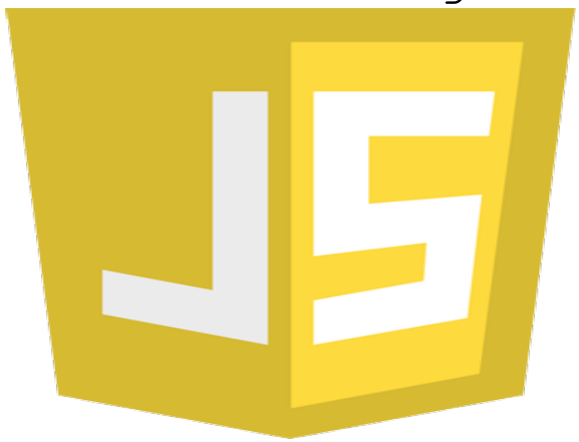
JavaScript : manipulation du **DOM**

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com



Plan

Plan

Plan

JavaScript

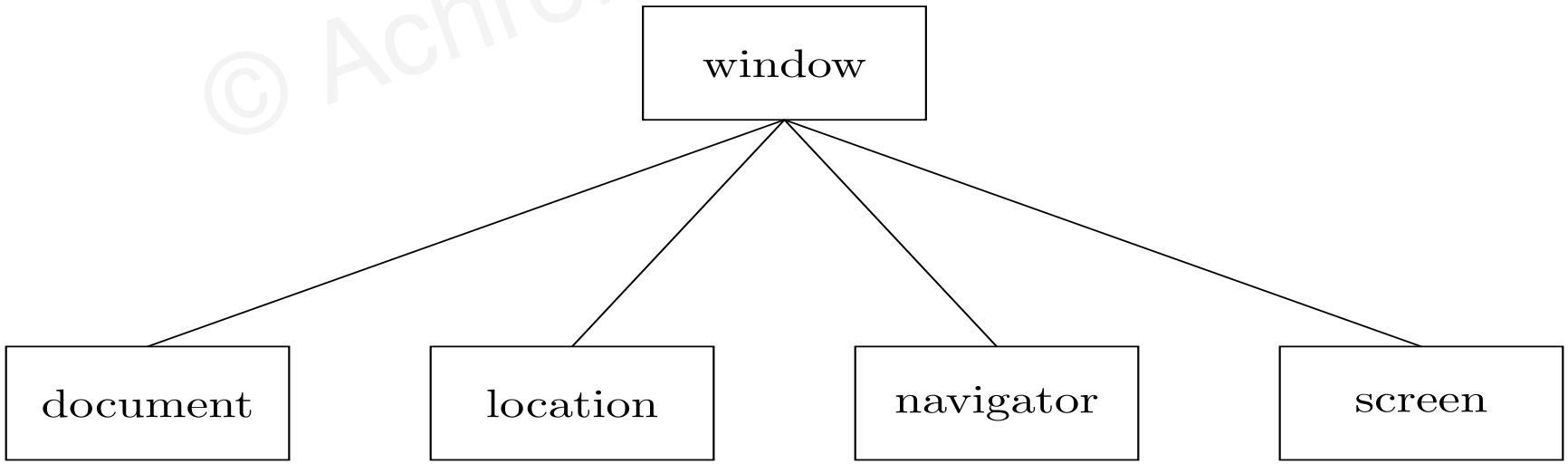
JavaScript nous permet, via des objets prédéfinis, d'avoir des informations sur

- le visiteur
- son navigateur
- la structure de la page (document) qu'il consulte
- son écran
- ...

JavaScript

JavaScript nous permet, via des objets prédéfinis, d'avoir des informations sur

- le visiteur
- son navigateur
- la structure de la page (document) qu'il consulte
- son écran
- ...



JavaScript

L'objet `Window` contient

- quatre autres objets importants : `document`, `location`, `navigator`, `screen`
- des méthodes basiques comme `alert`, `confirm`
- les objets comme `String`, `Date`, `Math`...
- toute variable, objet... définis par le développeur
- ...

L'objet `document` contient

- `contentType` : text/html, text/xml...
- toutes les balises **HTML** constituant le document (**DOM**)
- les attributs et leurs valeurs
- ...

L'objet `document` contient

- `contentType` : text/html, text/xml...
- toutes les balises **HTML** constituant le document (**DOM**)
- les attributs et leurs valeurs
- ...

L'objet `location` contient

- `hostname` : localhost...
- `port` : 3000
- `protocol` : http
- ...

L'objet `navigator` contient

- `language` : fr_FR...
- `connection` : 4G
- `cookieEnabled` : true
- ...

L'objet `navigator` contient

- `language` : fr_FR...
- `connection` : 4G
- `cookieEnabled` : true
- ...

L'objet `screen` contient

- `height`
- `width`
- `orientation`
- ...

JavaScript

DOM : **D**ocument **O**bject **M**odel

- Une représentation en mémoire de la structure d'un document **HTML** ou **XML**.
- Une hiérarchie en forme d'arbre, où chaque nœud représente une partie du document.
- Permet de visualiser le document sous une forme structurée qui peut être manipulée par des scripts.
- Avant la standardisation par le **W3C**, chaque navigateur implémentait son propre modèle de **DOM**, ce qui entraînait des incohérences. .

JavaScript

DOM vs API DOM

- **DOM** : le modèle de données en forme d'arbre qui représente la structure du document.
- **API DOM** : l'interface de programmation qui permet aux développeurs de manipuler le **DOM**
  - Sélectionner des éléments du **DOM** (`document.getElementById()`, `document.querySelector()`...).
  - Modifier les propriétés des éléments (`element.textContent`, `element.style`).
  - Ajouter, supprimer ou remplacer des nœuds (`element.appendChild()`, `element.removeChild()`).
  - Gérer les événements associés aux éléments (`element.addEventListener()`...).

JavaScript

Quelques classes de l'**API DOM**

Les classes suivantes font partie de l'**API DOM** et permettent de manipuler différents types de nœuds dans un document

- `Node` : classe de base de presque tous les objets du **DOM**. Tous les types de nœuds (éléments, attributs, textes, commentaires, etc.) héritent de `Node`.
- `Element` : sous-classe de `Node` qui représente un élément **HTML** ou **XML** dans le **DOM**.
- `HTMLElement` : sous-classe de `Element` qui représente spécifiquement les éléments **HTML**.
- `Text` : nœud représentant du texte brut dans le **DOM**.
- `Comment` : nœuud de commentaire dans le **DOM**, qui correspond à un commentaire **HTML** ou **XML**.
- `Event` : classe de base pour tous les événements dans le **DOM**.
- ...

