



M4103C - Programmation Web Orientée Client

---

# C2 – AJAX & JQUERY

2019 - Carl VINCENT ( [carl.vincent@iut2.univ-grenoble-alpes.fr](mailto:carl.vincent@iut2.univ-grenoble-alpes.fr) )

(Basé sur le cours de Christophe BROUARD)

# SOMMAIRE DU COURS



1. Les requêtes AJAX
2. Les différences entre navigateurs et les déclarations de variables
3. La librairie jQuery

## AJAX : LE SIGLE



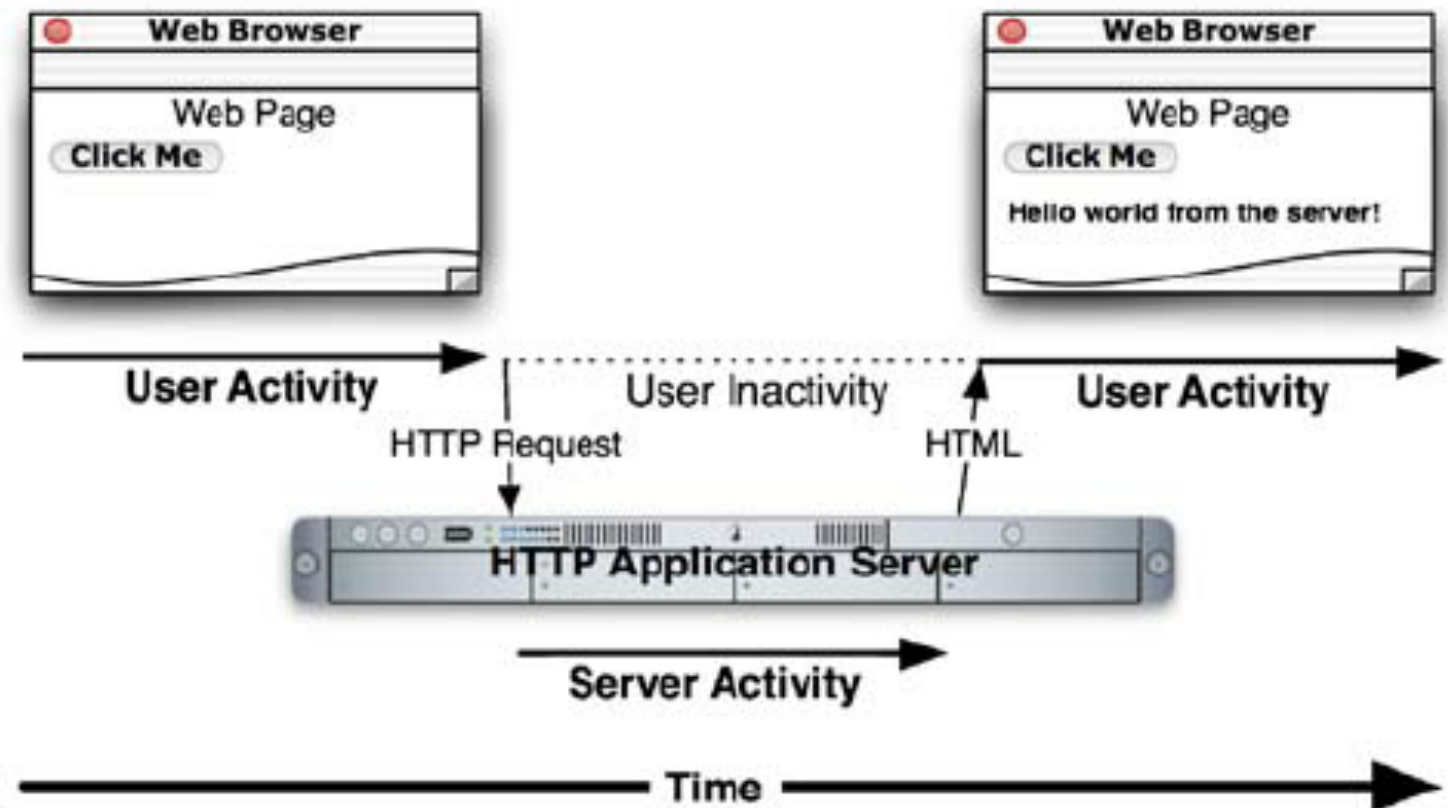
- ▶ **Asynchronous Javascript And XML** :
  - ➔ **Asynchrone** : Communication client-serveur qui se fait sans avoir à recharger la page (donc pas de blocage côté client)
  - ➔ **JavaScript** : L'appel AJAX et le traitement des données reçues se fait en JS
  - ➔ **XML** : Format standardisé des données échangées (MAIS aujourd'hui, le format XML est souvent remplacé par un format plus "compact" : le **JSON**)
- ▶ L'**AJAX** n'est donc pas une nouvelle technologie, mais **une combinaison de technologies** qui permettent **d'améliorer la rapidité et le confort d'utilisation des sites/appli web riches**

( Pour plus de détails : [OpenClassrooms - Récupérez des données d'un service web \(AJAX\)](#) )

