

# WEB 1 : Initiation à la programmation WEB





Dev Web / Frontend

# Dev Web / Backend : Plan



JavaScript ?



BOM / DOM



Hello World !



AJAX



Le Langage ☹



ES6 vs JQuery



# Dev Web / Frontend : JavaScript ?

# Dev Web / Frontend : JavaScript ?



JavaWat?



Le langage a été créé en dix jours par **Brendan Eich** en mai **1995** pour le compte de la **Netscape** Communications Corporation (*future Mozilla Foundation*) qui souhaite intégrer un langage client à son Navigateur – poussé par la concurrence de **Microsoft** et **Internet Explorer**. (*futur JScript*) **LiveScript** deviendra **JavaScript** juste avant sa sortie.

Né de l'idée que l'on peut générer du dynamisme sans avoir à recharger une page côté serveur. Que l'on peut « **en demander plus** » au client

**Aucun lien réel avec Java !**

*Préfix Java n'est là qu'à des fins marketing (+ rapprochement Netscape / Sun)*

Existe depuis plus récemment en langage **Backend** (Serveur) → **NodeJS**

« **JavaScript** » devient une marque déposée par Sun Microsystems aux États-Unis en mai 1997. Netscape soumet alors JavaScript à Ecma International pour standardisation.

# Dev Web / Frontend : JavaScript ?



## Version 1 : Naissance du standard ECMA-262 - 1997 :

Naissance du standard ECMA-262 qui spécifie le langage **ECMAScript**

## Version 2 : Homogénéisation avec le standard ISO/CEI 16262 - 1998

## Version 3 : Améliorations et constitution du langage – 1999 :

Expressions rationnelles plus puissantes, amélioration de la manipulation des strings, nouvelles instructions de contrôle, gestion des exceptions avec les instructions try/catch, formatage des nombres.

Adoption massive par tous les navigateurs Web.

## Version 4 : Inachevée

## Version 5 : Désambiguïsation et nouvelles fonctionnalités – 2009 :

La 5e édition du standard ECMA-262 clarifie les ambiguïtés de la 3e édition et introduit les accesseurs, l'introspection (réflexivité), le contrôle des attributs, des fonctions de manipulation de tableaux supplémentaires, le support du format JSON et un mode strict pour la vérification des erreurs.

# Dev Web / Frontend : JavaScript ?



## Version 6 : Amélioration du support et des fonctionnalités - 2015 :

Cette édition introduit notamment les **modules**, les **classes**, la **portée au niveau des blocs**, les **itérateurs** et les **générateurs**, les **promesses pour la programmation asynchrone**, de nouvelles **structures de données** (*tableaux associatifs, ensembles, tableaux binaires*), le support de caractères **Unicode** supplémentaires dans les chaînes de caractères et les expressions rationnelles et la possibilité **d'étendre les structures de données prédéfinies**.

## De la version 7 à nos jours : une adaptation permanente aux outils du web : 2016

La 7e édition du standard ECMA-262 est la première édition issue du nouveau processus de **développement ouvert** et **du rythme de publication annuel** adoptés par le comité Ecma. Un document au format texte est créé à partir de la 6e édition et est mis en ligne sur **GitHub** comme base de développement pour cette nouvelle édition. Après la correction de milliers de bugs et d'erreurs rédactionnelles ainsi que l'introduction de l'opérateur d'exponentiation et d'une nouvelle méthode pour les prototypes de tableaux, la 7e édition est publiée en juin 2016.

# Dev Web / Frontend : JavaScript ?

## PROGRAMMING LANGUAGE ID card

Name  
JavaScript



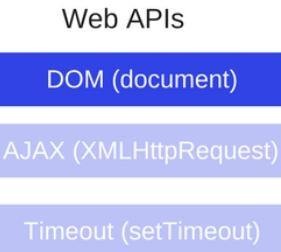
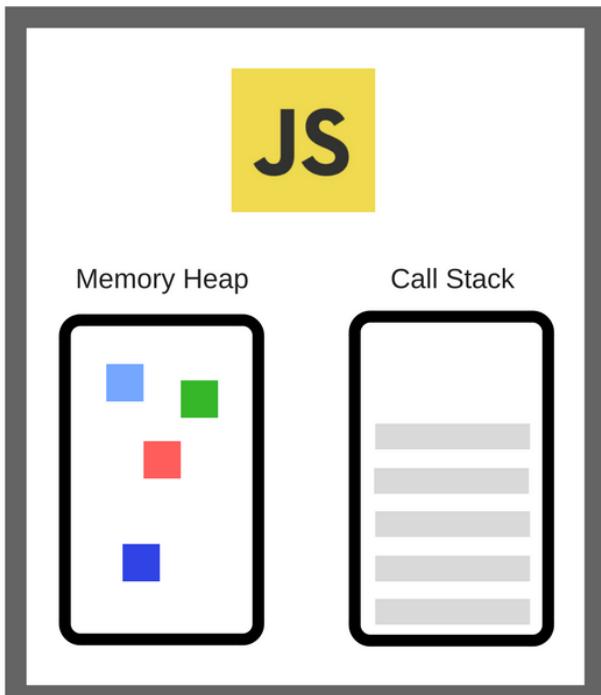
Type  
Interpreted

