Le DOM : Rappels et compléments

9 octobre 2025

Le DOM

- DOM : Document Object Model
- Interface de programmation standardisée par le W3C.
- La version actuelle est le DOM niveau 3
- Représentation d'un document HTML sous la forme d'un ensemble d'objets.
- L'imbrication des éléments est modélisée par un arbre.
- Le langage javascript utilise le DOM et peut modifier le document présent dans le navigateur en ajoutant ou en supprimant des nœuds de l'arbre.
- Les nœuds peuvent être de différents types, principalement des **nœuds éléments** et des **nœuds texte**.

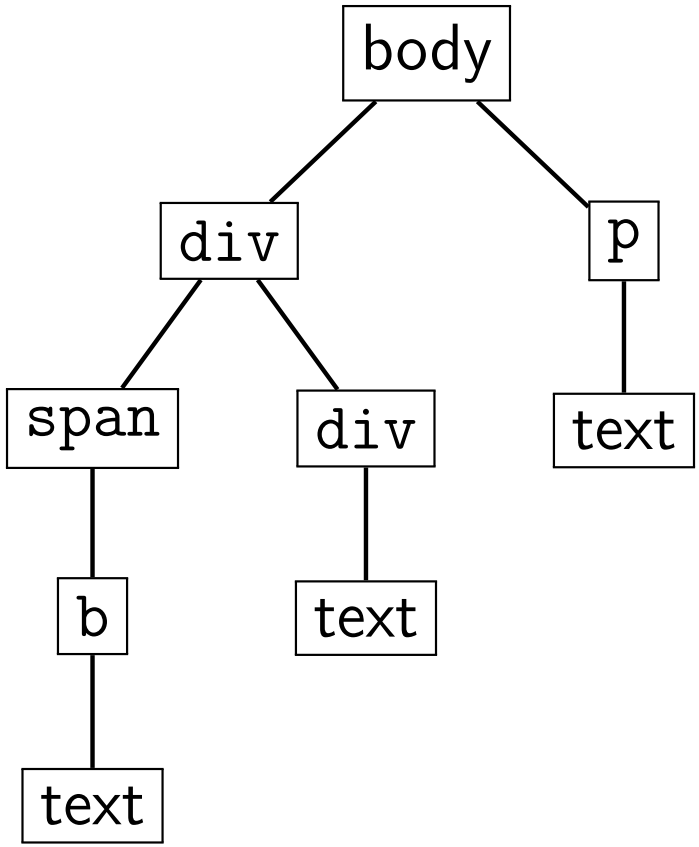
Le DOM

Par exemple, le fragment d'HTML suivant :

```
1  <body>
2     <div id="uneZone">
3        <span><b>le texte du span</b></span>
4        <div id="uneAutreZone">
5           Le texte du div
6        </div>
7     </div>
8     <p>
9        Le texte du paragraphe
10    </p>
11 </body>
```

Le DOM

peut être représenté par :



Accéder à des nœuds particuliers

- La fonction `getElementById` de l'objet `document` permet de récupérer un élément du DOM à partir de la valeur de son id s'il en possède un.
- Par exemple, pour le `div` de plus haut niveau du fragment d'HTML précédent :

```
1 let element=document.getElementById("uneZone");
```

- La variable `element` est de type `object`.
- Il est alors possible de manipuler ses propriétés, par exemple :

```
1 element.style.borderStyle = "solid";
```

Accéder à des nœuds particuliers

- La fonction `getElementsByTagName` de l'objet `document` permet de récupérer les éléments d'un document par le nom de balise.
- Par exemple, dans le fragment d'HTML précédent :

```
1 lesDiv = document.getElementsByTagName("div");
```

- La variable lesDiv contient un tableau de la taille du nombre d'éléments `div` présents dans le document, ici 2.
- Chaque cellule contient un objet modélisant un `div` du document.
- Par exemple, pour changer la couleur du `div` de plus bas niveau :

```
1 lesDiv[1].style.color = "red";
```

Accéder à des nœuds particuliers

De meme :

- La fonction `getElementsByName` de l'objet `document` permet de récupérer les éléments d'un document par la valeur de l'attribut `name`, s'il existe.

```
1 lesElements = document.getElementsByName("
      laValeurDuName");
```

- La fonction `getElementsByClassName` de l'objet `document` permet de récupérer les éléments d'un document par la valeur de l'attribut `class`, s'il existe.

```
1 lesElements = document.getElementsByClassName("
      laValeurDeLaClasse");
```

Accéder à des nœuds particuliers

Depuis peu :

- La fonction `querySelector` de l'objet `document` permet de récupérer le premier élément d'un document correspondant à un ensemble de sélecteurs CSS placés en paramètre de la fonction dans une chaine de caractères.

```
1 element = document.querySelector("div span");
```

- La fonction `querySelectorAll` de l'objet `document` permet de récupérer tous les éléments d'un document correspondant à un ensemble de sélecteurs CSS placés en paramètre de la fonction dans une chaine de caractères.

```
1 lesElements = document.querySelectorAll("div
      span");
```

Accéder à des nœuds particuliers

```
1 lesElements=document.querySelectorAll("div span");
```

Attention, la fonction `querySelectorAll` ne renvoie pas un tableau d'éléments mais un objet de type `nodeList` (liste de nœuds)

- La propriété `length` donne la taille de la liste.

```
1 let taille = lesElements.length;
```