JavaScript

Quelques sous-classes de `HTMLElement` et leur balise correspondante

- `HTMLDivElement` : `<div>`
- `HTMLParagraphElement` : `<p>`
- `HTMLAnchorElement` : `<a>`
- `HTMLImageElement` : `<img>`
- `HTMLTableElement` : `<table>`
- `HTMLTableRowElement` : `<tr>`
- `HTMLTableCellElement` : `<td>` ou `<th>`
- `HTMLHeadingElement` : `<h1>`, `<h2>`, `<h3>`...
- ...

JavaScript

Quelques sous-classes de `HTMLElement` avec les balises de formulaire correspondantes

- `HTMLInputElement` : `<input>`
- `HTMLTextAreaElement` : `<textarea>`
- `HTMLSelectElement` : `<select>`
- `HTMLOptionElement` : `<option>`
- `HTMLButtonElement` : `<button>`
- `HTMLLabelElement` : `<label>`
- `HTMLFieldSetElement` : `<field>`
- ...

JavaScript

Plusieurs méthodes pour récupérer un élément **HTML**

- selon l'identifiant : `getElementById()`
- selon le nom de la classe : `getElementsByClassName()`
- selon le nom de la balise : `getElementsByTagName()`
- selon l'attribut `name` : `getElementsByName()`
- selon un sélecteur **CSS** : `querySelectorAll()` ou `querySelector()`

JavaScript

Considérons la page HTML suivante où nous plaçons `<script>` à la fin du `<body>` pour attendre la fin du chargement du `DOM`

```
<!DOCTYPE html>
<html lang="fr">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>My JS Page</title>
    <link rel="stylesheet" href="style.css">
</head>

<body>
    <div id=container>
        Considérons les paragraphes suivants :
        <p class=blue name=parConteneur> bonjour </p>
        <p class=red name=parConteneur> bonsoir, voici
            <a href="http://www.lsis.org/elmouelhia/" target="_blank">
                ma page
            </a>
        </p>
        <p class=blue name=parConteneur> salut </p>
    </div>
    <script src="script.js"></script>
</body>

</html>
```

JavaScript

Pour ordonner la lecture de `script.js` après le chargement du `DOM`, on peut utiliser l'attribut `defer` et garder `<script>` dans `<head>`

```
<!DOCTYPE html>
<html lang="fr">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>My JS Page</title>
    <link rel="stylesheet" href="style.css">
     <script src="script.js" defer></script>
</head>

<body>
    <div id=container>
        Considérons les paragraphes suivants :
        <p class=blue name=parConteneur> bonjour </p>
        <p class=red name=parConteneur> bonsoir, voici
            <a href="http://www.lsis.org/elmouelhia/"> ma page </a>
        </p>
        <p class=blue name=parConteneur> salut </p>
    </div>
</body>

</html>
```

JavaScript

Contenu de `style.css`

```
.red {
  color: red;
}

.blue {
  color: blue;
}

.green {
  color: green;
}
```

JavaScript

Récupérer un élément selon son `id`, on ajoute dans `script.js`

```
var container = window.document.getElementById("container");
console.log(container.innerHTML);
```

JavaScript

Récupérer un élément selon son `id`, on ajoute dans `script.js`

```
var container = window.document.getElementById("container");
console.log(container.innerHTML);
```

Ou sans passer par l'objet `window`

```
var container = document.getElementById("container");
console.log(container.innerHTML);
```

JavaScript

Récupérer un élément selon son `id`, on ajoute dans `script.js`

```
var container = window.document.getElementById("container");
console.log(container.innerHTML);
```

Ou sans passer par l'objet `window`

```
var container = document.getElementById("container");
console.log(container.innerHTML);
```


Remarque

Il ne faut pas utiliser `#` dans `getElementById()`.

JavaScript

Récupérer des éléments selon le nom de la balise

```
var paragraphes = document.getElementsByTagName("p");

for (let paragraphe of paragraphes) {
    console.log(paragraphe.innerHTML);
}
```

JavaScript

Récupérer des éléments selon le nom de la balise

```
var paragraphes = document.getElementsByTagName("p");

for (let paragraphe of paragraphes) {
    console.log(paragraphe.innerHTML);
}
```

Ou encore

```
var paragraphes = container.getElementsByTagName("p");

for (let paragraphe of paragraphes) {
    console.log(paragraphe.innerHTML);
}
```

Récupérer un élément selon sa classe

```
var bleus = document.getElementsByClassName("blue");

for (let bleu of bleus) {
    console.log(bleu.innerHTML);
}
```

Récupérer un élément selon sa classe

```
var bleus = document.getElementsByClassName("blue");

for (let bleu of bleus) {
    console.log(bleu.innerHTML);
}
```

Ou aussi

```
var bleus = container.getElementsByClassName("blue");

for (let bleu of bleus) {
    console.log(bleu.innerHTML);
}
```

Récupérer un élément selon sa classe

```
var bleus = document.getElementsByClassName("blue");

for (let bleu of bleus) {
    console.log(bleu.innerHTML);
}
```

Ou aussi

```
var bleus = container.getElementsByClassName("blue");

for (let bleu of bleus) {
    console.log(bleu.innerHTML);
}
```


Remarque

Il ne faut pas utiliser `.` dans `getElementsByClassName()`.