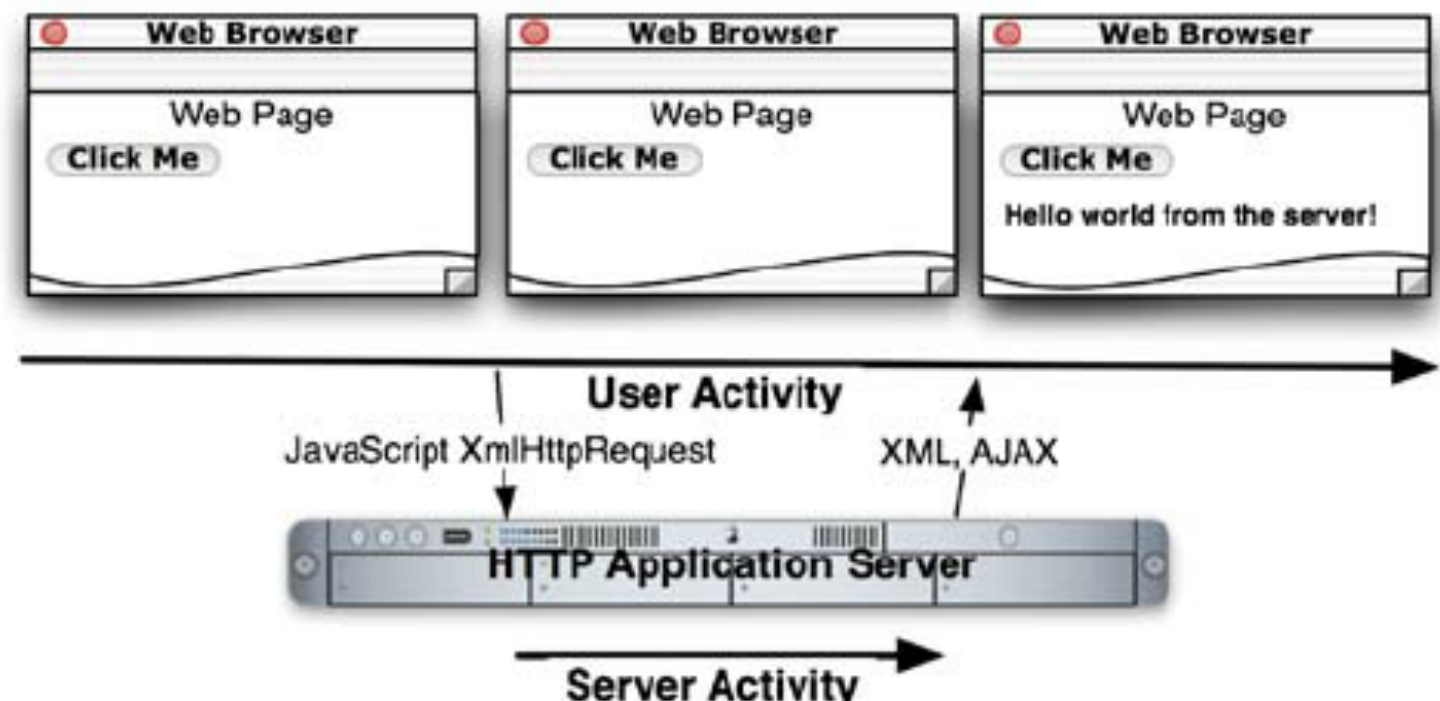
## AJAX : LE PRINCIPE

- ▶ Il s'agit d'un ensemble d'objets et de fonctions mis à disposition par le langage JavaScript, afin d'exécuter des requêtes HTTP de manière **asynchrone**
- ▶ L'AJAX permet du coup d'exécuter des requêtes HTTP **sans avoir besoin de recharger la page du navigateur** (on recharge que ce qui est nécessaire)

Traditional Web Interaction Model (Synchronous)

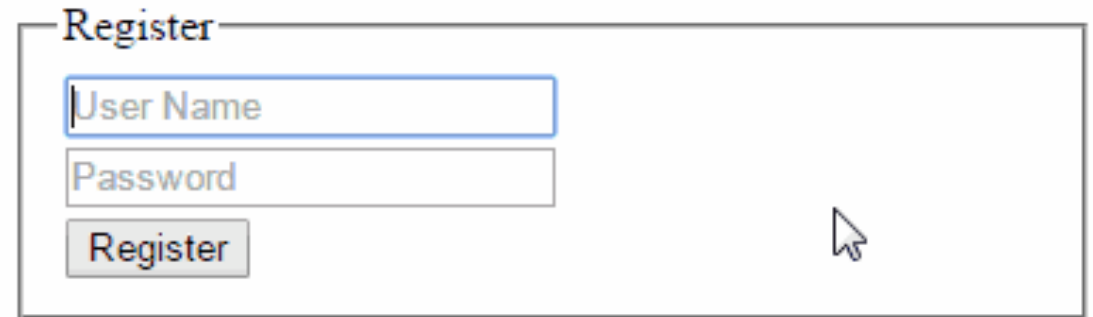


AJAX Web Interaction Model (Asynchronous)



# AJAX : LES CAS D'UTILISATION

- ▶ L'**AJAX** permet entre autres :
  - La vérification de formulaire en direct (avertissement si problème sur un champ)
  - L'auto-complétion sur un champ de recherche
  - La création de page avec "infinite scrolling" (le contenu se charge quand on défile la page)
  - Appeler l'API d'un Service Web (cf. démo un peu après)
- ➔ Cela permet globalement d'améliorer **la réactivité d'un site**, et du coup **l'expérience utilisateur**

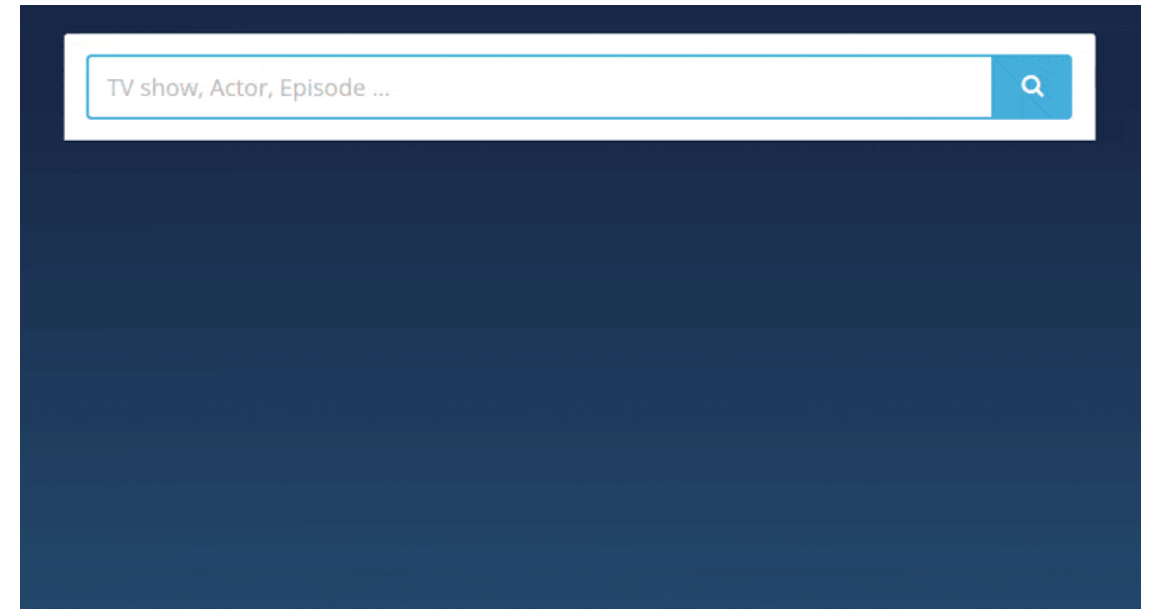


Register

User Name

Password

Register



TV show, Actor, Episode ...