Platform  
Multi-platform

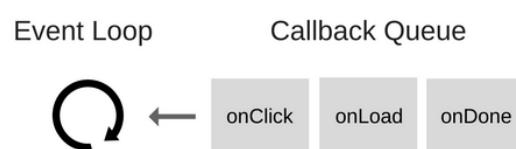
Birthdate  
1995

Licence  
Standard (ECMA)

Langage « Interprété » (*Non compilé - Exécuté instruction par instruction*), JavaScript est un langage de programmation orienté objet (à prototype) principalement employé dans les pages web interactives et à ce titre est une partie essentielle des applications web.



JS permet d'agir **dynamiquement** sur le **contenu HTML/CSS** des **pages web** en interagissant avec le **DOM** (Document Object Model) et les **actions utilisateurs** (boucle événementielle).



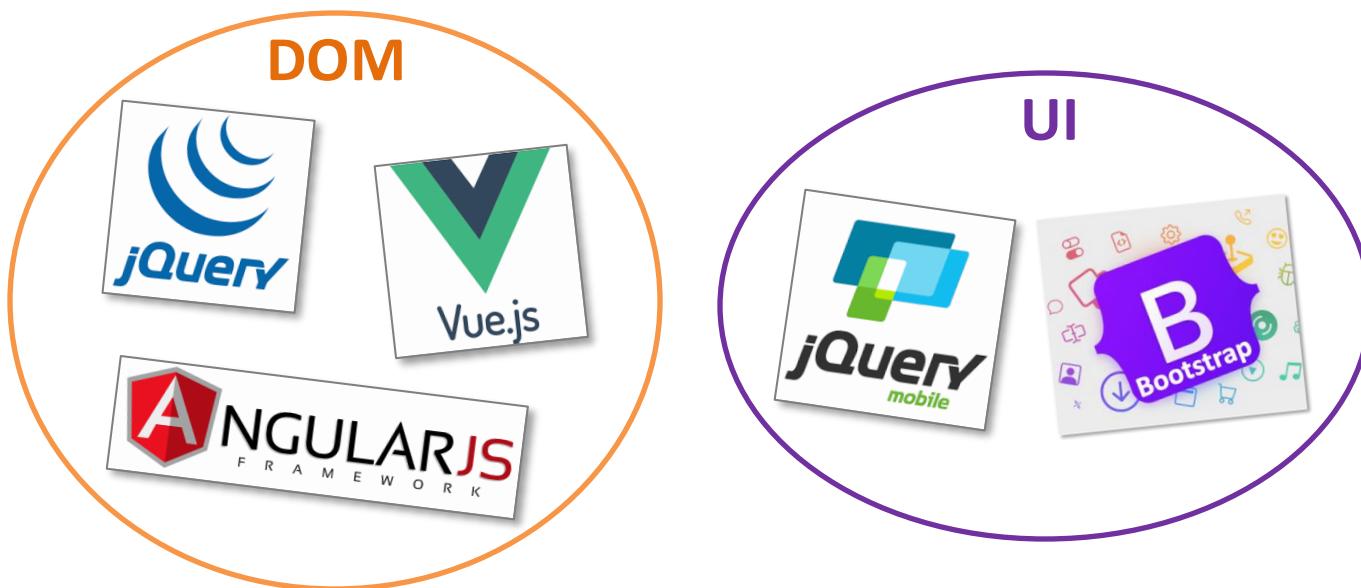
Il s'intègre directement dans la « syntaxe » HTML. Il faut donc générer le code JS au même titre que le contenu HTML/CSS depuis le serveur.

# Dev Web / Frontend : JavaScript ?



JavaScript est le langage possédant le plus large écosystème grâce à son gestionnaire de dépendances npm, avec environ 500 000 paquets en août 2017

De grands **Frameworks** de développement Frontend sont disponibles en JS :



# Dev Web / Frontend : JavaScript ?



[https://developer.mozilla.org/  
fr/docs/Web/JavaScript](https://developer.mozilla.org/fr/docs/Web/JavaScript)



