

Ce que l'on a vu

1. HTML

- ✓ les balises et les éléments
- ✓ les attributs
- ✓ l'imbrication
- ✓ navigateurs et protocole HTTP

2. CSS

- ✓ la syntaxe
- ✓ les sélecteurs : simples, combineurs, pseudo-classes
- ✓ les propriétés : texte et blocs, `display`, marges et padding, etc.
- ✓ l'agencement Flex (Flexible Box)

Dans ces diapos

1. Les concepts du Javascript

- ✓ dynamique

2. la syntaxe

- ✓ variables, types et opérateurs
- ✓ les tableaux
- ✓ les objets

3. Manipulation du DOM

- ✓ sélection globale d'éléments
- ✓ sélection par filiation
- ✓ manipuler un élément : attributs, classes, contenu
- ✓ modifier le DOM : créer/ajouter/supprimer

4. Événements

5. Requêtes asynchrones

- ✓ API fetch

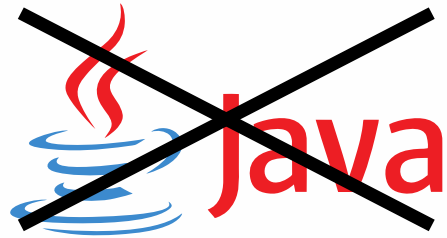


Javascript

(*Abrégé JS*) C'est un langage de script orienté objet qui permet d'ajouter des interactions statiques et dynamique aux page HTML :

- ✓ manipulation des éléments HTML (ajouter, modifier, supprimer)
- ✓ chargement dynamique du contenu (requêtes asynchrones)
- ✓ gestion des événements (clics, survols, ...)

Mais Javascript...



Attention, Javascript :

- ✓ n'est pas du Java interprété, c'est un langage indépendant avec des concepts très différents.
- ✓ est aussi utilisé en dehors des navigateurs web, comme par exemple avec **Node.js** ou **CouchDB**

Intégration à HTML

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <script> /* Code */ </script>
    <title>Ma page</title>
  </head>
  <body>
    ...
    <script> /* Autre code */ </script>
  </body>
</html>
```

index_with_code.html

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <script src="script1.js"></script>
    <title>Ma page</title>
  </head>
  <body>
    ...
    <script src="script2.js"></script>
  </body>
</html>
```

index_with_file.html

✓ Le code JS peut être écrit :

1. directement dans l'élément `<script>`
2. dans un fichier externe référencé avec l'attribut `src` de `<script>`

✓ Un document HTML peut contenir plusieurs éléments `<script>`

```
// Premier type de commentaire sur une seule ligne

/* Second type de commentaire sur plusieurs lignes.
 * Seules les étoiles initiales et finales (après et
 * avant les deux slashes) sont nécessaires, les autres
 * c'est de la décoration */
```

La syntaxe des commentaires est identique à celle de Java et C# :

- ✓ sur une ligne avec `//`
- ✓ sur une ou plusieurs lignes avec `/* */`

Les variables

```
/* Déclarer une variable  
 * avec var */  
var a;  
var b = 5;
```

var est l'instruction la plus ancienne :

- ✓ ces variables ont une portée de fonction
- ✓  À ÉVITER AU MAXIMUM 

```
/* Déclarer une variable  
 * avec let */  
let c;  
let d = 5;
```

let est une instruction plus récente :

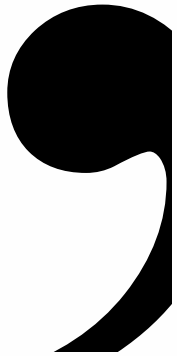
- ✓ ces variables ont une portée de bloc
- ✓ elle est à préférer à **var**

```
/* Déclarer une constante  
 * avec const */  
const E = 5;
```

const permet de déclarer des constantes :

- ✓ elle doit être affectée à la déclaration
- ✓ sa valeur ne pourra pas être modifiée

Point virgule et casse



- ✓ Le point-virgule n'est pas nécessaire mais il est très fortement recommandé et d'usage de le mettre après chaque instruction.
- ✓ Tous les noms de variables sont quasiment possibles, à l'exception des mots-clés réservés.
- ✓ Javascript est sensible à la casse : `maVariable` et `MaVariable` sont deux variables différentes !