## DIFFÉRENTS FORMATS DE DONNÉES : XML vs. JSON

- ▶ Le **XML** est un langage à balises standardisé (proche du HTML)

```
<employees>
  <employee>
    <firstName>John</firstName> <lastName>Doe</lastName>
  </employee>
  <employee>
    <firstName>Anna</firstName> <lastName>Smith</lastName>
  </employee>
  <employee>
    <firstName>Peter</firstName> <lastName>Jones</lastName>
  </employee>
</employees>
```

- ▶ Le **JSON** (JavaScript Object Notation) est un format de données qui a été spécialement conçu pour l'échange entre le navigateur et le serveur

```
{ "employees": [
  { "firstName": "John", "lastName": "Doe" },
  { "firstName": "Anna", "lastName": "Smith" },
  { "firstName": "Peter", "lastName": "Jones" }
]}
```

=> Le JSON est globalement **plus compact**, et **plus rapide à parser** !

( Article à lire : [w3schools - JSON vs XML](#) )

## LE "PARSING" DU JSON

- Pour les requêtes AJAX, la nouvelle norme étant de renvoyer **une chaine au format JSON**, vous aurez besoin de "**parser**" cette chaine pour pouvoir **manipuler les résultats comme un objet JS classique**

```
var text = '{ "employees" : [' +  
  '{ "firstName":"John" , "lastName":"Doe" },' +  
  '{ "firstName":"Anna" , "lastName":"Smith" },' +  
  '{ "firstName":"Peter" , "lastName":"Jones" } ]}';  
  
// Parsing de la chaine au format JSON  
// (ATTENTION, si la chaine est vide => SyntaxError !!)  
var obj = JSON.parse(text);  
console.log(obj.employees[1].lastName); // Smith
```

- À l'inverse, pour transformer un objet JS en une chaine au format JSON, il faut utiliser la fonction **JSON.stringify()**

```
var obj = { liste:[1,2,3], valeur:42 };  
console.log(JSON.stringify(obj)); // {"liste":[1,2,3], "valeur":42}
```

## L'OBJET XMLHttpRequest : EXEMPLE POUR REQUÊTE GET

- ▶ **XMLHttpRequest** est une "classe" standardisée qui est implémentée par tous les navigateurs actuels (et qui offre la possibilité de faire de l'**AJAX**)
- ▶ Elle permet d'échanger des données textuelles (**XML ou JSON**) avec un serveur via des requêtes HTTP (**GET ou POST**) en synchrone ou async.

```
function ajax_get_request(callback, url, async) {  
    var xhr = new XMLHttpRequest(); // Création de l'objet  
  
    // Définition de la fonction à exécuter à chaque changement d'état  
    xhr.onreadystatechange = function(){  
        if (callback && xhr.readyState == 4 && xhr.status == 200){  
            // Si le serveur a fini son travail  
            // et que le code d'état indique que tout s'est bien passé  
            // => On appelle la fonction callback en lui passant la réponse  
            callback(xhr.responseText);  
        }  
    };  
  
    xhr.open("GET", url, async); // Initialisation de l'objet  
    xhr.send(); // Envoi de la requête  
}
```



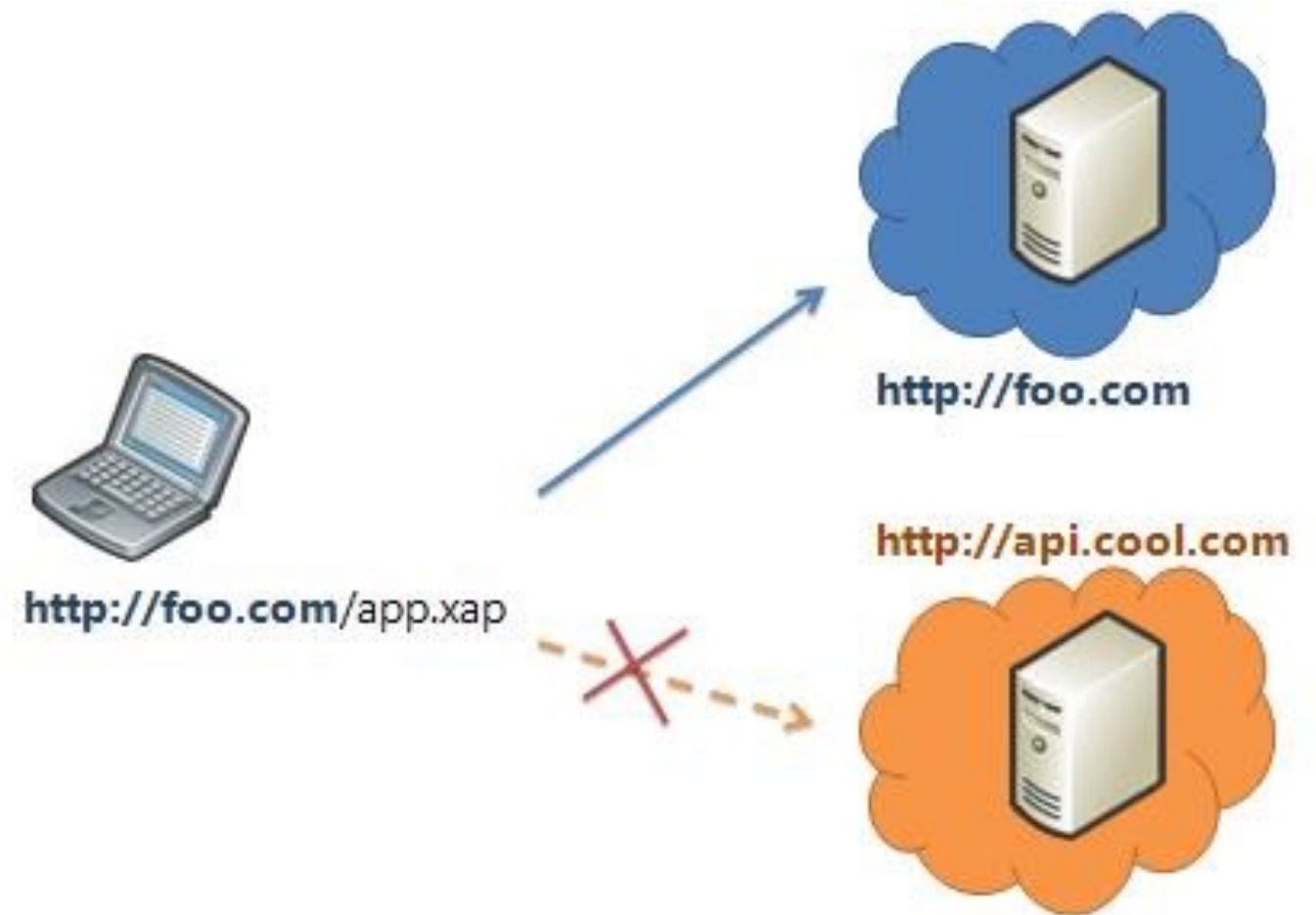
## L'OBJET XMLHttpRequest : EXEMPLE POUR REQUÊTE POST