# Hello, World!



OK

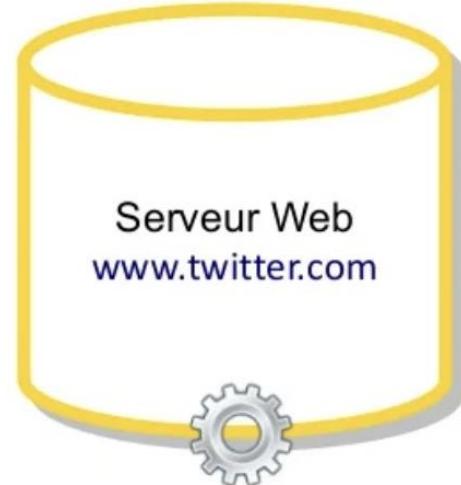
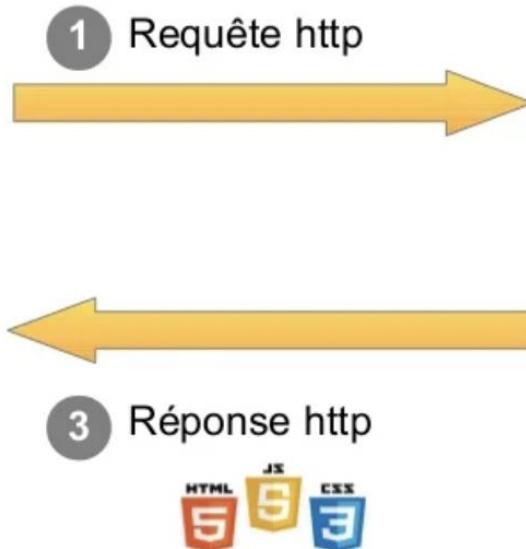
Dev Web / Frontend : JS : « Hello World »

# Dev Web / Frontend : JS : « Hello World »

Client Web : Chrome



Execution sur le client  
(HTML, CSS, **JavaScript**, ...)



Execution sur le serveur  
(ASP, PHP, JSP, ...)



Langage inclus dans tous les navigateurs actuels

Aucune installation nécessaire (sauf paquets additionnels)

Utilisable en console et dans le contenu HTML (cf. ci-après)

# Dev Web / Frontend : JS : « Hello World »

Où écrire le code JS ?

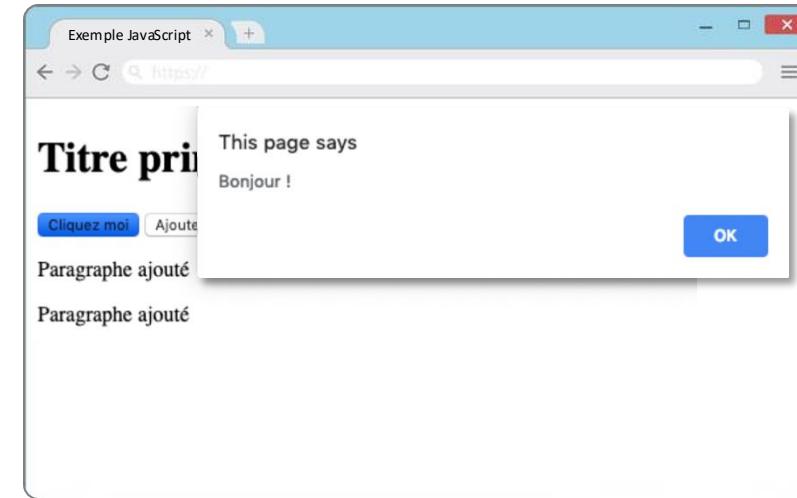
Directement au niveau des balises ouvrante HTML → Evènements :

```
<!DOCTYPE html>
<html>

<head>
    <title>Exemple JavaScript</title>
    <meta charset="utf-8">
    <link rel="stylesheet" href="cours.css">
</head>

<body>
    <h1>Titre principal</h1>
    <button onclick="alert('Bonjour!')">Cliquez moi</button>
    <button onclick="(
        function(){
            let para = document.createElement('p');
            para.textContent = 'Paragraphe ajouté';
            document.body.appendChild(para);
        }
    );">Ajouter un paragraphe
    </button>
</body>

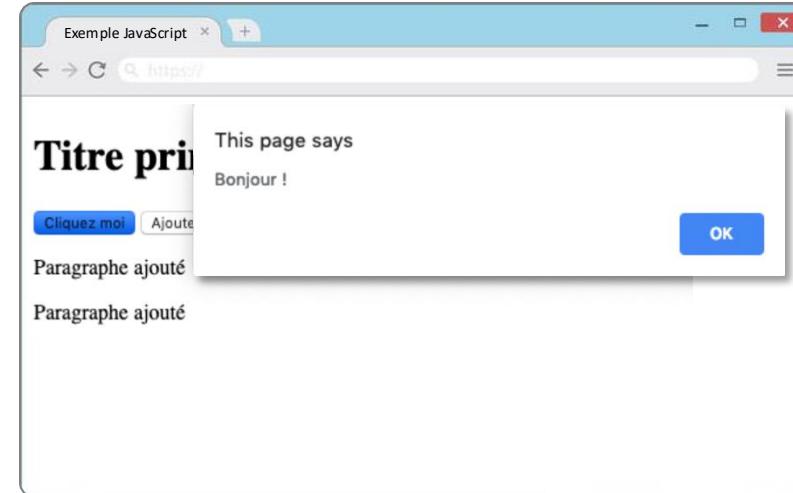
</html>
```