- La propriété `item` permet d'accéder à un élément de la liste par son indice.

```
1 let element = lesElements.item(i); // élément
    n°i de la liste
```

Ajouter des nœuds

- Pour ajouter un nœud élément, on utilise la fonction `createElement` de l'objet `document`. Par exemple, pour créer un nœud correspondant à un élément `div` :

```
1 let newNoeud = document.createElement("div");
```

- Pour ajouter un nœud texte, on utilise la fonction `createTextNode` de l'objet `document`. Par exemple :

```
1 let newText = document.createTextNode("le texte
      du noeud");
```

- Il faut ensuite attacher le nœud à son parent dans l'arbre du DOM,
  - soit après tous les autres enfants avec la fonction `appendChild`
  - soit avant un autre enfant précisé en paramètre de la fonction `insertBefore`

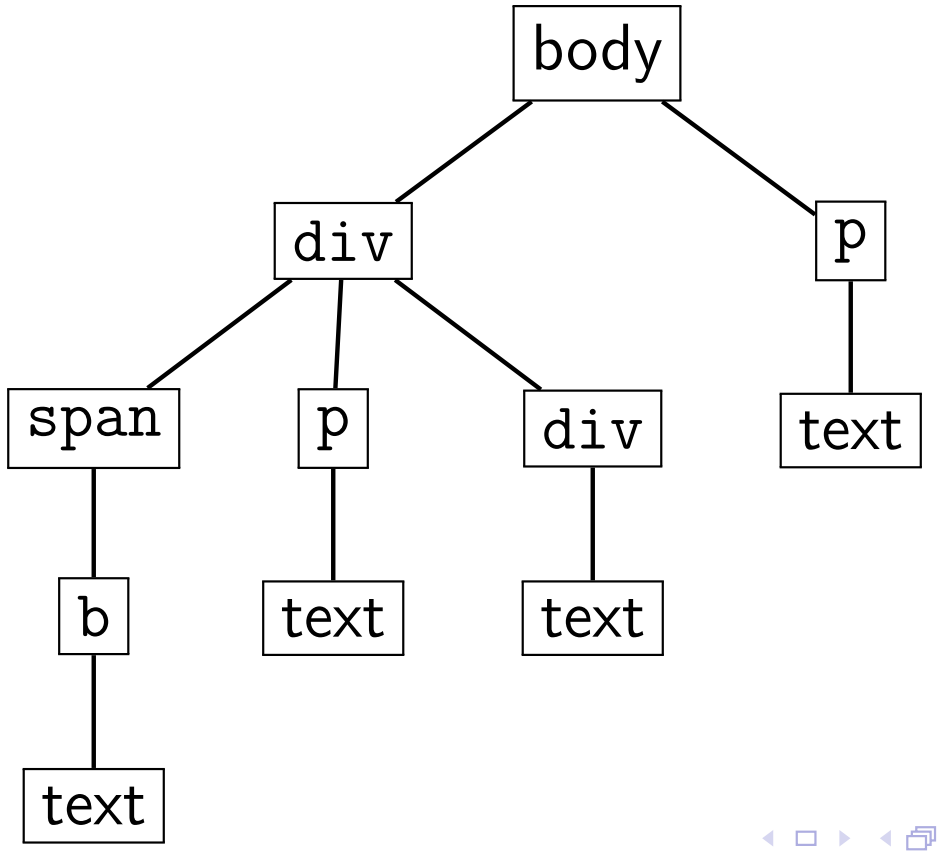
Ajouter des nœuds

Par exemple, dans l'arbre qui nous a servi d'exemple précédemment, on veut ajouter dans le `div` possédant l'id "uneZone" un paragraphe contenant du texte et juste avant le `div` possédant l'id "uneAutreZone" :

```
1 let newP = document.createElement("p");
2 let texte = document.createTextNode("Le texte du
    paragraphe");
3 newP.appendChild(texte);
4 let parent = document.getElementById("uneZone");
5 let enfant = document.getElementById("uneAutreZone"
    );
6 parent.insertBefore(newP,enfant);
```

Ajouter des nœuds

Le résultat :



Supprimer des nœuds

Pour supprimer un nœud, on utilise la fonction `removeChild` de l'objet correspondant au parent :

```
1  parent.removeChild(enfant);
```

Manipulation des attributs

- On peut ajouter un attribut à un élément en utilisant la fonction `setAttribute` de l'objet référençant l'élément en question. Exemple :

```
1 element.setAttribute("nomAttribut","
      valeurAttribut");
```

- On peut retirer un attribut en utilisant la fonction `removeAttribute`. Exemple :

```
1 element.removeAttribute("nomAttribut");
```

Qu'est-ce qu'un événement ?

- Un événement est un **changement d'état** de l'environnement qui peut être intercepté par le code JavaScript.
- Ce changement d'état peut être provoqué par l'utilisateur (pression d'une touche, clic de souris, validation d'un formulaire, ...), par le document (chargement d'une image, ...), ou même par le développeur lui-meme.

L'objet `Event`

- Un objet `Event` est créé à chaque fois qu'un événement survient.
- Ses propriétés permettent de décrire l'événement en question.
- Cet objet se propage dans l'arbre du DOM selon 3 phases.
- Les propriétés suivantes sont communes à tous les objets `Event` :
  - L'objet `target` fait référence à la cible de l'événement.
  - La fonction `stopPropagation` permet de stopper la propagation de l'événement dans l'arbre du DOM.
  - La fonction `preventDefault` empeche l'action normalement prévue de se dérouler.