- ▶ Pour une requête **POST**, la principale différence se trouve au niveau du passage des paramètres : On les passe en argument de **send()** (avec la même syntaxe : "param1=valeur1&param2=valeur2")
- ▶ Il faut aussi **changer le type MIME de la requête** avec la méthode **setRequestHeader(...)**, sinon le serveur ignorera la requête !

```
function ajax_post_request(callback, url, sync, data) {  
    var xhr = new XMLHttpRequest(); // Création de l'objet  
    xhr.onreadystatechange = function() {  
        if (callback && xhr.readyState == 4 && xhr.status == 200){  
            callback(xhr.responseText);  
        }  
    };  
  
    xhr.open("POST", url, sync); // Initialisation de l'objet  
  
    // Format des données (MIME) envoyées dans le corps de la requête HTTP  
    xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");  
  
    xhr.send("data=" + encodeURIComponent(data)); // Envoi de la requête  
}
```

## MAIS ATTENTION AU CROISEMENT DE DOMAINE (CROSS DOMAIN)

- ▶ Par défaut, il y a une restriction qui bloque un appel AJAX vers un serveur/domaine différent à celui d'origine ([Same-origin policy](#))
- ▶ Mais il est possible d'autoriser le "[Cross-Origin Resource Sharing \(CORS\)](#)", mais cela doit se faire avec précaution !



```
<?php
// Autorisation de toutes les requêtes
// provenant de n'importe quel domaine
header( "Access-Control-Allow-Origin: *" );
```

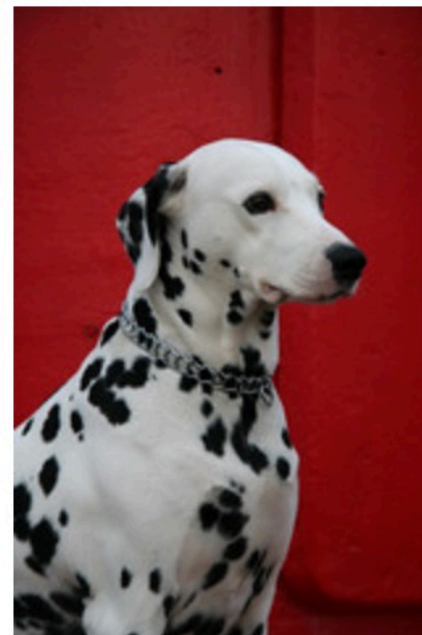
( Pour plus de détails : [OpenClassrooms - L'XMLHttpRequest cross-domain](#) )

( Et si ça bloque côté client (JavaScript) => Utiliser un "proxy" :

[How to bypass 'Access-Control-Allow-Origin' error with XMLHttpRequest](#) )

## DÉMO AJAX : RÉCUPÉRATION D'INFOS VIA UNE API PUBLIQUE

**Démo AJAX : Récupération des dernières images publiques de chien via l'API Flickr**





# LES DIFFÉRENCES ENTRE LES NAVIGATEURS

- ▶ Il faut garder en tête que votre code JS va s'exécuter "côté client", et qu'il existe **des différences entre les "moteurs JS" des navigateurs** !
- ▶ Avant d'utiliser une nouvelle fonction (ou élément de langage), il faut donc **vérifier que cela fonctionne bien sur la plupart des navigateurs**
- ▶ Vous trouverez ces infos sur : [CanIUse.com](https://caniuse.com) ou [MDN web docs](https://developer.mozilla.org/fr/docs/Web/JavaScript)

Can I use ? Settings

# JavaScript statement: for...of

Current aligned

Usage relative

Date relative

Apply filters

Show all

?

IE	Edge	Firefox	Chrome	Safari	Opera	iOS
		2-12	4-37	3.1-7.1	10-24	3..
6-10	12-17	13-68	38-76	8-12	25-60	8
11	18	69	77	12.1	62	
	76	70-71	78-80	13-TP		