# Dev Web / Frontend : JS : « Hello World »

Entre les balises <script></script>

```
<!DOCTYPE html>
<html>
  <head>
    <title>Exemple JavaScript</title>
    <link rel="stylesheet" href="cours.css">
    <script>
      document.addEventListener('DOMContentLoaded',function(){
        let bonjour = document.getElementById('b1');
        bonjour.addEventListener('click',alerte);
        function alerte(){
          alert('Bonjour');
        }
      });
    </script>
  </head>
  <body>
    <h1>Titre principal</h1>
    <button id='b1'>Cliquez moi</button>
    <button id='b2'>Ajouter un paragraphe</button>
    <script>
      let ajouter = document.getElementById('b2');
      ajouter.addEventListener('click',ajout);
      function ajout(){
        let para = document.createElement('p');
        para.textContent = 'Paragraphe ajouté';
        document.body.appendChild(para);
      }
    </script>
  </body>
</html>
```



# Dev Web / Frontend : JS : « Hello World »

Dans des fichiers séparés :

cours.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Exemple JavaScript</title>
    <meta charset="utf-8">
    <link rel="stylesheet" href="cours.css">
    <script src='cours.js'></script>
  </head>

  <body>
    <h1>Titre principal</h1>
    <button id='b1'>Cliquez moi</button>
    <button id='b2'>Ajouter un paragraphe</button>
  </body>

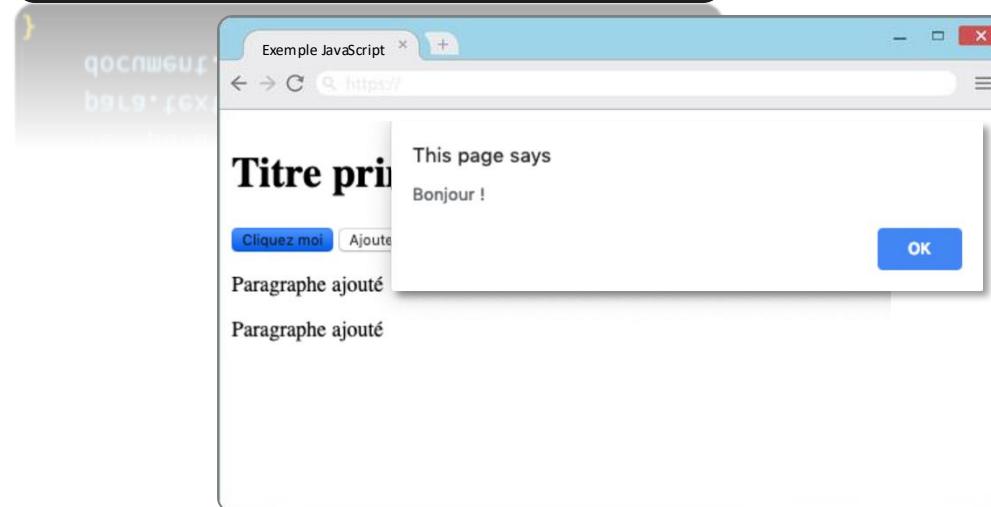
</html>
<\utf8>
<\body>
  <button id='b1'>Cliquez moi</button>
  <button id='b2'>Ajouter un paragraphe</button>
```

cours.js

```
let bonjour = document.getElementById('b1');
let ajouter = document.getElementById('b2');
bonjour.addEventListener('click', alerte);
ajouter.addEventListener('click', ajout) ;

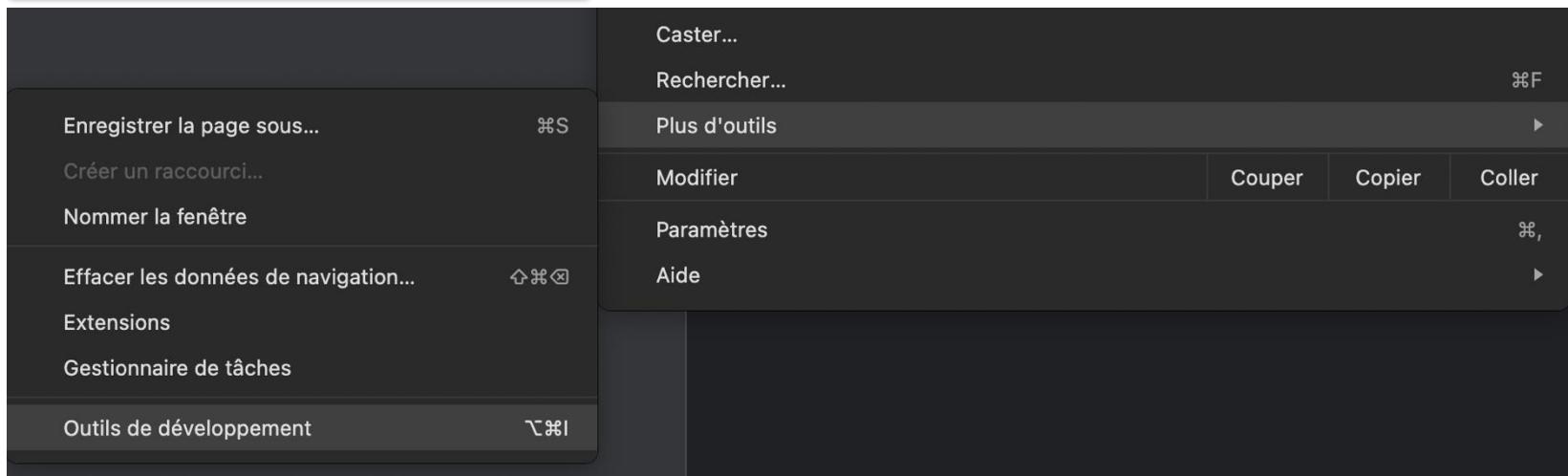
function alerte(){
  alert("Bonjour");
}