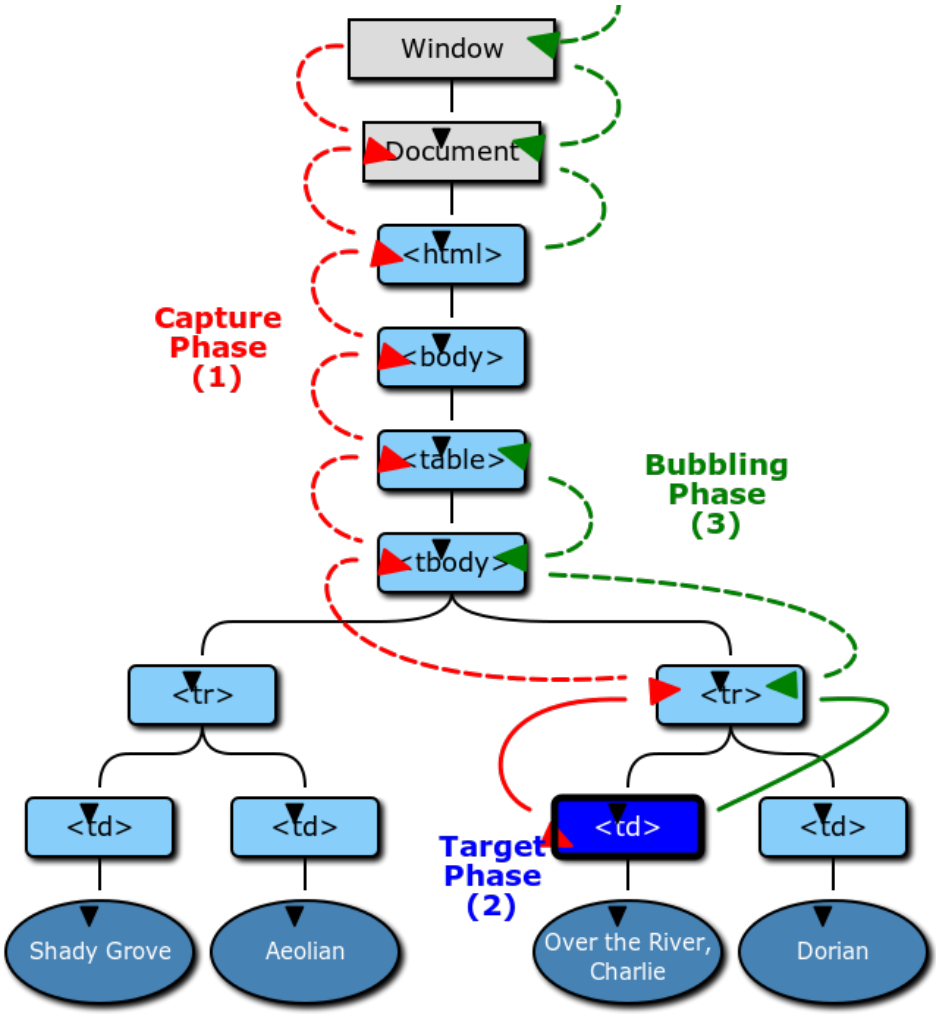
La propagation des événements

L'objet de type `Event` se propage dans l'arbre du DOM selon 3 phases déterminées par la cible de l'événement :

- La **capture** : propagation en descendant dans l'arbre de la racine (incluse) à la cible (exclue)
- La **cible** : l'événement atteint la cible
- Le **bouillonnement** propagation en remontant de la cible (exclue) à la racine (incluse).

Attention, certains événements ne bouillonnent pas (exemple : `load`).

La propagation des événements : les 3 phases



Les gestionnaires d'événements

- Lorsqu'un événement se produit, il est possible de lui associer une action, c'est à dire d'invoquer une fonction liée à cet événement.
- Pour cela, tout objet javascript faisant référence à un élément du document HTML possède une propriété de type `function` nommée `addEventListener`
- Cette fonction possède 3 paramètres :
  - Le nom de l'évènement.
  - Le nom d'une fonction appelée lorsque l'événement est déclenché.
  - Un booléen valant `true` pour la phase de capture ou `false` pour la phase de bouillonnement.

Premier exemple

Dans l'exemple suivant, la fonction `run` est invoquée dès que l'événement `load` se produit dans le document :

```
1  window.addEventListener("load",run,false);
2
3  function run(){
4   /* Placer ici le code javascript
5   à exécuter une fois le document
6   chargé dans le navigateur */
7  }
```

Disponibilité du DOM au chargement du document (1)

Tant que le navigateur n'a pas fini de charger le document, le DOM ne peut pas être manipulé.