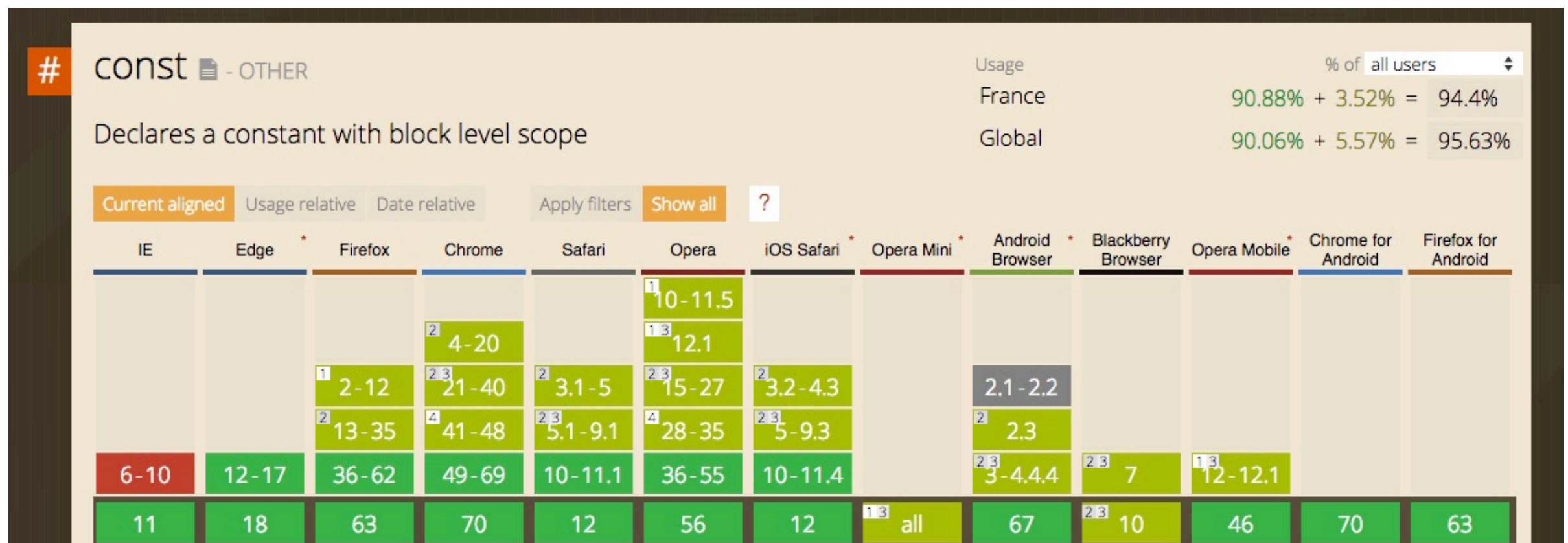
Compatibilité des navigateurs 🔗

Update compatibility data on GitHub

Desktop						Mobile					Other	
Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	Webview Android	Chrome pour Android	Firefox pour Android	Opera pour Android	Safari sur iOS	Samsung Internet	Node.js
for...of	38	12	13 *	Non	25	8	38	Oui	14 *	25	8	Oui

# LES DIFFÉRENTES MANIÈRES DE DÉCLARER UNE VARIABLE EN JS

- ▶ Jusque là, on a utilisé que le mot-clé d'origine "**var**" pour déclarer des variables
- ▶ ES6 (aussi appelé ES2015) a standardisé 2 nouvelles façons de déclarer des variables : **let** & **const**
- ▶ Le mot-clé "**const**" permet de définir des **constantes** et est plutôt bien géré par la plupart des navigateurs



## LES CONSTANTES EN JS : UN CONTENU IMMuable

- ▶ En JS, une fois une constante définie (grâce au mot-clé "const"), **le contenu de la variable n'est plus modifiable**

```
// Initialisation d'une constante numérique  
const NOMBRE = 42;  
try { NOMBRE = 99; } catch(err) {  
    console.log(err); // "TypeError: Assignment to constant variable"  
}
```

- ▶ Cependant, si la variable contient **une référence vers un objet, cet objet est modifiable !**

```
// Initialisation d'une constante objet  
const TAB = [ "un", "deux" ];  
TAB.push( "trois" );  
console.log(TAB); // [ "un", "deux", "trois" ]
```

## LES CONSTANTES EN JS : UNE PORTÉE LIMITÉE AU BLOC

- ▶ **La portée de const est celle du bloc** (Pour rappel, un bloc en JS, c'est ce qu'on retrouve **entre 2 accolades**)
- ▶ Pour rendre une constante **globale**, il faut simplement la **définir hors de toute fonction**
- ▶ Et à la différence des variables définies avec "**var**", les constantes déclarées au niveau global **ne sont pas des propriétés de l'objet global** ("window" dans un navigateur)

```
function testConst(){  
    // Bloc de la fonction  
    if(2 === 2){  
        // Bloc du if  
        var val = 12  
        const VAL = 21;  
        console.log(val); // "12"  
        console.log(VAL); // "21"  
    }  
    console.log(val); // "12"  
    console.log(VAL); // ERREUR  
}
```

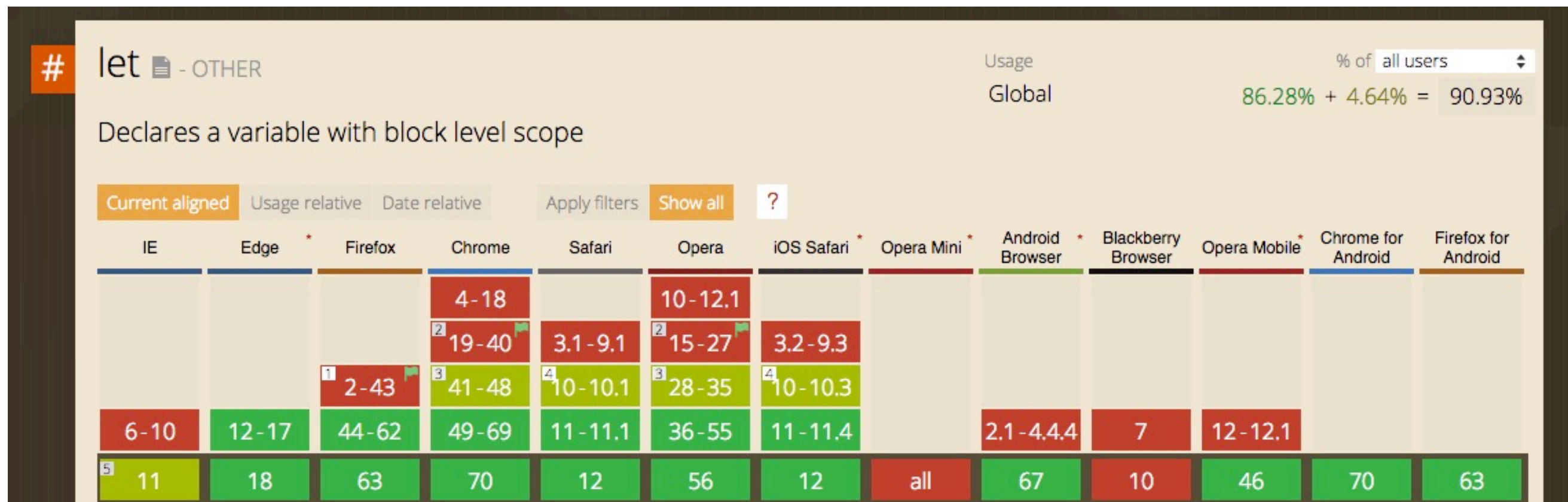
## LES VARIABLES AVEC "LET"

- ▶ Pour rappel, l'instruction "**var**" permet de déclarer une variable dont la portée est celle **de la fonction courante**
- ▶ L'instruction "**let**" permet de déclarer une variable dont la portée est celle **du bloc courant**
- ▶ Cette portée de niveau bloc est **équivalente à la portée classique des variables en C#, Java, ...**

```
function compareVarLet() {  
  var vVar = 1; let vLet = 2;  
  if(true){  
    var vVar; //(Même scope que vVar => Ne fait rien)  
    let vLet; //(Différent scope => Nouveau vLet)  
    vVar = 11;  
    vLet = 22;  
    console.log(vVar, vLet); // 11 22  
  }  
  console.log(vVar, vLet); // 11 2  
}
```