Typage

```
let a;  
typeof a;  
// "undefined"
```

```
let b = true;  
typeof b;  
// "boolean"
```

```
let c = 12;  
typeof c;  
// "number"
```

```
let d = 12.5;  
typeof d;  
// "number"
```

```
let e = 'CAWEB';  
typeof e;  
// "string"
```

- ✓ Javascript est un langage à **typage dynamique** : il n'y a pas de type déclaré... mais il y a un type quand même !
- ✓ Il n'y a qu'un seul **type** pour les nombres, pas de distinction entre les entiers et les décimaux.
- ✓ Les chaînes de caractères peuvent se déclarer avec des guillemets simples ou doubles.

Attention au typage dynamique !

```
let a = 10;           // a est un nombre  
alert(typeof a);      // affiche "number"  
a = "dix";           // a est maintenant une chaîne  
alert(typeof a);      // affiche "string"
```

Opérateurs

`=` permet d'affecter des variables.

`+`

permet d'additionner des nombres et de concaténer des chaînes.

```
let a = 0+2; // 2
let b = 'M'+ '2'; // M2
```

`==` et `!=` : comparateurs “faibles”

Les opérandes sont transtypés avant d'être comparés.

```
let c = (5 == '5'); // Vrai
let d = (5 != '5'); // Faux
```

`===` et `!==` : comparateurs “stricts”

Les opérandes ne sont pas transtypés avant d'être comparés.

```
let e = (5 === '5'); // Faux
let f = (5 !== '5'); // Vrai
```

Les tableaux

```
// Déclarer un tableau  
let tab1 = [ 11, 22 ];  
let tab2= [ "A", 11, "B", 22 ];
```

```
// Taille d'un tableau  
let size1 = tab1.length; // 2
```

```
// Accéder à un élément  
let val1 = tab1[0]; // 11
```

```
// Modifier un élément  
tab2[0] = "S";
```

```
// Indice d'un élément  
let i = tab2.indexOf("A"); // 1
```

```
// Ajouter à la fin  
tab1.push(33); // [ 11, 22, 33 ]
```

```
// Supprimer le dernier  
tab1.pop(); // [ 11, 22 ]
```

```
// Supprimer le premier  
tab1.shift(); // [ 22 ]
```

```
// Ajouter au début  
tab1.unshift(33); // [ 33, 22 ]
```

```
// Trier  
tab1.sort(); // [ 22, 33 ]
```

- ✓ Les tableaux peuvent contenir des éléments de types différents.
- ✓ La classe `Array` implémente de nombreuses fonctions permettant de manipuler les tableaux : en plus des fonctions ci-dessus, vous pouvez regarder `join()`, `filter()`, `reduce()`, ...

Les objets, level 1

```
let moi = {  
  nom: 'Papeur',  
  prenom: 'Jé',  
};
```

```
let lui = {};  
lui.nom = 'peur';  
lui.prenom = 'Ila';  
lui.age = 20;
```

```
alert( moi.prenom ); // "Jé"  
alert( lui['nom'] ); // "peur"  
alert( lui.age );    // 20
```

Un objet Javascript :

- ✓ se déclare avec des accolades
- ✓ est une structure associative (≈ dictionnaire) `property: value`.
- ✓ peut se voir attribuer de nouvelles propriétés dynamiquement

Les objets, level 2

```
let moi = {  
  prenom: 'Julien',  
  age: 28,  
  frere: {  
    prenom: 'Mathieu',  
    age: 25  
  },  
  sports: [ 'Échecs', 'Curling' ]  
};
```

```
alert( moi.prenom );           // "Julien"  
alert( moi.frere.prenom );     // "Mathieu"  
alert( moi.sports.length );    // 2
```

