; charset=utf-8" >
<title>Evenement</title>
</head>
<body>
<p id=para1 onclick = "this.parentNode.removeChild(this);"> Cliquer ici pour supprimer le paragraphe</p >

</body>
</html>
```

Exemple (suite)

```
<script>
document.getElementById(id).addEventListener("click", function(event)
{
        var p = document.createElement("p");
        p.innerHTML = "<em> Mon deuxième paragraphe ! </em> "+
                        "<em> et des éléments em </em>";
        p.id = "para2";
        document.body.appendChild(p);
});
</script>
```

Exemple Move

```
<script>
var position = document.getElementById('plan');

    document.addEventListener('mousemove', function(e)
        {
                position.innerHTML = 'Abscisse X : ' + e.clientX +
                'px<br />Ordonné Y : ' + e.clientY + 'px';
        }, false);
</script>
```

Création d'un élément

- Deux étapes
  - Créer l'élément

    `document.createElement(nom_Balise)`
  - Ajouter l'élément comme fils d'une autre balise

    `parent.appendChild(Fils)`

    `parent.insertBefore(Fils,frère)`

Exemple insertion

```
<!DOCTYPE html>
<html>
<head>
<meta content="text/html; charset=utf-8" >
<title>Insertion Élément</title>
</head>
<body>
<p id=para1 >
<em> Mon premier paragraphe ! </em>
<em> et des élémnets em </em>
<p>
</body>
</html>


<script>
var p = document.createElement("p");
p.innerHTML = "<em> Mon deuxième paragraphe ! </em> "+
                                "<em> et des élémnets em </em>";
p.id = "para2";
document.body.appendChild(p);
</script>
```

Modifier le contenu

```
<script>
var p = document.getElementById("para2");
var ems = p.querySelectorAll("em:first-child");
var str = "Mon deuxième paragraphe que je viens de
remplacer" + "<br>";

ems[0].innerHTML += str; // Mon deuxième paragraphe!
</script>
```

Autres possibilités

- **`cloneNode(bool)`** : avec ou sans les fils
- **`replaceChild(nœud1, nœud2)`**
- **`hasChildNodes()`**
- **`insertBefore(nouveauNoeud, noeudExistant)`**