## MAIS MALHEUREUSEMENT...



- ▶ Même si le nouveau mot-clé "**let**" est globalement bien pris en charge, **il reste des irréductibles navigateurs qui ne le gèrent pas...**
- ▶ Du coup, si vous faites du JS **côté Front** et que vous voulez une **compatibilité maximum**, préférez le mot-clé "**var**"

## CONST VS. LET VS. VAR : LE TABLEAU RÉCAPITULATIF

keyword	const	let	var
global scope	NO	NO	YES
function scope	YES	YES	YES
block scope	YES	YES	NO
can be reassigned	NO	YES	YES

- ▶ "**const**" introduit la notion de **constante** en JS
- ▶ La grande différence entre "**var**" et "**let**", c'est **la portée de la variable** (le "scope")

## L'UTILISATION D'UNE BIBLIOTHÈQUE JS POUR GÉRER LA COMPATIBILITÉ



- ▶ Les bibliothèques ("library") JS étant pensées pour être multi-navigateurs, **elles s'adaptent à un grand nombre de navigateurs (et de versions) différent(e)s !**
- ▶ Même si les différences entre les moteurs JS des navigateurs tendent à diminuer, **l'utilisation d'une bibliothèque peut vous simplifier la vie**

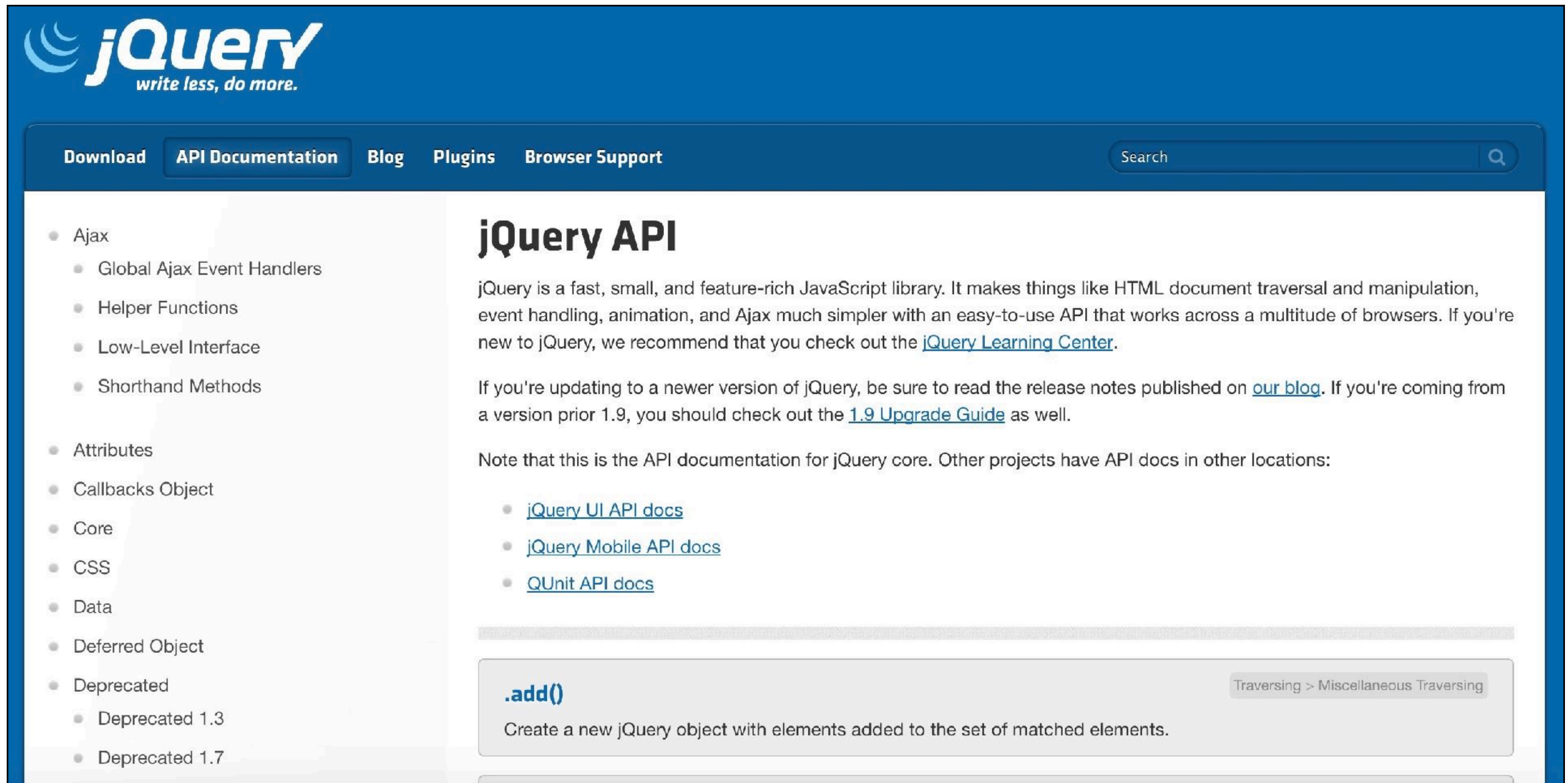
# jQuery, c'est quoi ?



- ▶ **jQuery** est une **bibliothèque JavaScript** inventée par John RESIG en 2006
- ▶ jQuery a été conçu pour **simplifier l'écriture de scripts JS** et d'aider à **gérer la compatibilité** entre les différents navigateurs Web

```
// Ajoute la classe CSS 'element' à chaque élément de liste
var listItems = document.querySelectorAll('li');
for (var i = 0 ; i < listItems.length ; i++) {
    listItems[i].className += ' element';
}
// Fait la même chose en une seule ligne grâce à jQuery
$('li').addClass('element');
```

# LA DOCUMENTATION JQUERY : VOTRE BIBLE



The screenshot shows the jQuery API documentation website. The header features the jQuery logo with the tagline "write less, do more." and a navigation bar with links for "Download", "API Documentation" (which is highlighted), "Blog", "Plugins", and "Browser Support". A search bar is located on the right side of the navigation bar. The main content area is titled "jQuery API" and contains a paragraph describing jQuery as a fast, small, and feature-rich JavaScript library. It also mentions the "jQuery Learning Center" and provides links to "our blog" and the "1.9 Upgrade Guide". A note indicates that this is the API documentation for jQuery core, with links to "jQuery UI API docs", "jQuery Mobile API docs", and "QUnit API docs". On the left side, there is a sidebar with a list of categories: Ajax, Attributes, Callbacks Object, Core, CSS, Data, Deferred Object, Deprecated, and deprecated 1.3, deprecated 1.7. The bottom section shows a specific API entry for ".add()" with a description: "Create a new jQuery object with elements added to the set of matched elements." and a breadcrumb trail: "Traversing > Miscellaneous Traversing".