JavaScript

Récupérer un élément selon la valeur de son attribut `name`

```
var parConteneurs = document.getElementsByName("parConteneur");
console.log(parConteneurs);
```

JavaScript

Récupérer un élément selon la valeur de son attribut `name`

```
var parConteneurs = document.getElementsByName("parConteneur");
console.log(parConteneurs);
```

Ou aussi

```
for (let parConteneur of parConteneurs) {
    console.log(parConteneur.innerHTML);
}
```

JavaScript

Récupérer un élément selon un sélecteur CSS 3

```
var pbleus = document.querySelectorAll("p.blue");
for (let bleu of pbleus) {
    console.log(bleu.innerHTML);
}
```

JavaScript

Récupérer le premier élément selon un sélecteur CSS 3

```
var pbleu = document.querySelector("p.blue");
console.log(pbleu.innerHTML);
```

JavaScript

Récupérer le premier élément selon un sélecteur CSS 3

```
var pbleu = document.querySelector("p.blue");
console.log(pbleu.innerHTML);
```

Ceci déclenche une erreur

```
var prouges = document.querySelector("p.red");
for (let rouge of prouges) {
    console.log(rouge.innerHTML);
}
```

JavaScript

Remarque

- Les méthodes `getElementsByX()` retournent un `HTMLCollection` (itérable)
- La méthode `querySelectorAll` retourne un `NodeList` (itérable)

JavaScript

Remarque

- Les méthodes `getElementsByX()` retournent un `HTMLCollection` (itérable)
- La méthode `querySelectorAll` retourne un `NodeList` (itérable)

`HTMLCollection` VS `NodeList`

- `HTMLCollection` : dynamique et se met à jour en temps réel (par exemple, si des éléments correspondant à la sélection sont ajoutés ou supprimés).
- `NodeList` : statique et reste inchangée après l'appel initial.

JavaScript

HTMLCollection et NodeList : méthodes communes et méthodes différentes

- Méthodes et opérateurs communs :
  - `item(index)` : permet de récupérer un élément à un index spécifique dans la collection ou la liste.
  - Accès par indice ([index]).
- Méthode spécifique à `HTMLCollection` (non présentes dans `NodeList`) :
  - `namedItem(name)` : permet de récupérer un élément dans la collection à partir de son attribut `id` ou `name`.
- Méthode spécifique à `NodeList` (non présentes dans `HTMLCollection`) :
  - `forEach(callback)` : permet d'itérer facilement sur chaque élément de la liste de nœuds, similaire à la méthode `forEach` des tableaux.

JavaScript

Récupérer l'attribut d'un élément HTML

```
var lien = document.querySelector('a');

var href = lien.getAttribute('href');
console.log(href);
```

JavaScript

Récupérer l'attribut d'un élément HTML

```
var lien = document.querySelector('a');
lien.setAttribute('href', 'https://www.w3schools.com');

console.log(lien);
// affiche https://www.w3schools.com
```

JavaScript

Supprimer l'attribut d'un élément HTML

```
var lien = document.querySelector('a');
lien.removeAttribute('href');
```

JavaScript

Récupérer tous les attributs d'un élément HTML

```
var lien = document.querySelector('a');
var attributes = lien.attributes;

for (let i = 0; i < attributes.length; i++) {
    console.log(attributes[i].name + " = " + attributes[i].value);
}

for (const attr of attributes) {
    console.log(attr.name, attr.value);
}
```

JavaScript

Remarque

`attributes` retourne un objet itérable de type `NamedNodeMap`.

JavaScript

Pour certains attributs comme `href`, on peut accéder directement à la valeur via son nom

```
var lien = document.querySelector('a');
var href = lien.href;
console.log(href);
```

JavaScript

Pour certains attributs comme `href`, on peut accéder directement à la valeur via son nom

```
var lien = document.querySelector('a');
var href = lien.href;
console.log(href);
```

Modifier l'attribut d'un élément HTML

```
lien.href = 'https://www.w3schools.com';
console.log(lien);
```

JavaScript

Remarque

Les attributs `class` et `for` sont des mots-clés réservés en **JavaScript**, on ne peut donc les utiliser pour modifier leurs valeurs **HTML**.

JavaScript

Remarque

Les attributs `class` et `for` sont des mots-clés réservés en **JavaScript**, on ne peut donc les utiliser pour modifier leurs valeurs **HTML**.

Solution

- Utiliser `className` ou `classList` pour l'attribut `class`
- Et `forHtml` pour l'attribut `for`

JavaScript

className

Une chaîne de caractère contenant les classes d'un `HTMLElement` séparées par un espace.

JavaScript

Récupérer la classe d'un élément HTML

```
var container = document.getElementById("container");
console.log(container.className);
```

JavaScript

Récupérer la classe d'un élément HTML

```
var container = document.getElementById("container");
console.log(container.className);
```

Modifier une classe d'un élément HTML

```
container.className = "blue";
console.log(container.className);
```

JavaScript

Récupérer la classe d'un élément HTML

```
var container = document.getElementById("container");
console.log(container.className);
```

Modifier une classe d'un élément HTML

```
container.className = "blue";
console.log(container.className);
```


Ajouter une nouvelle classe à un élément HTML

```
container.className += " red";
console.log(container.className);
```

JavaScript

classList

- retourne un objet itérable de type `DOMTokenList`.
- représente les classes **CSS** d'un `HTMLElement` sous forme de liste de tokens (chaînes de caractères) : chaque token est une classe distincte.

JavaScript

Récupérer la liste des classes d'un élément HTML

```
var container = document.getElementById("container");
var list = container.classList;
console.log(list);
```

JavaScript

Récupérer la liste des classes d'un élément HTML

```
var container = document.getElementById("container");
var list = container.classList;
console.log(list);
```

Ajouter une classe à un élément HTML

```
var container = document.getElementById("container");
var list = container.classList;
list.add('green');
console.log(list);
```

JavaScript

Récupérer la liste des classes d'un élément HTML

```
var container = document.getElementById("container");
var list = container.classList;
console.log(list);
```

Ajouter une classe à un élément HTML

```
var container = document.getElementById("container");
var list = container.classList;
list.add('green');
console.log(list);
```

Ajouter des classes à un élément HTML

```
var container = document.getElementById("container");
var list = container.classList;
list.add('red', "blue");
console.log(list);
```

JavaScript

Supprimer une classe d'un élément HTML

```
list.remove('red');
console.log(list);
```

JavaScript

Supprimer une classe d'un élément HTML

```
list.remove('red');
console.log(list);
```

Supprimer plusieurs classes d'un élément HTML

```
list.remove('red', 'green');
console.log(list);
```

JavaScript

Supprimer une classe si elle existe, sinon l'ajouter

```
list.toggle('red');
console.log(list);
```

JavaScript

Supprimer une classe si elle existe, sinon l'ajouter

```
list.toggle('red');
console.log(list);
```

Remplacer une classe

```
list.replace('red', 'blue');
console.log(list);
```

JavaScript

Autres méthodes de `classList`

- `length` : retourne le nombre de classes.
- `contains(classe)` : retourne `true` si `classe` appartient à `classList`, `false` sinon.
- `item(i)` : retourne la classe d'indice `i` si `i` est inférieur à la longueur de la liste, `null` sinon.