Une propriété d'un objet :

- ✓ peut être un type primitif, un tableau, un objet ou une fonction.
- ✓ peut être accéder avec `obj.property` ou `obj['property']`

Dialogue avec l'utilisateur

```
let nom = prompt('Entrez votre nom :');  
let classe = prompt('Entrez votre classe :');  
alert('Bonjour ' + nom + '. Vous êtes un ' + classe);  
  
let start = confirm("Souhaitez-vous démarrer l'aventure ?");  
if (start) {  
    alert("C'est parti !");  
}
```

alert()

Affiche du texte dans une fenêtre surgissante (popup).

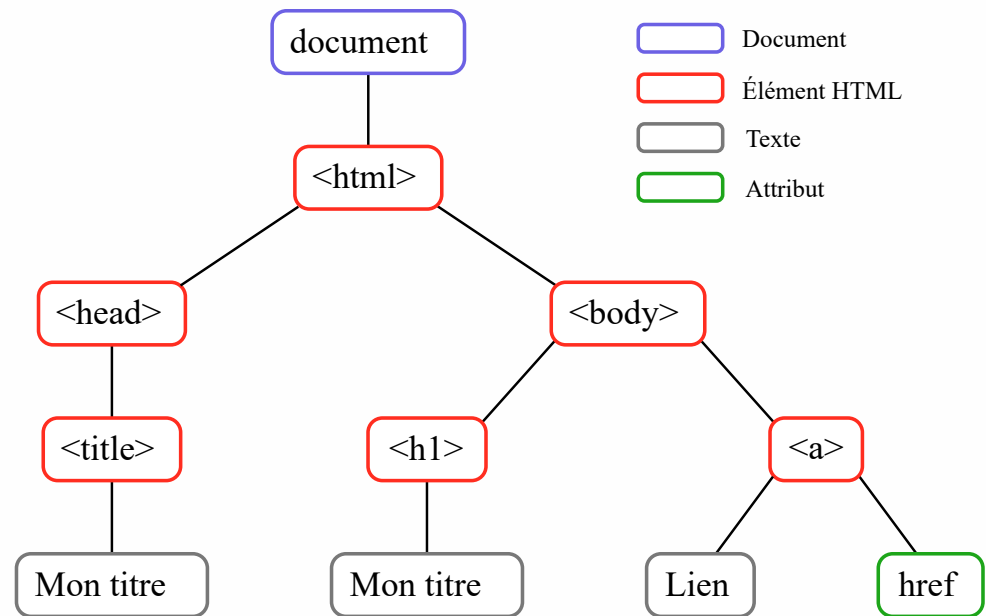
let val = prompt()

Demande à l'utilisateur de saisir une valeur. La valeur sera `null` en cas d'appui sur le bouton "Annuler".

let val = confirm()

Propose un choix booléen à l'utilisateur. La valeur dépend du clic sur l'un des boutons "Ok" et "Annuler".

```
<html>
  <head>
    <title>Mon titre</title>
  </head>
  <body>
    <h1>Mon titre</h1>
    <a href="#">Lien</a>
  </body>
</html>
```