Considérons le script nommé `monScript.js` contenant les 3 lignes de code suivantes :

```
1 const elt = document.body;
2 console.log(elt);
3 elt.style.backgroundColor = "red";
```

Disponibilité du DOM au chargement du document (2)

Le code suivant va produire une erreur. Le DOM n'est pas encore chargé au moment de l'exécution du script.

```
1  <!DOCTYPE html>
2  <html lang="fr">
3     <head>
4        <meta charset="UTF-8" />
5        <title>Accès au DOM</title>
6        <script src="monScript.js"></script>
7     </head>
8     <body></body>
9  </html>
```

Disponibilité du DOM au chargement du document (3)

Une première solution consiste à placer le (ou les) scripts à la fin du fichier HTML. Le navigateur aura fini le chargement du DOM au moment de l'exécution.

```
1  <!DOCTYPE html>
2  <html lang="fr">
3      <head>
4          <meta charset="UTF-8" />
5          <title>Accès au DOM</title>
6      </head>
7      <body></body>
8      <script src="monScript.js"></script>
9  </html>
```

Disponibilité du DOM au chargement du document (4)

Une deuxième solution consiste à placer tout le code dans une fonction et exécuter celle-ci dès que l'événement `load` se produit sur la fenêtre du navigateur en utilisant la syntaxe suivante :

```
1 window.addEventListener("load",run,false);
2
3 function run(){
4     const elt = document.body;
5     console.log(elt);
6     elt.style.backgroundColor = "red";
7 }
```

Disponibilité du DOM au chargement du document (5)

Une troisième solution plus récente consiste à utiliser l'attribut `defer` sur la balise `script`.

```
1 <!DOCTYPE html>
2 <html lang="fr">
3    <head>
4       <meta charset="UTF-8" />
5       <title>Accès au DOM</title>
6       <script src="monScript.js" defer="defer"></
            script>
7    </head>
8    <body></body>
9 </html>
```

Les événements souris

- click
- dblclick
- mousedown
- mouseover
- mouseenter
- mouseleave
- mousemove
- mouseout

Exemple :

```
1 element.addEventListener("click",change,false);
2 function change(evt){
3     evt.target.style.color = "red";
4 }
```

Les événements souris (1)

L'objet `Event` associé à un événement déclenché par l'action de l'utilisateur sur la souris (`MouseEvent`) possède, entre autres, les propriétés suivantes :

- `clientX` : Coordonnée horizontale de la souris par rapport à la zone de la page visible à l'écran.
- `clientY` : Coordonnée verticale de la souris par rapport à la zone de la page visible à l'écran.
- `pageX` : Coordonnée horizontale de la souris par rapport à l'intégralité de la page.
- `pageY` : Coordonnée verticale de la souris par rapport à l'intégralité de la page.

Les événements souris (2)

- `screenX` : Coordonnée horizontale de la souris par rapport à l'écran.
- `screenY` : Coordonnée verticale de la souris par rapport à l'écran.
- `offsetX` : Coordonnée horizontale de la souris par rapport à l'élément cible.
- `offsetY` : Coordonnée verticale de la souris par rapport à l'élément cible.

Les événements souris (3)

- `button` : contient un entier indiquant quel bouton de la souris est cliqué. La valeur est :
  - 0 pour le bouton principal, en général celui de gauche.
  - 1 pour le bouton auxiliaire, en général celui du milieu.
  - 2 pour le bouton secondaire, en général celui de droite.

Cette liste concernant les propriétés de `MouseEvent` est non exhaustive.

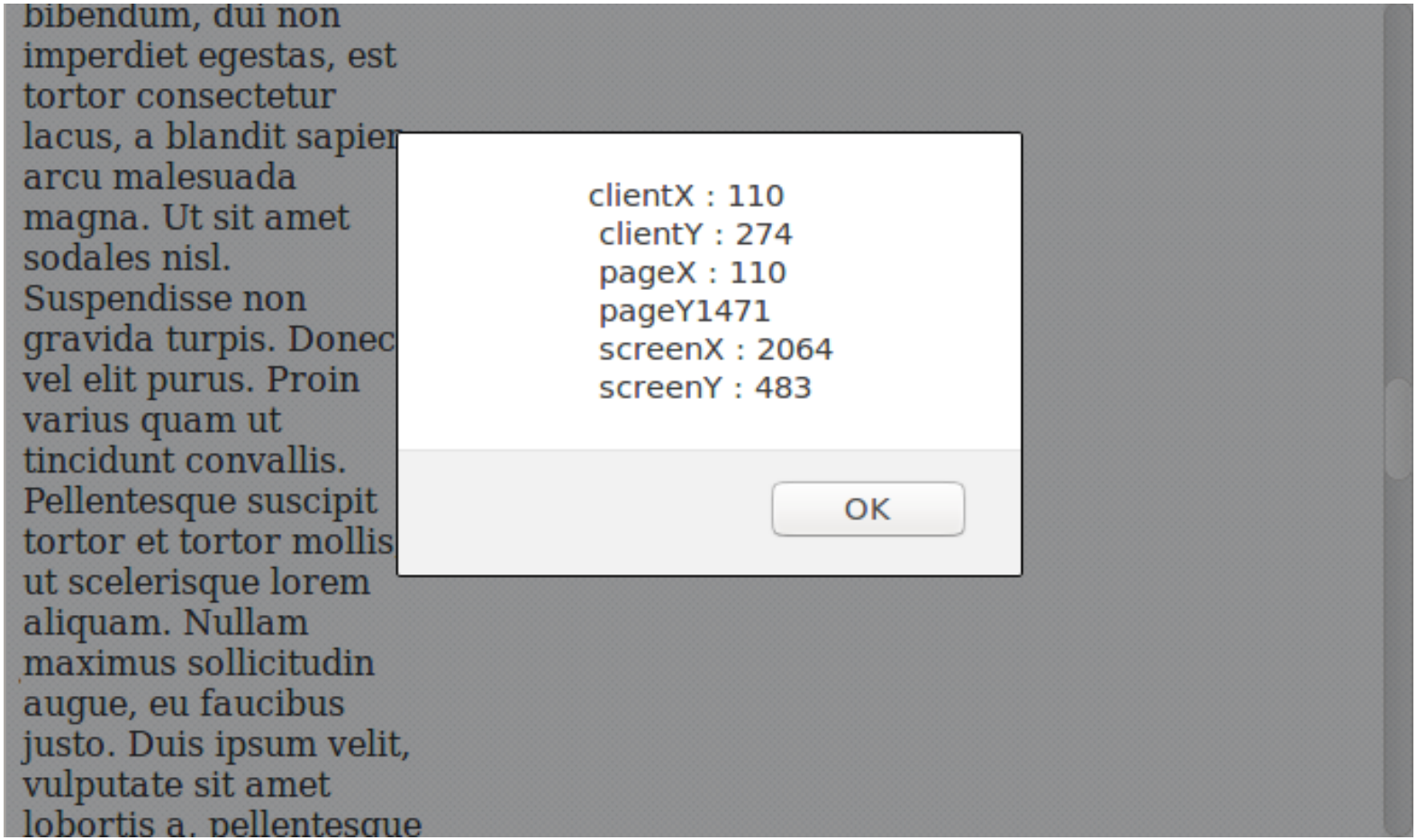
Exemple 1 : le HTML

```
1  <!DOCTYPE html>
2  <html>
3     <head>
4        <title>Les propriétés de MouseEvent</title>
5        <meta charset="utf-8" />
6        <script src="script.js"></script>
7        <style>
8           #contenu {width : 30%;}
9        </style>
10    </head>
11    <body>
12       <p id="contenu">
13          Un  contenu très long...
14       </p>
15    </body>
16 </html>
```