JavaScript

Les attributs d'un formulaire

- `value` : pour récupérer la valeur saisie dans une zone de saisie (input, textarea...)
- `checked` : pour déterminer si une case à cocher ou un bouton radio a été coché ou non (`true` ou `false`)
- `options` : pour récupérer les `option` d'un `select` sous forme d'une liste
- `selected` : pour tester si l'`option` d'un `select` a été sélectionnée ou non (`true` ou `false`)
- `selectedIndex` : pour récupérer l'indice de l'`option` sélectionnée
- `selectedOptions` : pour récupérer une collection des options sélectionnées (si la propriété `multiple` est présente dans la balise `select`)
- ...

JavaScript

Pour récupérer l'indice de l'élément sélectionné d'une liste déroulante (`<select>`)

```
document.getElementById("select").selectedIndex;
```

JavaScript

Pour récupérer l'indice de l'élément sélectionné d'une liste déroulante (<select>)

```
document.getElementById("select").selectedIndex;
```

Pour supprimer l'élément sélectionné d'une liste déroulante (<select>)

```
var liste = document.getElementById("#select");
liste.remove(liste.selectedIndex);
```

JavaScript

Pour récupérer l'indice de l'élément sélectionné d'une liste déroulante (<select>)

```
document.getElementById("select").selectedIndex;
```

Pour supprimer l'élément sélectionné d'une liste déroulante (<select>)

```
var liste = document.getElementById("#select");
liste.remove(liste.selectedIndex);
```

Pour vider tous les champs d'un formulaire

```
document.getElementById("form").reset();
```

JavaScript

Récupérer le contenu d'un élément HTML (y compris les balises)

```
var container = document.getElementById("container");
console.log(container.innerHTML);
```

JavaScript

Récupérer le contenu d'un élément HTML (y compris les balises)

```
var container = document.getElementById("container");
console.log(container.innerHTML);
```

Récupérer le contenu d'un élément HTML (sans les balises)

```
var container = document.getElementById("container");
console.log(container.textContent);
```

JavaScript

Modifier le contenu d'un élément HTML avec `innerHTML`

```
var container = document.getElementById("container");
container.innerHTML += "<p> hello </p>";
console.log(container.innerHTML);
```

JavaScript

Modifier le contenu d'un élément HTML avec `innerHTML`

```
var container = document.getElementById("container");
container.innerHTML += "<p> hello </p>";
console.log(container.innerHTML);
```

Récupérer le contenu d'un élément HTML avec `textContent`

```
var container = document.getElementById("container");
container.textContent += "<p> hello </p>";
console.log(container.textContent);
console.log(container.innerHTML);
```

JavaScript

Récapitulatif

- `innerHTML` : manipule et récupère du contenu **HTML** (balises et texte).
- `innerText` : récupère uniquement le texte visible (pas le texte caché) en tenant compte des styles **CSS**.
- `textContent` : récupère tout le texte, sans balises et sans tenir compte des styles **CSS** (affiche aussi le texte caché).

JavaScript

Modifier la couleur

```
var container = document.getElementById("container");
container.style.color = "red";
```

JavaScript

Modifier la couleur

```
var container = document.getElementById("container");
container.style.color = "red";
```

Pour les propriétés CSS composées de deux mots séparés par -, il faut supprimer - mettre la propriété en camelCase

```
var container = document.getElementById("container");
container.style.backgroundColor = "red";
```

JavaScript

Modifier la couleur

```
var container = document.getElementById("container");
container.style.color = "red";
```

Pour les propriétés CSS composées de deux mots séparés par -, il faut supprimer - mettre la propriété en camelCase

```
var container = document.getElementById("container");
container.style.backgroundColor = "red";
```


Remarque

Un élément **HTML**, ayant classe `red` qui modifie la couleur de fond en rouge, n'a pas forcément la valeur `red` attribuée au `style.backgroundColor`

JavaScript

Récupérer le parent d'un élément HTML (l'objet)

```
var container = document.getElementById("container");
var parent = container.parentNode;
console.log(parent);
```

JavaScript

Récupérer le parent d'un élément HTML (l'objet)

```
var container = document.getElementById("container");
var parent = container.parentNode;
console.log(parent);
```

Récupérer le parent d'un élément HTML (le nom)

```
var container = document.getElementById("container");
var parent = container.parentNode;
console.log(parent.nodeName);
```

JavaScript

Récupérer le premier élément fils d'un élément HTML

```
var container = document.getElementById("container");
var premierFils = container.firstElementChild;
console.log(premierFils.nodeName);
```

JavaScript

Récupérer le premier élément fils d'un élément HTML

```
var container = document.getElementById("container");
var premierFils = container.firstElementChild;
console.log(premierFils.nodeName);
```

Récupérer le dernier élément fils d'un élément HTML

```
var container = document.getElementById("container");
var dernierFils = container.lastElementChild;
console.log(dernierFils.nodeName);
```

JavaScript

Récupérer le premier fils (élément, texte ou autre) d'un élément HTML

```
var container = document.getElementById("container");
var premierFils = container.firstChild;
console.log(premierFils.nodeName);
```

JavaScript

Récupérer le premier fils (élément, texte ou autre) d'un élément HTML

```
var container = document.getElementById("container");
var premierFils = container.firstChild;
console.log(premierFils.nodeName);
```

Récupérer le dernier fils (élément, texte ou autre) d'un élément HTML

```
var container = document.getElementById("container");
var dernierFils = container.lastChild;
console.log(dernierFils.nodeName);
```

JavaScript

Récupérer tous les enfants (élément, texte ou autre) d'un élément HTML

```
var enfants = container.childNodes;
for(let enfant of enfants)
    console.log(enfant);
```

JavaScript

Récupérer tous les enfants (élément, texte ou autre) d'un élément HTML

```
var enfants = container.childNodes;
for(let enfant of enfants)
    console.log(enfant);
```

Récupérer tous les éléments enfants d'un élément HTML

```
var enfants = container.children;
for(let enfant of enfants)
    console.log(enfant);
```

JavaScript

Enfants suivant et précédent

- `previousSibling`
- `nextSibling`
- `previousElementSibling`
- `nextElementSibling`

JavaScript

Enfants suivant et précédent

- `previousSibling`
- `nextSibling`
- `previousElementSibling`
- `nextElementSibling`

À chaque appel d'une de méthodes précédentes, le pointeur passe à l'élément suivant (ou précédent).