function ajout(){
  let para = document.createElement("p");
  para.textContent = "Paragraphe ajouté";
  document.body.appendChild(para);
}
```



# Dev Web / Frontend : JS : « Hello World »

## Console Développeur



A screenshot of the Chrome DevTools Console tab. A notification at the top says 'i DevTools is now available in French!' with buttons to 'Always match Chrome's language', 'Switch DevTools to French' (which is selected), and 'Don't show again'. The console interface includes tabs for Elements, Console (selected), Sources, Network, and a message count of '1'. Below the tabs is a toolbar with icons for back, forward, search, and filter. The main area shows the following output:

```
> ajout()
<- undefined
> console.log(document.getElementById('b1'));
<button id="b1">Cliquez moi</button>
```

The status bar at the bottom right shows 'VM140:1'.

```
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```



JavaScript ça rime avec ‘Fête du Slip’



```
⋮ Console What's New
🚫 top
▼ Filter
> 2 + 2
< 4
> "2" + "2"
< "22"
> 2 + 2 - 2
< 2
> "2" + "2" - "2"
< 20
```

```
> typeof NaN
< "number"
> 9999999999999999
< 1000000000000000
> 0.5+0.1==0.6
< true
> 0.1+0.2==0.3
< false
> Math.max()
< -Infinity
> Math.min()
< Infinity
> []+[]
< ""
> []+{}
< "[object Object]"
> {}+[]
< 0
> true+true+true==3
< true
> true-true
< 0
```



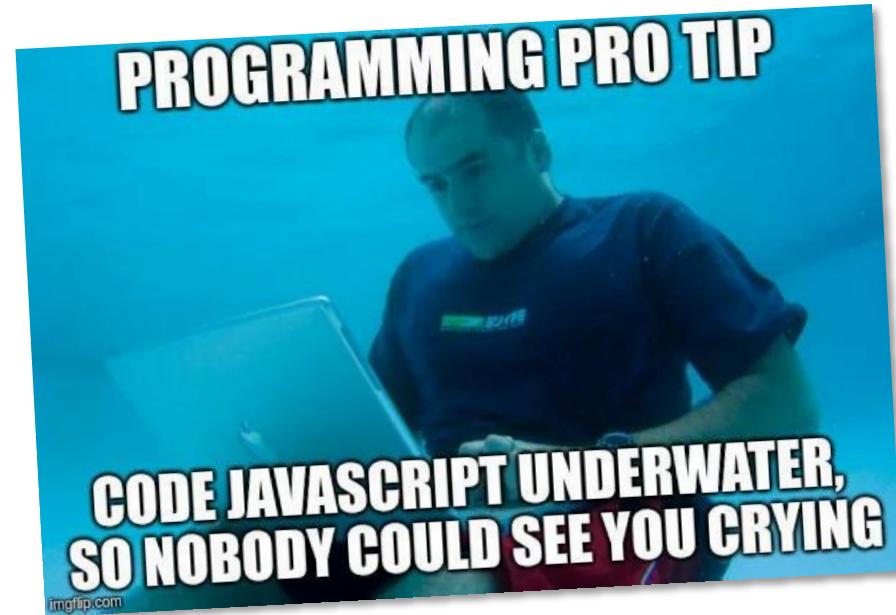
Thanks for inventing Javascript

Dev Web / Frontend : JS : Le langage ☹

## Généralités / Syntaxe



Oui bon ben JS,  
en gros, c'est du  
Java mais où on  
peut faire ce  
qu'on veut ...