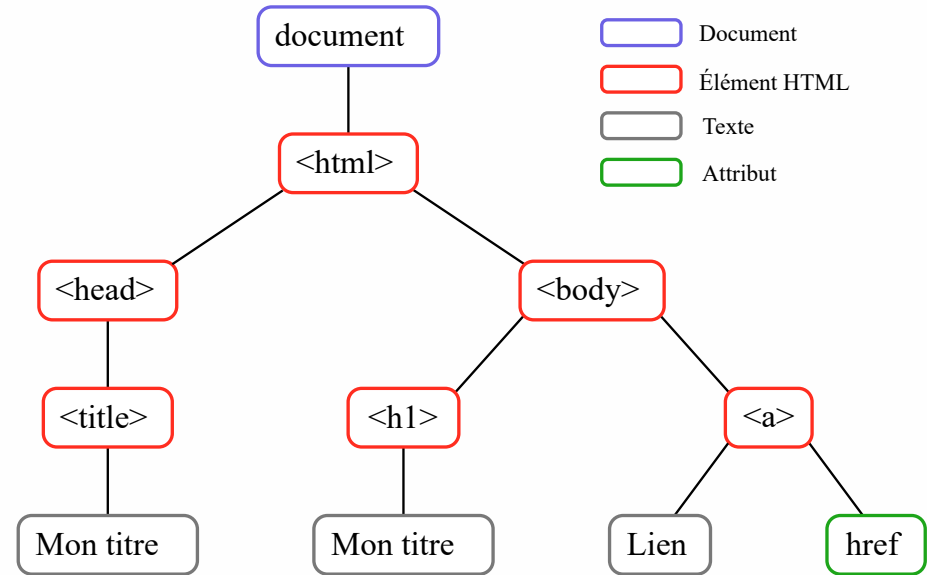


Le Document Object Model est une représentation arborescente d'un document HTML qui peut être manipulé pour :

- ✓ ajouter et supprimer des éléments
- ✓ modifier un élément : son contenu, ses attributs ou son style

Le DOM en Javascript

```
<html>
  <head>
    <title>Mon titre</title>
  </head>
  <body>
    <h1>Mon titre</h1>
    <a href="#">Lien</a>
  </body>
</html>
```



L'objet **document**

regroupe un ensemble de fonctionnalités globales pour manipuler le DOM.

La classe **Element**

représente n'importe quel élément HTML et permet de manipuler ses attributs et ses propriétés CSS.


```
let elem = document.getElementById('monId');
```

Retourne l'unique élément d'identifiant `monId`.

```
<body>
  <p id="monId">Les jeux :</p>
  <article class="recent">
    <h3>Medina</h3>
    <a href="m.html">lien</a>
    <p id="i"> Pas mal </p>
    <p> Surprenant </p>
  </article>
  <article class="old">
    <h3>Kingdomino</h3>
    <a href="k.html">lien</a>
    <p> Cool </p>
  </article>
</body>
```

Selection par balise

```
let elems = document.getElementsByTagName('a');
```

Retourne un tableau contenant tous les éléments de type `<a>`.

```
<body>
  <p id="monId">Les jeux :</p>
  <article class="recent">
    <h3>Medina</h3>
    <a href="m.html">lien</a>
    <p id="i"> Pas mal </p>
    <p> Surprenant </p>
  </article>
  <article class="old">
    <h3>Kingdomino</h3>
    <a href="k.html">lien</a>
    <p> Cool </p>
  </article>
</body>
```

Sélection par classe

```
let elems = document.getElementsByClassName('recent');
```

Retourne un tableau contenant tous les éléments de classe `recent`.

```
<body>
  <p id="monId">Les jeux :</p>
  <article class="recent">
    <h3>Medina</h3>
    <a href="m.html">lien</a>
    <p id="i"> Pas mal </p>
    <p> Surprenant </p>
  </article>
  <article class="old">
    <h3>Kingdomino</h3>
    <a href="k.html">lien</a>
    <p> Cool </p>
  </article>
</body>
```

Sélection CSS

```
let elem = document.querySelector('article > h3');
```

Retourne le 1^{er} élément correspondant au sélecteur CSS `article > h3`.

```
<body>
  <p id="monId">Les jeux :</p>
  <article class="recent">
    <h3>Medina</h3>
    <a href="m.html">lien</a>
    <p id="i"> Pas mal </p>
    <p> Surprenant </p>
  </article>
  <article class="old">
    <h3>Kingdomino</h3>
    <a href="k.html">lien</a>
    <p> Cool </p>
  </article>
</body>
```

Sélection CSS multiple

```
let elems = document.querySelectorAll('article > h3');
```