JavaScript

Étapes

- Créer l'élément
- Préparer ses attributs [et valeurs]
- L'ajouter au document

Créer un élément de type `p`

```
var par = document.createElement("p");
```

Créer un élément de type `p`

```
var par = document.createElement("p");
```

Préparer ses attributs [et valeurs]

```
par.id = "intro";
par.setAttribute("class", "blue");
var text = document.createTextNode("JS paragraph");
par.appendChild(text);
```

Créer un élément de type `p`

```
var par = document.createElement("p");
```

Préparer ses attributs [et valeurs]

```
par.id = "intro";
par.setAttribute("class", "blue");
var text = document.createTextNode("JS paragraph");
par.appendChild(text);
```


Ajouter l'élément au DOM

```
var container = document.getElementById("container");
container.appendChild(par);
```

Créer un élément de type `p`

```
var par = document.createElement("p");
```

Préparer ses attributs [et valeurs]

```
par.id = "intro";
par.setAttribute("class", "blue");
var text = document.createTextNode("JS paragraph");
par.appendChild(text);
```

Ajouter l'élément au DOM

```
var container = document.getElementById("container");
container.appendChild(par);
```

`appendChild()` ajoute l'élément comme dernier fils de conteneur

JavaScript

`appendChild()` vs `append()`

- `append()` permet d'ajouter plusieurs éléments **HTML** ou chaînes de caractères en une seule fois.
- `appendChild()` n'ajoute qu'un seul élément et n'accepte que des éléments **HTML**.

JavaScript

On peut aussi utiliser

- `.insertBefore(newnode, existingnode)` : pour insérer `newnode` avant `existingnode`

JavaScript

La méthode `insertAdjacentHTML(position, text)`

permet d'insérer un texte HTML (`text`), dans une position spécifiée (`position`)

- `afterbegin`,
- `afterend`,
- `beforebegin`,
- `beforeend`.

JavaScript

Exemple : HTML

```
<!DOCTYPE html>
<html>
  <head>
    <title> float et clear </title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <p id=first>
      bonjour
    </p>
    <script  src="script.js"></script>
  </body>
</html>
```

Le fichier `style.css`

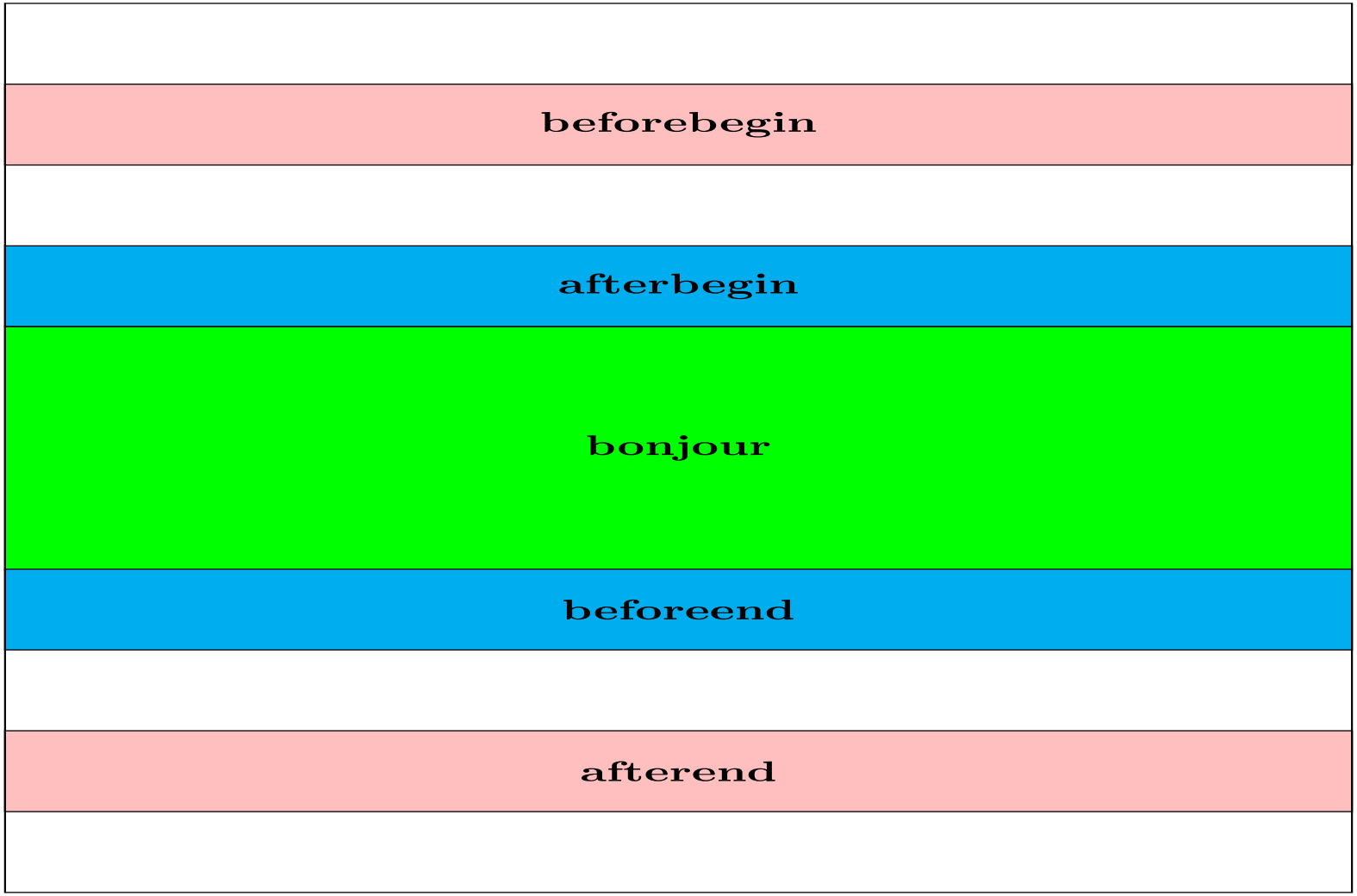
```
.skyblue {
  background-color:
     skyblue;
}
.pink {
  background-color: pink;
}
.green {
  background-color: green;
}
```