- Proche du Java sauf pour la définition des fonctions et des objets ou classes
- Complètement permissif – on peut mélanger des choux avec des carottes 😊 et il n'y a aucune règle ... (même le ; est optionnel)
- Langage Objet à Prototype
- Utilisé partout

## //Les commentaires

- *Simple ligne* : `// Texte`
- *Multi-ligne* : `/* Texte */`

```
<script>  
  console.log("Ceci est un test"); // Ceci est un commentaire sur une ligne  
                                /* Ceci est un commentaire sur  
                                 plusieurs lignes */  
</script>
```



## Les Variables :

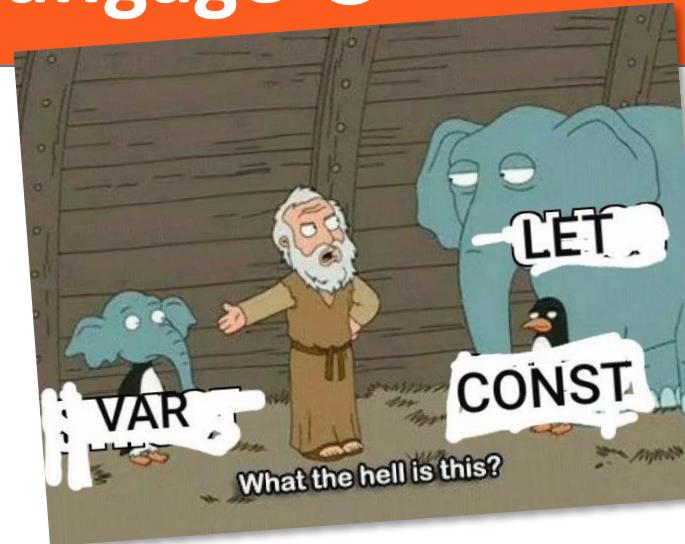
3 manières de déclarer des variables :

- **var** : Méthode historique. Portée au niveau de la fonction (ou de l'espace global) où elle est déclarée → **déconseillée** : **comportement bizarre... + deprecated avec ES6**
- **let** : Déclaration standard - **Portée "normale"**
- **const** : Déclaration de constantes de type primitif ou une variable qui *ne sera pas réallouée ultérieurement.*

### Déclaration simple

```
let strVar = "Aire du cercle";
const pi = Math.PI; //3.14159
let radius = 42;

// affichage:
console.log(strVar + " : " + (pi * radius^2));
```



### null et undefined

- **null** : possède une seule valeur, null, qui signifie l'absence de données.
- **undefined** : contenu non défini car elle n'a jamais stocké de valeur, pas même null

```
let a;
console.log(a); // undefined

let b = null;
console.log(b); // null
```

# Dev Web / Frontend : JS : Le langage 😞

## Typage :

- JS **ne nécessite pas de déclaration explicite** du type d'une variable.
- Le type d'une variable est déterminé par **le contexte d'utilisation**.
- le mot clé **typeof** permet de connaître le type d'une variable.

## Types de base :

Booléens	<pre>let vrai = true; let faux = false;</pre>
Number	<pre>let x = 10;    //x stocke un entier positif let y = -2;    //y stocke un entier négatif let z = 3.14;  //z stocke un nombre décimal positif</pre>
Strings ( <i>objet</i> )	<pre>let prenom = "Je m'appelle Pierre"; let age = 29; let age2 = "29";  console.log(typeof prenom); // string console.log(typeof age); // number console.log(typeof age2); // string</pre>

# Dev Web / Frontend : JS : Le langage 😞

## Les objets :

Possibilité de créer ses propres objets :

### Création Directe

```
let p1 = new Object();
p1.firstname = 'John';
p1.age = 42;
p1.toString = function() {
    return
        (this.firstname + ' a ' + this.age + ' ans');
}

// affichage:
console.log(p1.toString()); // John a 42 ans
```

### Avec un constructeur

```
function Person(_firstname, _age){
    this.firstname = _firstname;
    this.age = _age;
    this.toString = function() {
        return
            (this.firstname + ' a ' + this.age + ' ans');
    }
}

let p1 = new Person("John", 42);
console.log(p1.toString()); // John a 42 ans
```

A défaut de Classes comme en Java, on va utiliser les Prototypes pour « simuler » de l'héritage

# Dev Web / Frontend : JS : Le langage 😞

## Les objets :

A défaut de Classes comme en Java, on va utiliser les Prototypes pour « simuler » de l'héritage

### Prototype Child

```
function Child(_firstname, _age){  
    this.firstname = _firstname;  
    this.age = _age;  
  
    this.pleurer = function() {  
        console.log("Je pleure");  
    }  
    this.manger = function() {  
        console.log("Je mange");  
    }  
}  
  
Child.prototype = new Person();  
// ou Object.create(Person.prototype)
```

```
let c1 = new Child("Justin", 3);  
  
console.log(c1.toString()); // Justin a 3 ans  
c1.bleurer(); // Je pleure  
  
delete c1;
```