Retourne tous les éléments correspondant au sélecteur `article > h3`.

```
<body>
  <p id="monId">Les jeux :</p>
  <article class="recent">
    <h3>Medina</h3>
    <a href="m.html">lien</a>
    <p id="i"> Pas mal </p>
    <p> Surprenant </p>
  </article>
  <article class="old">
    <h3>Kingdomino</h3>
    <a href="k.html">lien</a>
    <p> Cool </p>
  </article>
</body>
```

La classe `Element` propose plusieurs méthodes pour accéder aux éléments relatifs à l'élément courant.

```
let p = document.querySelector('.recent');  
// Le parent de p  
let body = p.parentElement;  
// Les enfants de p  
let childs = p.childNodes;  
// Le premier enfant de p  
let h3 = p.firstElementChild;  
// Le voisin direct de p  
let a = p.nextElementSibling;
```

```
<body>  
<p id="monId">Les jeux :</p>  
<article class="recent">  
  <h3>Medina</h3>  
  <a href="m.html">lien</a>  
  <p id="i"> Pas mal </p>  
  <p> Surprenant </p>  
</article>  
<article class="old">  
  <h3>Kingdomino</h3>  
  <a href="k.html">lien</a>  
  <p> Cool </p>  
</article>  
</body>
```

`nextElementSibling`

Accéder aux attributs HTML

```
<article class="recent">
  <h3>Médina</h3>
  <a href="http://m.html">lien</a>
  <p id="i">Pas mal</p>
</article>
```

```
let url = document.querySelector('a').href;
let id = document.querySelector('p').id;
```

Médina

lien

Pas mal

On accède à la valeur de l'attribut `att` d'un élément HTML `elem` avec la notation `elem.att`.

Manipuler les classes

```
<article class="recent">
  <h3>Médina</h3>
  <a href="http://m.html">lien</a>
  <p id="i">Pas mal</p>
</article>
```

```
let p = document.getElementById('i');
p.classList.add('droite', 'vert');
p.classList.remove('droite');
p.classList.toggle('vert');
```

```
.droite { text-align: right; }
.vert   { background-color: green; }
```

Médina

lien

Pas mal

Le membre `classList` de la class `Element` permet de manipuler les classes d'un élément, notamment :

- ✓ d'en ajouter avec `add(String [, String])`
- ✓ d'en supprimer avec `remove(String [, String])`
- ✓ d'en inverser la présence avec `toggle(String)`

Modifier le contenu

```
<article class="recent">
  <h3>Médina</h3>
  <a href="http://m.html">lien</a>
  <p id="i">Pas mal</p>
</article>
```

```
let p = document.getElementById('i');
p.innerHTML = "Incroyable !";
p.innerHTML = "<a href='t.html'>lien 2</a>";
```

Médina

[lien](#)

Pas mal

Le membre `innerHTML` de la class `Element` donne un accès direct en écriture au contenu de l'élément qui peut être :

- ✓ du texte
- ✓ du contenu HTML

Combo : manipuler le DOM

```
<article class="recent">
  <h3>Médina</h3>
  <a href="http://m.html">lien</a>
  <p id="i">Pas mal</p>
</article>
```

```
let ar = document.getElementsByTagName('article')[0];

let p = document.createElement("p");
p.innerHTML = "Incroyable !";
ar.appendChild(p); // Ajout

let last = ar.lastElementChild;
ar.removeChild(last); // Suppression
```