Exemple 1 : le javascript

```
1  window.addEventListener("load",run,false);
2  function run(){
3     document.getElementById("contenu").
          addEventListener("click",posSouris,false);
4     function posSouris(evt){
5        alert("clientX : "+evt.clientX+
6              "\n clientY : "+evt.clientY+
7              "\n pageX : "+evt.pageX+
8              "\n pageY : "+evt.pageY+
9              "\n screenX : "+evt.screenX+
10             "\n screenY : "+evt.screenY);
11    }
12 }
```

Exemple 1 : le résultat



Exemple 2 : le HTML

```
1  <!DOCTYPE html>
2  <html>
3     <head>
4        <title>Les propriétés de MouseEvent</title>
5        <meta charset="utf-8" />
6        <script src="script.js"></script>
7        <link rel="stylesheet" href="style.css" />
8     </head>
9     <body>
10       <div id="rouge"></div>
11       <div id="bleu"></div>
12    </body>
13 </html>
```

Exemple 2 : le CSS

```
1  div {
2      width : 150px;
3      height : 150px;
4      position : absolute;
5  }
6  #rouge {
7      background-color : red;
8      top : 50px;
9      left : 50px;
10 }
11 #bleu {
12     background-color : aqua;
13     top : 50px;
14     left : 300px;
15 }
```

Exemple 2 : Le rendu dans le navigateur

Exemple 2 : le javascript

```
1  window.addEventListener("load",run,false);
2
3  function run(){
4      document.getElementById("rouge").
           addEventListener("mouseover",change,false);
5      document.getElementById("bleu").addEventListener
           ("mouseover",change,false);
6
7      function change(evt){
8          this.style.backgroundColor = "yellow";
9      }
10 }
```

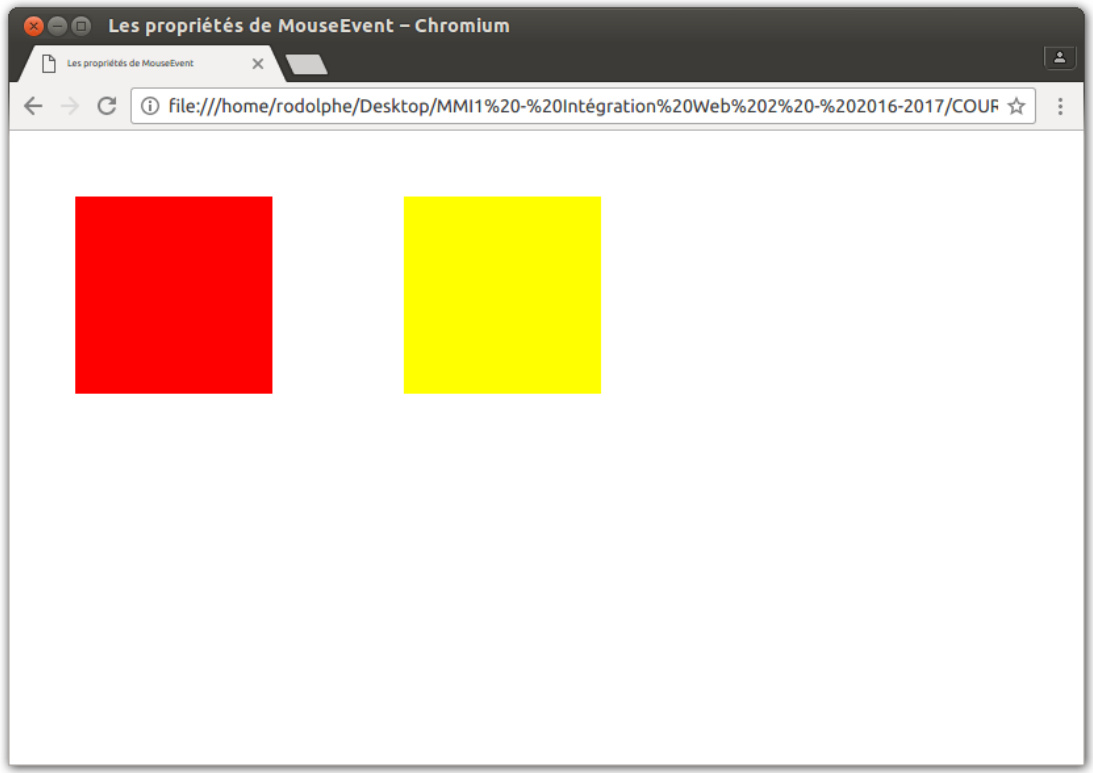
`this` fait référence à l'objet qui provoque l'événement !