**jQuery API**

jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. If you're new to jQuery, we recommend that you check out the [jQuery Learning Center](#).

If you're updating to a newer version of jQuery, be sure to read the release notes published on [our blog](#). If you're coming from a version prior 1.9, you should check out the [1.9 Upgrade Guide](#) as well.

Note that this is the API documentation for jQuery core. Other projects have API docs in other locations:

- [jQuery UI API docs](#)
- [jQuery Mobile API docs](#)
- [QUnit API docs](#)

**.add()** Traversing > Miscellaneous Traversing

Create a new jQuery object with elements added to the set of matched elements.

- La [documentation jQuery](#) est **complète** et contient pleins d'exemples => **Allez la voir** dès que vous avez une question !

## AJOUTER JQUERY À VOTRE SITE WEB

- ▶ Il existe **2 façons** d'ajouter jQuery à votre site web :
  1. [Aller sur le site de jQuery](#), télécharger la dernière version de jQuery en version **développement** ("uncompressed") ou **production** ("min(ified)"), puis intégrer simplement ce script JS à votre site web :

```
<script src="folder_name/jquery-3.2.1.min.js"></script>
```

2. Ou [récupérer le lien statique](#) de la dernière version de jQuery et l'inclure dans votre site web (Aussi appelé **"inclusion via CDN"** ([Content Delivery Network](#)) ) :

```
<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>
```

([Depuis la version 3.0 de jQuery](#), il existe aussi une version **"slim"** n'intégrant pas les fonctions AJAX, ni les "modules d'effets")



## FACILEMENT SÉLECTIONNER DES ÉLÉMENTS HTML AVEC JQUERY

- ▶ Une des forces du jQuery, c'est de pouvoir **facilement sélectionner des éléments HTML** en utilisant un **sélecteur CSS** (comme avec **querySelector()**)

Selector	Example	Example description
<u><a href="#">.class</a></u>	.intro	Selects all elements with class="intro"
<u><a href="#">#id</a></u>	#firstname	Selects the element with id="firstname"
<u><a href="#">*</a></u> <u><a href="#">-</a></u>	*	Selects all elements
<u><a href="#">element</a></u>	p	Selects all <p> elements

- ▶ La **syntaxe jQuery** à respecter étant :

```
$ ( "monSelecteurCSS" ) .méthodejQuery (paramètres) ;
```

## SÉLECTIONNER DES ÉLÉMENTS HTML AVEC JQUERY : EXEMPLES (1)

- ▶ Sélectionner tous les **<li>** au sein d'un **<ol>** en utilisant la syntaxe : **\$("ancêtre descendant")**

```
$("ol li")
```

```
<ol>
  <li>Premier élément</li> <!-- Sélectionné -->
  <li>Second élément</li> <!-- Sélectionné -->
  <ul>
    <li>Élément sans ordre</li> <!-- Sélectionné -->
  </ul>
</ol>
```

- ▶ Sélectionner tous les **<li>** qui descendent directement d'un **<ol>** en utilisant la syntaxe : **\$("parent > enfant")**

```
$("ol > li")
```

```
<ol>
  <li>Premier élément</li> <!-- Sélectionné -->
  <li>Second élément</li> <!-- Sélectionné -->
  <ul>
    <li>Élément sans ordre</li>
  </ul>
</ol>
```



## SÉLECTIONNER DES ÉLÉMENTS HTML AVEC JQUERY : EXEMPLES (2)

- ▶ Sélectionner de multiples éléments en combinant des sélecteurs CSS avec la syntaxe : `$("sel1, sel2, ..., selN")`

```
$("div, p.laClasse")
```

```
<div>div</div> <!-- Sélectionné -->  
<p class="laClasse">1er paragraphe</p> <!-- Sélectionné -->  
<p class="pasLaClasse">2nd paragraphe</p>  
<span class="laClasse">span avec la classe</span>
```

- ▶ Sélectionner des éléments qui ont un attribut en particulier avec la syntaxe : `$("selecteur[attribut]")`

```
$("a[onclick]")
```

```
<a href="#">Lien qui mène nul part</a>  
<a href="http://www.qwant.com">Lien vers Qwant</a>  
<a href="#" onclick="alert('Yo !');">Affiche "Yo !" <!-- Sélec. -->  
<button onclick="alert('Yeah !');">Affiche "Yeah !"</button>
```

- ▶ Et il existe encore pleins d'autres **sélecteurs** et **filtres** ! (cf. [la doc](#))

## L'UTILISATION DES MÉTHODES JQUERY

- ▶ Une fois qu'on a récupéré des éléments HTML sous la forme d'un **objet jQuery** avec `$("...")`, on peut alors appliquer à ces objets l'ensemble des **méthodes jQuery** qui permettent une multitudes d'actions et de transformations
- ▶ Au final, pour utiliser une méthode jQuery, il suffit d'utiliser la syntaxe : `$("...").nomDeLaMethode(param1, param2, ...)`

```
<a href="http://www.qwant.com/">Qwant</a>  
<a href="http://www.google.com">Google</a>  
<a href="http://www.bing.com">Bing</a>
```

```
// Ajoute du texte avant chaque lien  
$( "a" ).before( "Lien vers " );  
// et un retour à la ligne après chaque lien  
$( "a" ).after( "<br>" );
```

Qwant Google Bing



Lien vers Qwant  
Lien vers Google  
Lien vers Bing

## LA PUISSANCE DES MÉTHODES JQUERY

- ▶ La plupart des méthodes jQuery **s'appliquent directement à tous les éléments de l'objet jQuery** sans avoir à faire une boucle (comme **addClass()**, **before()**, **after()** ou **css()**)

```
// Passe tous les paragraphes en rouge  
$( 'p' ).css( 'color', 'red' );
```

- ▶ La plupart des méthodes jQuery **retournant un objet jQuery** sur lesquels elles se sont exécutées, on peut donc **combinaisonner les méthodes jQuery** en une seule instruction :

```
// Ajoute du texte avant chaque lien  
// et un retour à la ligne après chaque lien  
$( 'a' ).before( "Lien vers " ).after( "<br>" );
```