Médina

lien

Pas mal

`document.createElement(tagName)`

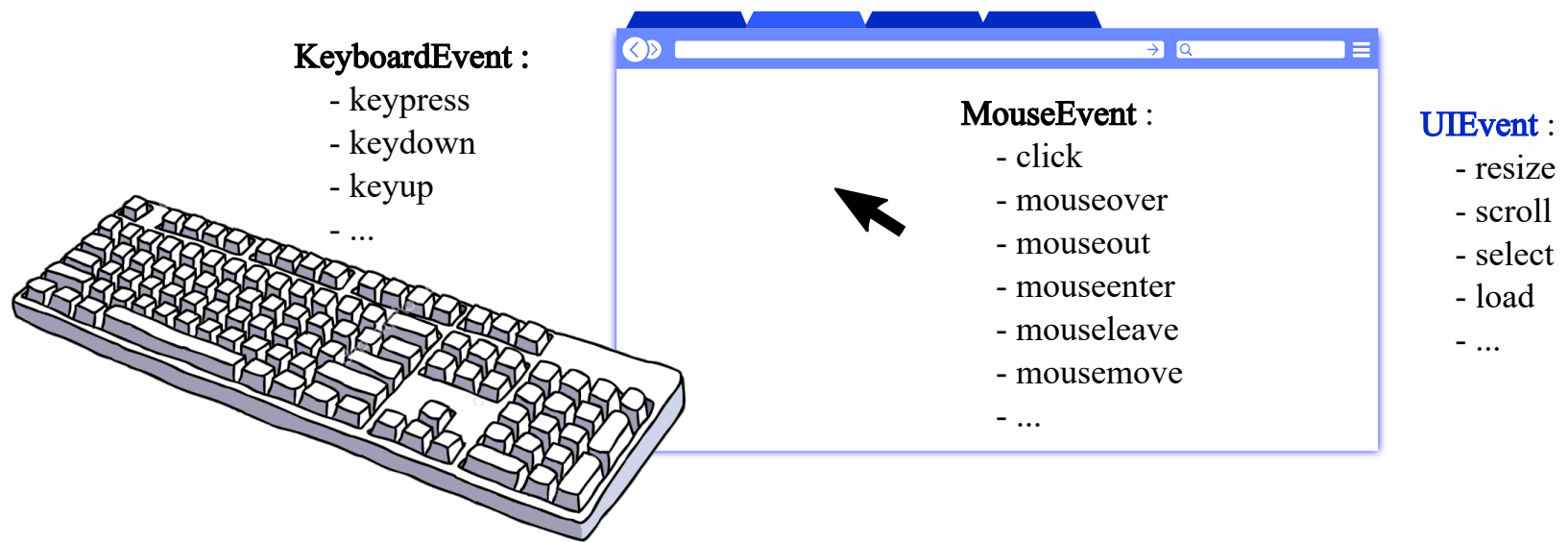
Crée un élément.

`elemA.appendChild(elemB)`

Ajoute `elemB` comme enfant de `elemA`

`elemA.removeChild(elemB)`

Supprime `elemB` des enfants de `elemA`.



La class **Event**

Elle représente n'importe quel événement survenu lors de l'interaction de l'utilisateur avec le DOM. Tous les événements sont notifiés au DOM et JavaScript permet :

- ✓ d'écouter ces événements
- ✓ d'exécuter du code lorsqu'ils surviennent.

Exemple avec la souris

```
function changeColor(event) {  
    event.preventDefault();  
    let a = event.target;  
    a.style.color = 'green';  
}  
  
let h = document.getElementsByTagName('h3')[0];  
  
h.addEventListener('click', changeColor);  
h.nextElementSibling  
    .addEventListener('mouseover', changeColor);
```

```
<article class="recent">  
<h3>Médina</h3>  
<a href="http://m.html">lien</a>  
<p id="i">Pas mal</p>  
</article>
```