Exemple 2 : Le résultat

Si on survole le carré rouge :

Exemple 2 : Le résultat

Si on survole le carré bleu :



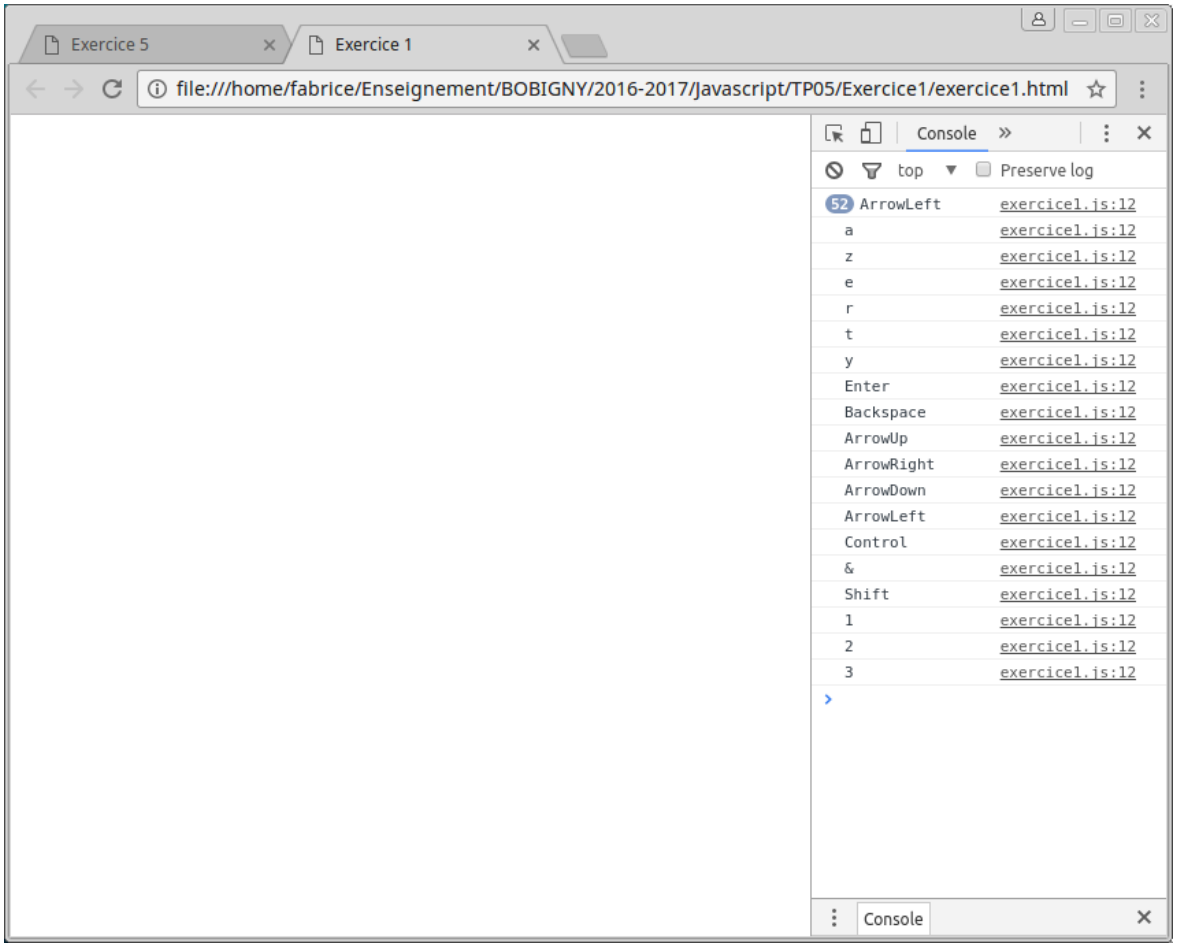
Les événements liés au clavier

- `keydown` : une touche du clavier est enfoncée
- `keypress` : un caractère est tapé au clavier (`Ctrl` n'est pas un caractère)
- `keyup` : une touche du clavier est relâchée

Exemple

```
1  window.addEventListener("load",run,false);
2
3  function run(){
4      document.addEventListener("keydown",affiche,
           false);
5
6      function affiche(evt){
7          console.log(evt.key);
8      }
9  }
```