# Dev Web / Frontend : JS : Le langage 😞

## Les objets :

Depuis EcmaScript 6 : Ajout de la notions de Classes (*sucré syntaxique sur Prototype*)

### Class Person

```
class Person {
  constructor(_firstname, _age) {
    this.firstname = _firstname;
    this.age = _age
  }

  toString() {
    return
      (this.firstname + ' a ' + this.age + ' ans');
  }
}
```

### Class Child

```
class Child extends Person {
  constructor (_firstname, _age) {
    super (_firstname, _age);
  }

  pleurer() {
    console.log("Je pleure");
  }
  manger() {
    console.log("Je mange");
  }
}
```

```
let c1 = new Child("Justin", 3);

console.log(c1.toString()); // Justin a 3 ans
c1.pleurer(); // Je pleure

delete c1;
```

# Dev Web / Frontend : JS : Le langage 😞

## Les objets littéraux :

Il est possible de créer des objets (avec seulement des attributs) par une définition littérale :

### Exemple

```
const vide = {};
let point = {x:0, y:0};
let etudiant = {
  'nom': "John Doe",
  'naissance': "04/05/2000",
  'formation': {
    'intitule': "DUT Info",
    'annee': "2022-2023",
    'groupe': "XYZ"
  }
}

let nom = etudiant["nom"]
```

# Dev Web / Frontend : JS : Le langage 😞

## Les objets littéraux :

Il est possible de créer des objets (avec seulement des attributs) par une définition littérale :

### Exemple

```
const vide = {};
let point = {x:0, y:0};
let etudiant = {
  'nom': "John Doe",
  'naissance': "04/05/2000",
  'formation': {
    'intitule': "DUT Info",
    'annee': "2022-2023",
    'groupe': "XYZ"
  }
}

let nom = etudiant["nom"]
```



Ca vous rappelle qq chose ?

→ {JSON}

# Dev Web / Frontend : JS : Le langage 😊

## L'objet Strings :

Guillemets ou Cotes : (comme en Java)

```
let str1 = 'chaine contenant un \' '
let str2 = "chaine contenant un ' "
```

Accéder à un caractère :

```
let letter = str.charAt(2);
```

La concaténation : « + »

```
let str = "Bonjour" + 'toi' ;
```

Longueur d'une chaîne :

```
let size = str.length ;
```

Quelques autres méthodes :

- **concat( )** : Concatène 2 chaînes.
- **indexOf( )** : permet de trouver l'indice d'occurrence d'un caractère.
- **lastIndexOf( )** : permet de trouver le dernier indice d'occurrence d'un caractère
- **toLowerCase( )** : permet de passer toute la chaîne en minuscule
- **toUpperCase( )** : permet de passer toute la chaîne en majuscules

# Dev Web / Frontend : JS : Le langage 😊

## L'objet Strings : Quelques fonctions utiles :

- **isFinite** : Détermine si le paramètre est un nombre fini. Renvoie *false* si ce n'est pas un nombre ou l'infini positif ou infini négatif

```
isFinite(240) //retourne true
```

```
isFinite("Un nombre") //retourne false
```

- **parseFloat** : analyse une chaîne de caractères et retourne un nombre décimal.

```
parseFloat("125"); //retourne le nombre 125
```

- **Number** : convertit l'objet spécifié en valeur numérique

```
let jour = new Date("December 17, 1995");
Number(jour); //retourne le nombre de milisecs
```

- **Escape** : retourne la valeur hexadécimale d'une chaîne de caractère

- **isNaN** : Détermine si le paramètre n'est pas un nombre

```
isNaN(42) //retourne false
```

```
isNaN("Un nombre") //retourne true
```

- **parseInt** : analyse une chaîne de caractères et retourne un nombre entier.

```
parseInt("30.5", 10); //retourne 30 (base 10)
```

- **String** : analyse une chaîne de caractères et retourne un nombre entier.

```
let jour = new Date("430054663215");
String(jour); //retourne le nombre en date str
```

```
escape("!&"); //retourne %21%26%
```

# Dev Web / Frontend : JS : Le langage 😞

## L'objet Array :

### Creation et utilisation

```
let emptyArr = []; let emptyArr2 = new Array();  
let fruits = ["Apple", "Orange", "Plum"];  
alert( fruits[0] ); // Apple  
alert( fruits[1] ); // Orange  
alert( fruits[2] ); // Plum
```