Médina

lien

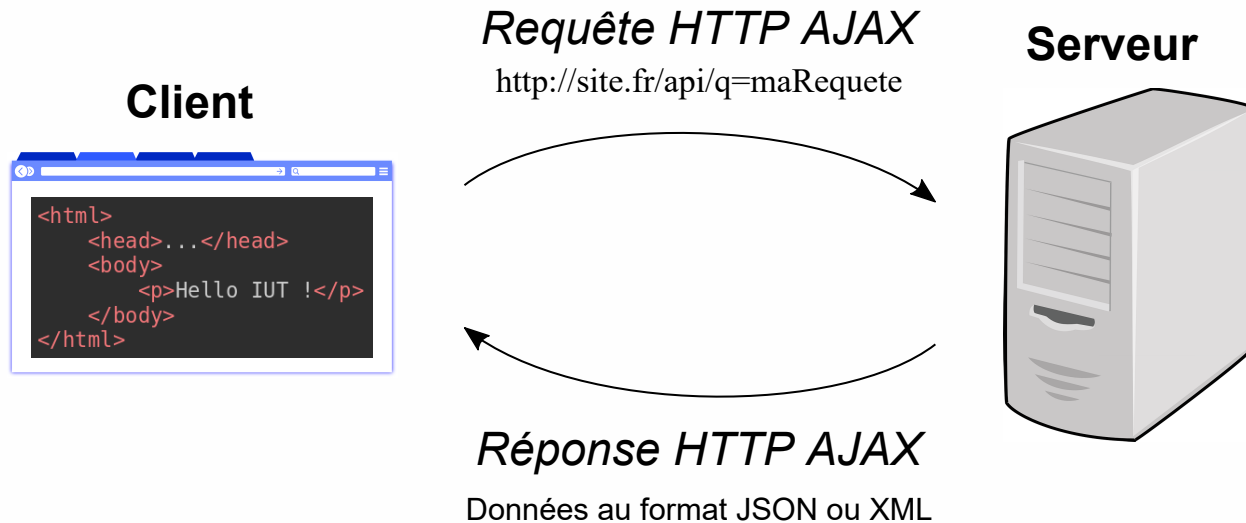
Pas mal

Pour tester, cliquer sur “Médina” et survoler “lien”.

- ✓ La fonction passée en paramètre de `addEventListener` prend un paramètre de type `Event` qui contient des informations sur l'événement déclenché.
- ✓ La méthode `preventDefault()` permet d'empêcher l'exécution du comportement par défaut de l'élément HTML associé à cet événement.

AJAX (*Asynchronous Javascript And XML*)

Méthode de gestion des requêtes asynchrones en JavaScript



Une requête asynchrone :

- ✓ est une requête non bloquante.
- ✓ permet de continuer à manipuler la page web jusqu'à réception de la réponse.

Fetch

```
fetch( URL ).then( fonction1 ).then( fonction2 ).catch( fonctionErreur );
```

```
fetch( URL )  
.then( function (response) { /* Exécuté lorsque l'on reçoit la réponse */ })  
.then( function (data)      { /* Exécuté lorsque le 1er then est terminé */ })  
.catch( function (error)    {  
    // Exécuté lorsqu'une exception est  
    // déclenchée dans l'un des blocs fetch  
    // ou then.  
});
```

Fetch permet d'effectuer des traitements asynchrones en cascade :

- ✓ `fetch()` envoie une requête HTTP et transmet la réponse.
- ✓ un bloc `then` exécute une fonction dont l'argument correspond au à la valeur de retour du bloc précédent.
- ✓ un bloc `catch` est exécuté lorsqu'une erreur survient dans l'un des blocs précédents.

Un exemple concret

```
fetch('http://monsite.fr/api/?s=bond')
.then( function (response) {
    if (response.ok)
        return response.json();
    throw new Error('Response is not OK');
})
.then( function (data) {
    doSomething(data);
})
.catch( function (error) {
    console.log(error.message);
});
```

```
// Contenu de la réponse au format JSON
{
    'nom': 'Bond',
    'prenom': 'James',
    'age': 35
}
```

```
function doSomething(data) {
    const p = document.querySelector('#p');
    p.innerHTML = `
        Vous êtes ${data.prenom} ${data.nom}.
        Vous avez ${data.age} ans.`;
}
```

Le retour de `fetch` est de type `Response` et permet :

- ✓ de savoir si c'est un succès avec le booléen `ok`.
- ✓ d'obtenir le `status` de la réponse HTTP (200, 400, etc.).
- ✓ d'obtenir les données aux formats JSON ou texte avec les méthodes `.json()` et `.text()`.