Résultat


Exercice 5
Exercice 1
file:///home/fabrice/Enseignement/BOBIGNY/2016-2017/Javascript/TP05/Exercice1/exercice1.html
Console
top
Preserve log
52 ArrowLeft exercice1.js:12
a exercice1.js:12
z exercice1.js:12
e exercice1.js:12
r exercice1.js:12
t exercice1.js:12
y exercice1.js:12
Enter exercice1.js:12
Backspace exercice1.js:12
ArrowUp exercice1.js:12
ArrowRight exercice1.js:12
ArrowDown exercice1.js:12
ArrowLeft exercice1.js:12
Control exercice1.js:12
& exercice1.js:12
Shift exercice1.js:12
1 exercice1.js:12
2 exercice1.js:12
3 exercice1.js:12
Console

Les événements liés aux formulaires

- `focus` : un champ est sélectionné, il est prêt à recevoir les caractères saisis au clavier
- `blur` : le champ perd le `focus`
- `change` : la valeur du champ est en cours de modification
- `select` : sélection à la souris du contenu de champ
- `submit` : validation du formulaires
- `reset` : remise à zéro du formulaire

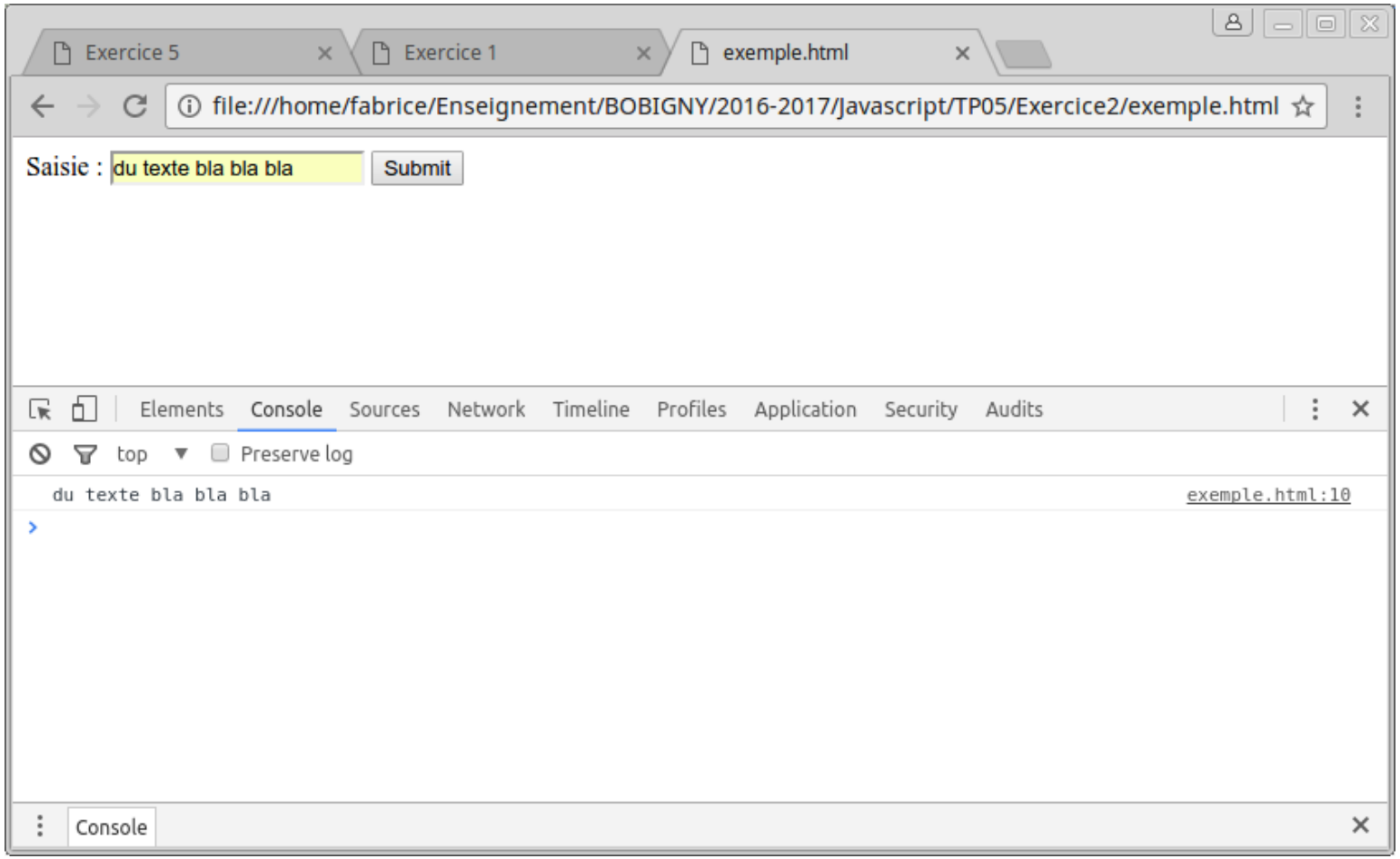
Exemple

```
1 <form action="exemple.html" method="get" id="
      formulaire">
2     <label for="input1">Saisie : </label>
3     <input type="text" id="input1" name="masaisie"
          />
4     <input type="submit" />
5 </form>
```

Exemple

```
1  window.addEventListener('load',run,false);
2
3  function run(){
4     document.getElementById("formulaire").
          addEventListener('submit',affiche,true);
5
6     function affiche(evt){
7        evt.preventDefault();
8        var champ = document.getElementsByName("
             masaisie")[0];
9        console.log(champ.value);
10    }
11 }
```

Résultat


Exercice 5
Exercice 1
exemple.html
file:///home/fabrice/Enseignement/BOBIGNY/2016-2017/Javascript/TP05/Exercice2/exemple.html
Saisie : du texte bla bla bla
Submit
Elements
Console
Sources
Network
Timeline
Profiles
Application
Security
Audits
top
Preserve log
du texte bla bla bla
exemple.html:10
Console

Le défilement (1)

Un événement `scroll` se propage dès qu'une barre de défilement est manipulée. Notons les propriétés de l'objet `window` en lecture seule suivantes :

- `window.scrollX` : Nombre de pixels scrollés dans la fenêtre horizontalement.
- `window.scrollY` : Nombre de pixels scrollés dans la fenêtre en mode verticalement.