Le script `script.js`

```
var p = document.getElementById('first');
p.insertAdjacentHTML("afterbegin","<p class=skyblue> afterbegin </p>");
p.insertAdjacentHTML("afterend","<p class=pink> afterend </p>");
p.insertAdjacentHTML("beforebegin","<p class=pink> beforebegin </p>");
p.insertAdjacentHTML("beforeend","<p class=skyblue> beforeend </p>");
```

JavaScript

Le résultat



JavaScript

Quelques opérations

- `parentNode.replaceChild(newNode, oldNode)` : pour remplacer un nœud fils par un autre.
- `element.replaceWith(newNode)` : pour remplacer un nœud par un autre.
- `node.cloneNode()` : pour cloner un nœud. La méthode accepte un argument booléen pour indiquer si la copie doit être profonde (`true` pour copier tous les nœud enfants) ou superficielle (`false` pour ne copier que le nœud lui-même).
- `parentNode.removeChild(childNode)` : pour supprimer un nœud fils.
- `node.remove()` : pour supprimer le nœud appelant sans avoir besoin de faire référence à son parent.
- `parentNode.hasChildNodes()` : pour vérifier l'existence d'au moins un nœud enfant.

JavaScript

Pour récupérer un élément selon son `id`, une solution possible consite à utiliser la méthode `getElementById()`

```
var container = document.getElementById("container");
console.log(container.innerHTML);
```

JavaScript

Pour récupérer un élément selon son `id`, une solution possible consite à utiliser la méthode `getElementById()`

```
var container = document.getElementById("container");
console.log(container.innerHTML);
```

Tous les éléments HTML ayant un identifiant seront chargés dans l'espace global, donc on peut directement faire (sans avoir besoin d'utiliser `getElementById()`)

```
console.log(window.container.innerHTML);;
```

JavaScript

Pour récupérer un élément selon son `id`, une solution possible consite à utiliser la méthode `getElementById()`

```
var container = document.getElementById("container");
console.log(container.innerHTML);
```

Tous les éléments HTML ayant un identifiant seront chargés dans l'espace global, donc on peut directement faire (sans avoir besoin d'utiliser `getElementById()`)

```
console.log(window.container.innerHTML);;
```

Ou en plus simple

```
console.log(container.innerHTML);;
```

JavaScript

Remarque

Pour éviter les conflits avec des éventuelles variables ou fonctions portant le même nom, il est conseillé d'utiliser `getElementById()`.

JavaScript

Évènements de souris

- `click` : Utilisateur clique sur un élément.
- `dblclick` : Utilisateur double-clique sur un élément.
- `mousedown` : Bouton de la souris est enfoncé sur un élément.
- `mouseup` : Bouton de la souris est relâché sur un élément.
- `mousemove` : Souris est déplacée sur un élément.
- `mouseover` : Souris entre sur un élément.
- `mouseout` : Souris quitte un élément.
- `mouseenter` : Souris entre sur un élément (ne se propage pas).
- `mouseleave` : Souris quitte un élément (ne se propage pas).

JavaScript

Évènements de clavier

- `keydown` : Touche est enfoncée ⇒ Il se produit dès que la touche est enfoncée, avant même que le caractère associé ne soit affiché dans le champ de texte.
- `keyup` : Touche est relâchée ⇒ Il se produit après que la saisie de texte ait été effectuée et qu'une touche ait été relâchée.
- `keypress` : Touche est enfoncée et produit un caractère ⇒ Il se déclenche spécifiquement lorsqu'une touche produit un caractère qui peut être affiché (lettre, chiffre...). Il ne se déclenche pas pour des touches comme `Ctrl`, `Shift`, ou les touches de fonction.

JavaScript

Évènements de formulaire

- `input` : Valeur d'un élément de formulaire change (par saisie de texte, sélection...).
- `change` : Valeur d'un élément de formulaire est changée et le focus est perdu.
- `submit` : Formulaire est soumis.
- `reset` : Formulaire est réinitialisé.
- `focus` : Élément reçoit le focus.
- `blur` : Élément perd le focus.

JavaScript

Évènements d'interface utilisateur (UI)

- `resize` : Fenêtre ou un élément est redimensionné.
- `scroll` : Défilement d'un élément ou de la fenêtre.
- `load` : Document, ressources, images sont chargés.
- `unload` : Page est en train d'être déchargée.
- `beforeunload` : Événement se déclenche juste avant qu'une page ne soit déchargée.
- `DOMContentLoaded` : **DOM** est prêt à être manipulé (chargement complet du **HTML**).

JavaScript

Évènements de toucher (pour les appareils tactiles)

- `touchstart` : Doigt est posé sur l'écran tactile.
- `touchmove` : Doigt est déplacé sur l'écran tactile.
- `touchend` : Doigt est retiré de l'écran tactile.
- `touchcancel` : Toucher est interrompu.

JavaScript

Évènements de drag & drop

- `drag` : Élément est déplacé.
- `dragstart` : Début du déplacement (drag) d'un élément.
- `dragend` : Fin du déplacement d'un élément.
- `dragover` : Élément est survolé pendant le déplacement.
- `dragenter` : Élément entre dans une zone de drop valide.
- `dragleave` : Élément quitte une zone de drop.
- `drop` : Élément est lâché sur une zone de drop.

JavaScript

Évènements divers

- `error` : Erreur se produit lors du chargement d'une ressource externe (image, fichier script, etc.).
- `progress` : Progression du téléchargement d'une ressource.
- `animationstart`, `animationend`, `animationiteration` : Contrôlent les animations **CSS**.
- `transitionend` : Fin d'une transition **CSS**.