Qwant Google Bing



Lien vers Qwant  
Lien vers Google  
Lien vers Bing

## LA GESTION DES ÉVÈNEMENTS AVEC JQUERY

- Pour associer des actions à des événements sur des objets jQuery, il suffit d'utiliser la méthode **on()** à la place de **addEventListener()**

```
// Affiche un message dès qu'on clique sur un des  
// paragraphes de la page  
$( 'p' ).on( 'click', function () {  
    alert( "Quelqu'un a cliqué sur un paragraphe !" );  
});
```

- jQuery propose aussi des **méthodes spécifiques** pour chaque événement comme on peut le voir dans cet exemple :

```
// Fait exactement la même chose qu'au-dessus  
$( 'p' ).click( function () {  
    alert( "Quelqu'un a cliqué sur un paragraphe !" );  
});
```

## QUELQUES DÉTAILS SUR LA SYNTAXE JQUERY

```
$(sélecteur).méthode(paramètres);
```

- ▶ Pour utiliser **une méthode jQuery**, il faut commencer par récupérer **un objet jQuery** (contenant une collection d'éléments) en utilisant **la fonction jQuery**, qui a comme alias le symbole "\$" :

```
// Les 2 instructions suivantes sont équivalentes  
jQuery("monSélecteur")...  
$("monSélecteur")...
```

- ▶ **/!\** La fonction jQuery retournant un **objet jQuery**, il faut bien penser à utiliser sur cet objet **une méthode jQuery** :

```
$("#h1").innerHTML = "Avec propriété innerHTML"; // Ne fera rien...  
$("#h2").html("Avec méthode html()"); // Fonctionnera bien !
```

- ▶ On peut aussi utiliser **la fonction jQuery (\$(...))** pour transformer un objet JavaScript natif ("Vanilla JS") en **un objet jQuery** :

```
// Action quand on clique sur un élément avec la classe "btn"  
$(".btn").click(function(){ $(this).text('Cliqué'); });
```

## PARENTHÈSE : ATTENDRE LA DISPONIBILITÉ DU DOM

- ▶ Rappel : Les actions effectuées depuis le JS sur des éléments HTML ne peuvent fonctionner **que si le DOM a été entièrement défini** ! (c'est-à-dire que le navigateur a chargé toute la page web)
- ▶ C'est pour cette raison qu'en TP, on vous demandait de définir l'attribut "**onload**" de votre élément **<body>** :

```
<body onload="init();">
```

- ▶ En jQuery, il existe la méthode **ready()** qui répond aussi à ce besoin en exécutant une fonction une fois que le DOM est défini :

```
// La définition d'une fonction qui s'exécutera  
// au moment où le DOM a fini de se charger :  
$(document).ready(function() { /* ... */ });  
  
// Une écriture raccourcie qui est équivalente au ready() :  
$(function() { /* ... */ });
```

## LES REQUÊTES AJAX EN JQUERY : \$.AJAX()

- ▶ Grâce à jQuery, on peut facilement effectuer de requêtes AJAX via la méthode **\$.ajax()** :

```
<body>
  <button id="action">Lancer la requête AJAX</button>

  <script>
    $(function() {
      $('#action').click(function() {
        $.ajax({
          type: 'GET',
          url: 'mon_script.php?param=42',
          timeout: 3000,
          success: function(data) { alert(data); },
          error: function() { alert("ERREUR !"); }
        });
      });
    });
  </script>
</body>
```

(Par défaut,  
l'appel AJAX est  
**asynchrone**)



## LES REQUÊTES AJAX EN JQUERY : \$.GET() ET \$.POST()

- Il existe aussi des méthodes spécifiques pour les méthodes de requêtes **GET** et **POST** :

```
// Méthode GET : $.get(url, callback)
$( "button" ).click( function() {
    $.get( "demo_test.asp?param=2", function( data, status ) {
        alert( "Data: " + data + "\nStatus: " + status );
    });
});

// Méthode POST : $.post(url, data, callback)
$( "button" ).click( function() {
    $.post( "demo_test_post.asp",
        { name: "Donald Duck", city: "Duckburg" },
        function( data, status ) {
            alert( "Data: " + data + "\nStatus: " + status );
        });
});
```



# JQUERY UI : EXEMPLE DU "DATEPICKER"

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>jQuery UI Datepicker</title>
```

```
  <!-- Fichiers CSS et JS à inclure pour avoir accès à jQuery et jQuery UI (avec
  le fichier de traduction en Français) -->
```

```
  <link rel="stylesheet" href="https://code.jquery.com/ui/1.12.1/themes/base/
  jquery-ui.css">
  <script src="https://code.jquery.com/jquery-1.12.4.js"></script>
  <script src="https://code.jquery.com/ui/1.12.1/jquery-ui.js"></script>
  <script src="https://jqueryui.com/resources/demos/datepicker/i18n/datepicker-
  fr.js"></script>
```

```
</head>
```

```
<body>
```

```
  <p>Date: <input type="text" id="datepicker"></p>
```

```
  <script>
```

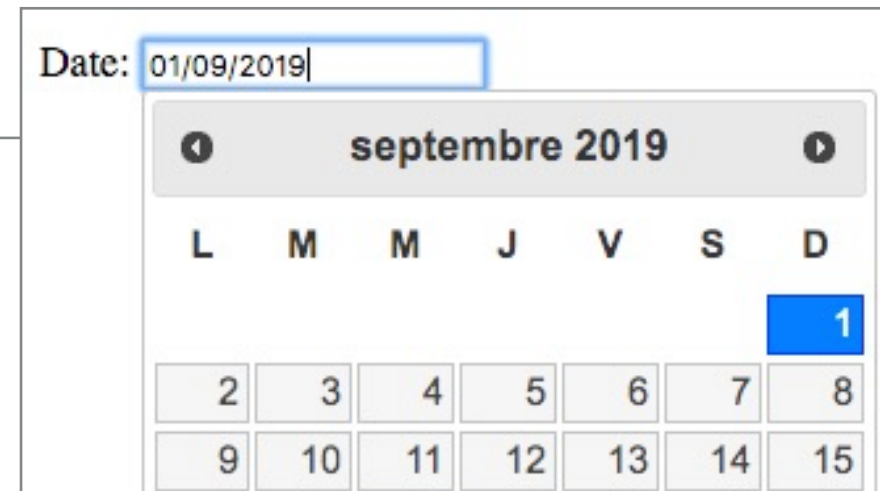
```
    // Au chargement de la page, on met en place le "Date Picker"
```

```
    $(function(){
      $( "#datepicker" ).datepicker( $.datepicker.regional[ "fr" ] ); });
```

```
  </script>
```

```
</body>
```

```
</html>
```



(Inspiré de l'exemple présent dans la doc de [jQuery UI](#))