Notons que les propriétés `window.pageXOffset` et `window.pageYOffset` sont des alias respectivement de `window.scrollX` et `window.scrollY`

Le défilement (2)

Pour la fenêtre du navigateur, nous avons accès également aux propriétés suivantes (en lecture seule) :

- `window.scrollLeft` : distance entre le bord gauche du document et la partie la plus à gauche de son contenu visible.
- `window.scrollTop` : distance entre le haut du document et la partie la plus haute de son contenu visible.
- `window.scrollWidth` : Largeur totale défilable du document.
- `window.scrollHeight` : Hauteur totale défilable du document.

Le défilement (3)

Pour les éléments d'un document qui possèdent une barre de défilement :

- `Element.scrollLeft` : distance entre le bord gauche de l'élément et la partie la plus à gauche de son contenu visible.
- `Element.scrollTop` : distance entre le haut de l'élément et la partie la plus haute de son contenu visible.

  Ex : `document.documentElement.scrollTop+=100`
- `Element.scrollWidth` : Largeur totale défilable d'un élément de la page. (lecture seule)
- `Element.scrollHeight` : Hauteur totale défilable d'un élément de la page. (lecture seule)

Le défilement (4)

Il existe également des méthodes pour contrôler le défilement d'un document ou d'un élément scrollable :

- La fonction `scroll(x,y)` permet de faire défiler un document ou un élément jusqu'à un couple de coordonnées `(x,y)` précisées en paramètres. L'unité est le pixel.

  Par exemple `scroll(0,0)` permet de défiler jusqu'au début du document.

Le défilement (5)

- Il existe aussi la syntaxe `scroll(options)` où `options` est un objet possédant les propriétés suivantes :
  - `left`
  - `top`
  - `behavior` : Indique si le défilement devrait être instantané ou doux. Cette option est une chaine de caractères qui doit être l'une de ces valeurs :
    - smooth : le défilement doit s'animer en douceur
    - instant : le défilement devrait se produire instantanément, en un seul saut
    - auto : le comportement de défilement est déterminé par une valeur calculée

Le défilement (6)

Exemple :

```
window.scroll(
   top: 100,
   left: 100,
   behavior: "smooth",
);
```

Notons qu'il existe la fonction `scrollTo` qui fonctionne exactement comme la fonction `scroll`. La plupart des navigateurs modernes supportent les 2 syntaxes.

le défilement (7)

La fonction `scrollBy` permet également de faire défiler un document selon les deux axes vertical ou horizontal

- `scrollBy(x,y)` fait défiler un document ou un élément de `x` pixels horizontalement et de `y` pixels verticalement depuis la position actuelle.
- La fonction `scrollBy` permet de réaliser un défilement **relatif** à la position actuelle. Les coordonnées passées en argument à la fonction `scroll` (ou `scrollTo`) sont des coordonnées **absolues**
- Comme pour la fonction `scroll`, il existe également la syntaxe `scrollBy(options)`.

le défilement (8)

La fonction `scrollByLines` permet de réaliser un défilement d'un certain nombre de lignes passé en paramètre.

La fonction `scrollByPages`. permet de réaliser un défilement d'un certain nombre de pages (hauteur d'écran ?) passé en paramètre.

Ces fonctions sont **non standards** et elles ne sont pas implémentées dans tous les navigateurs, notamment chrome.

Il est conseillé de ne pas les utiliser.

La molette de la souris (1)

L'événement `wheel` remplace les événements `mousewheel` et `mousescroll`, devenus obsolètes, non standards et non compatibles avec certains navigateurs.

Il permet de déclencher une action lorsque la molette de la souris est utilisée.

Attention, un événement `wheel` n'est pas forcément lié à un défilement. Par exemple, dans la plupart des navigateurs, la molette utilisée simultanément avec la touche `CTRL` permet de contrôler le zoom sur une page.

La molette de la souris (2)

Un objet de type `WheelEvent` possède les propriétés suivantes (en lecture seule) :

- `deltaX` : Un entier indiquant la valeur algébrique du défilement horizontal
- `deltaY` : Un entier indiquant la valeur algébrique du défilement vertical
- `deltaZ` : Un entier indiquant la valeur algébrique du défilement selon un troisième axe z.
- `deltaMode` : Un entier indiquant l'unité utilisée par les 3 propriétés précédentes : en pixels (valeur 0), en lignes (valeur 1) ou en pages (valeur 2).

...les autres événements

- `abort` : Le chargement d'une ressource (p. ex. une image) échoue.
- `error` : Une erreur survient pendant le chargement d'un fichier externe.
- `load` : Le chargement d'une ressource (p. ex. une image) est terminé.
- `resize` : Les dimensions de la partie visible du document changent.
- `unload` : Se déclenche à la fermeture du document.