JavaScript

Deux façons différentes pour installer un écouteur d'évènement

- comme attribut d'une balise **HTML** en précédant le nom d'évènement par le préfixe `on` ayant comme valeur le nom d'une fonction **JavaScript**
- dans le fichier **JavaScript** avec la méthode `addEventListener()`

JavaScript

Première méthode (la page HTML)

```
<!DOCTYPE html>
<html lang="fr">

<head>
    <link rel="stylesheet" href="style.css">
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale
      =1.0">
    <title>Event Test</title>
</head>

<body>
    <div id=container>
        <input type=text id=nom>
        <button onclick="direBonjour()" id=cliquer> cliquer </button>
    </div>
    <script src="script.js"></script>
</body>

</html>
```

JavaScript

Contenu du `script.js`

```
function direBonjour(){
    var nom = document.getElementById('nom').value;
    alert ("Bonjour " + nom);
}
```

JavaScript

Première méthode (la page HTML)

```
<!DOCTYPE html>
<html lang="fr">

<head>
    <link rel="stylesheet" href="style.css">
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale
      =1.0">
    <title>Event Test</title>
</head>

<body>
    <div id=container>
        <input type=text id=nom>
        <button id=cliquer> cliquer </button>
    </div>
    <script src="script.js"></script>
</body>

</html>
```

JavaScript

Contenu du `script.js`

```
var cliquer = document.getElementById('cliquer');

cliquer.addEventListener('click', function() {
    var nom = document.getElementById('nom').value;
    alert ("Bonjour " + nom);
});
```

JavaScript

L'objet `event` permet de récupérer des données sur l'évènement (rien à modifier dans le HTML)

```
var cliquer = document.getElementById('cliquer');

cliquer.addEventListener('click', function (event) {
    console.log(event);
    var nom = document.getElementById('nom').value;
    alert("Bonjour " + nom);
});
```

JavaScript

Pour supprimer un évènement associé à un élément HTML

```
element.removeEventListener(event, nomFunction);
```

JavaScript

Commençons par définir les deux boutons suivants

```
<div id=container>
    <input type=text id=nom>
    <button id=cliquer> cliquer </button>
    <button id=cliquer2> cliquer deux fois </button>
</div>
```

JavaScript

En cliquant sur le premier

```
var cliquer = document.getElementById('cliquer');

cliquer.addEventListener('click', function() {
    var nom = document.getElementById('nom').value;
    alert ("Bonjour " + nom);
});
```

JavaScript

En cliquant sur le premier

```
var cliquer = document.getElementById('cliquer');

cliquer.addEventListener('click', function() {
    var nom = document.getElementById('nom').value;
    alert ("Bonjour " + nom);
});
```

En cliquant sur le deuxième, on re-déclenche le premier

```
var cliquer2 = document.getElementById('cliquer2');

cliquer2.addEventListener('dblclick', function() {
    cliquer.click()
});
```

JavaScript

Considérons la page `index.html` suivante

```
<!DOCTYPE html>
<html lang="fr">

<head>
    <link rel="stylesheet" href="style.css">
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale
      =1.0">
    <title>Prevent event</title>
</head>

<body>
    <div>
        Saisir seulement des caractères alphabétiques en minuscule :
        <input type="text" id="nom">
    </div>
    <script src="script.js"></script>
</body>

</html>
```

JavaScript

Pour annuler une saisie qui n'est pas en minuscule

```
let nom = document.getElementById('nom');

nom.addEventListener('keypress', function (event) {
    let pressedKey = event.key;
    if (pressedKey < 'a' || pressedKey > 'z') {
        event.preventDefault();
        alert("Merci de tout écrire en minuscule");
    }
});
```

JavaScript

On ne peut annuler que certains évènements

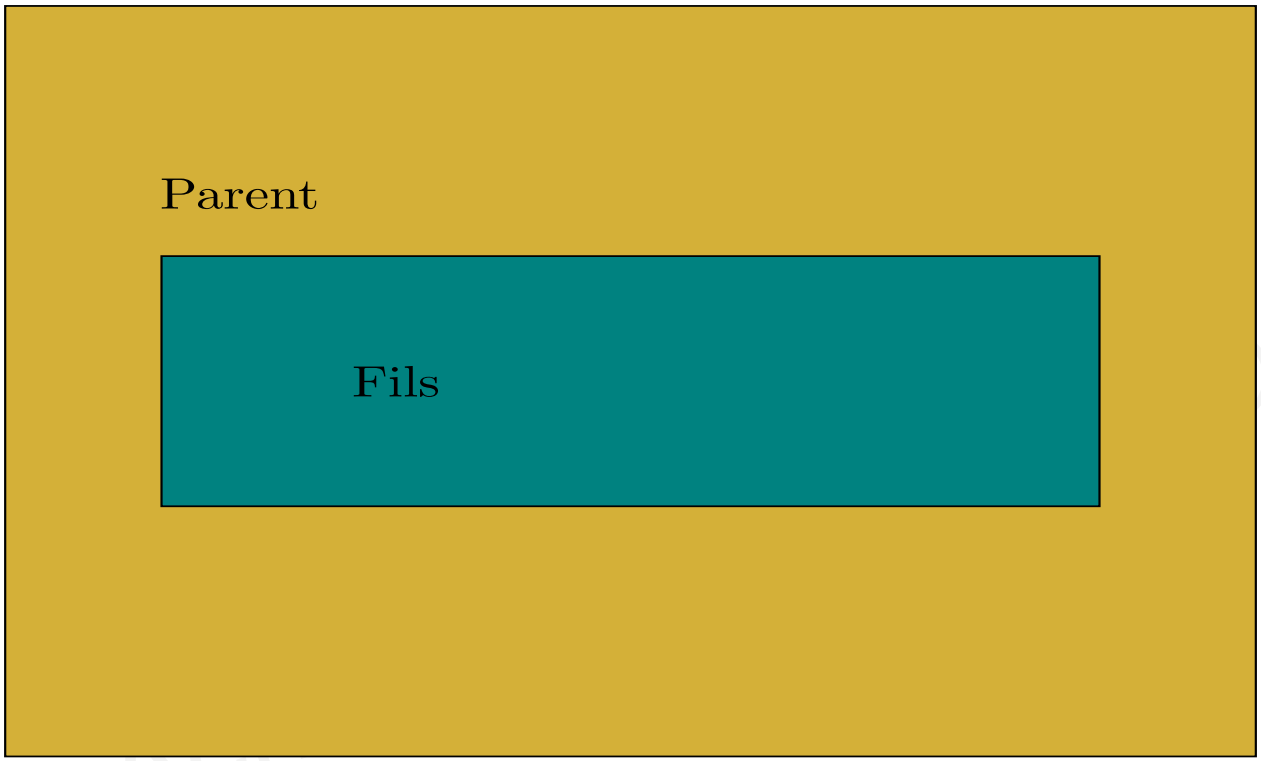
- `submit` : Empêche l'envoi d'un formulaire.
- `click` : Peut être utilisé pour empêcher le comportement par défaut d'un lien (par exemple, pour empêcher la navigation vers une **URL** spécifiée dans l'attribut `href` d'une balise `<a>`) ou une case à cocher.
- `mousedown` et `mouseup` : Peut être utilisé pour empêcher des actions par défaut lors du clic sur des éléments, comme la sélection de texte ou l'ouverture d'un menu contextuel.
- `keydown`, `keypress` et `keyup` : Peut être utilisé pour empêcher l'action par défaut des touches du clavier, comme empêcher le déplacement du curseur dans un champ de texte ou empêcher le déclenchement de raccourcis clavier du navigateur.
- `dragstart` : Empêche le début d'une opération de glisser-déposer.
- `contextmenu` : Empêche l'apparition du menu contextuel du clic droit sur un élément.
- `touchstart`, `touchmove` et `touchend` : Peuvent être utilisés pour empêcher le comportement par défaut sur les appareils tactiles, comme le défilement de la page ou le zoom.
- `focus` et `blur` : Bien que moins courants, empêcher ces événements peut être utile dans certains cas spécifiques pour gérer le focus manuellement.

JavaScript

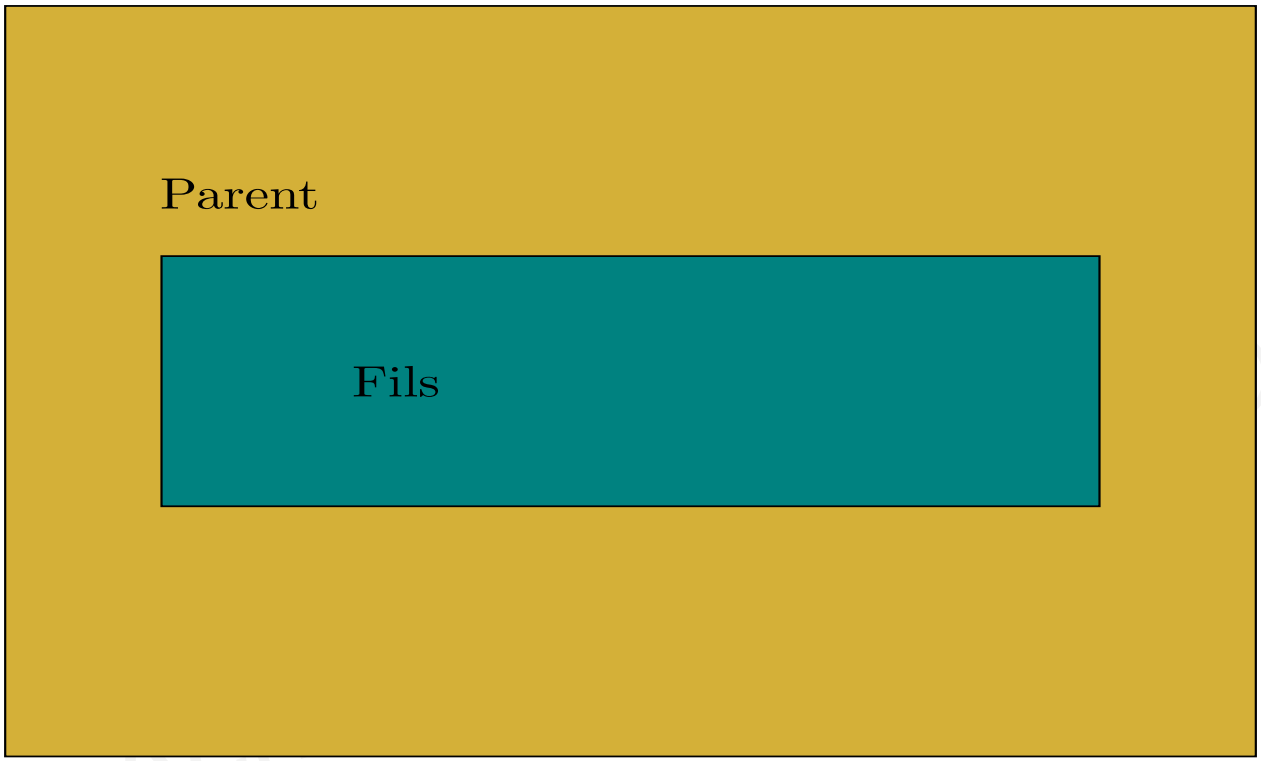
Considérons la page `index.html` suivante

```
<!DOCTYPE html>
<html lang="fr">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Stop propagation</title>
    <style>
        div {
            padding: 50px;
            cursor: pointer;
        }
    </style>
</head>

<body>
    <div style="background-color:gold" onclick="jaune()">
        Parent
        <div style="background-color:teal" onclick="vert()">
            Fils
        </div>
    </div>
    <script src="script.js"></script>
</body>

</html>
```

Les deux div parent et fils



Les deux div parent et fils



Code JavaScript de `vert()` et `jaune()`

```
function vert(){
  alert("vert");
}
function jaune(){
  alert("jaune");
}
```

JavaScript

Constats

- En cliquant sur le jaune, jaune est affiché
- En cliquant sur le vert, jaune et vert sont affichés

JavaScript

Constats

- En cliquant sur le jaune, jaune est affiché
- En cliquant sur le vert, jaune et vert sont affichés

Solution

Utiliser la méthode `stopPropagation`

JavaScript

Nouveau code de la fonction `vert()` (pas de changement pour `jaune()`)

```
function vert(event){
  alert("vert");
  event.stopPropagation();
}

function jaune(){
  alert("jaune");
}
```

JavaScript

Nouveau code de la fonction `vert()` (pas de changement pour `jaune()`)

```
function vert(event){
  alert("vert");
  event.stopPropagation();
}

function jaune(){
  alert("jaune");
}
```

Une modification dans le HTML

```
<div style="background-color:gold" onclick="jaune()" >
  Parent
  <div style="background-color:teal" onclick="vert(event)">
    Fils
  </div>
</div>
```

JavaScript

Remarques

- A l'appel de la fonction `vert`, le paramètre doit être nommé `event`.
- A la déclaration de la fonction `vert`, le paramètre peut avoir n'importe quel nom.