### Un attribut length aussi :

- **length** : Retourne la taille du tableau

### Quelques autres méthodes :

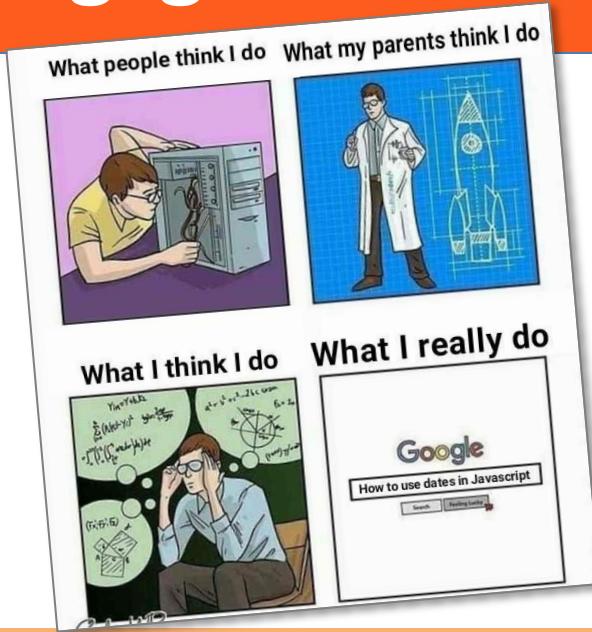
- **concat( )** : permet de concaténer 2 tableaux
- **join( )** : convertit un tableau en chaîne de caractères
- **reverse( )** : inverse le classement des éléments du tableau
- **slice( )** : retourne une section du tableau
- **sort( )** : permet le classement des éléments du tableau

# Dev Web / Frontend : JS : Le langage 😞

## L'objet Date :

### Creation et utilisation

```
let date1 = new Date(); let date1b = Date.now();
let date2 = new Date('March 23, 2019 20:00:00');
let date3 = new Date(1553466000000);
let date4 = new Date(2019, 0, 25, 12, 30);
```



### Quelques autres méthodes :

- **getFullYear()**, **getYear()**, **getMonth()**, **getDay()**, **getDate()**, **getHours()**, **getMinutes()**, **getSeconds()**, **getMilliseconds()** : retournent respectivement l'année complète, l'année (2chiffres), le mois, le jour de la semaine, le jour du mois, l'heure, les minutes, les secondes et les millisecondes stockés dans l'objet Date
- **getTime()** : retourne le temps stocké dans l'objet Date
- **getTimezoneOffset()** : retourne la différence entre l'heure du client et le temps universel
- **toGMTString()**, **toLocaleString()**, **toUTCString()** : convertissent la date en chaîne de caractère selon la convention GMT, selon la convention locale ou en temps universel

# Dev Web / Frontend : JS : Le langage 😊

## L'objet Math :

### Les attributs :

- **LN2** : renvoie le logarithme népérien de 2 (~0.693)
- **LN10** : renvoie le logarithme népérien de 10 (~2.302)
- **LOG2E** : renvoie le logarithme en base 2 de e (~1.442)
- **LOG10E** : renvoie le logarithme en base 10 de e (~0.434)
- **PI** : renvoie la valeur du nombre pi (~3.14159)

### Quelques autres méthodes :

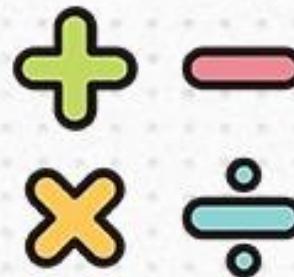
- **abs()**, **exp()**, **log()**, **sin()**, **cos()**, **tan()**, **asin()**, **acos()**, **atan()**, **max()**, **min()**, **sqrt()** : opérations mathématiques habituelles
- **atan2( )** : retourne la valeur radian de l'angle entre l'axe des abscisses et un point
- **ceil( )** : retourne le plus petit entier supérieur à un nombre
- **floor( )** : retourne le plus grand entier inférieur à un nombre
- **pow( )** : retourne le résultat d'un nombre mis à une certaine puissance
- **random( )** : retourne un nombre aléatoire entre 0 et 1
- **round( )** : arrondi un nombre à l'entier le plus proche.

# Dev Web / Frontend : JS : Le langage 😞

Les objets du navigateur :

Cf. BOM / DOM

## Les Opérateurs :



`++, --` Incrémentation, décrémentation

`!` Négation

`+, -, *, /, %` Arithmétique

`+` Concaténation

`<, <=, >, >=, ==, !=` Comparaison

`==, !=` Comparaison Forte

`&&` ET logique

`||` OU logique

## Les structures de contrôle



Comme en JAVA quoi ...

### if

```
if (conditional test){  
    // do this;  
}  
else if{  
    // do this;  
}  
else{  
    // do this;  
}
```

### switch

```
switch (expr){  
    case possible result #1 :  
        // do this;  
        break;  
    case possible result #2 :  
        // do this;  
        break;  
    default :  
        // do this;  
}
```

## Les structures de contrôle



Comme en JAVA quoi ...

### while

```
while (condition is true){  
    // do this;  
}
```

### do while

```
do {  
    // do this;  
} while (condition is true);
```

### for

```
for (init counter ; conditionel test ; update counter){  
    // do this;  
}
```

### for .. in ...

```
for (let p in obj1){  
    // k → propriétés  
    // obj1[k] → valeur  
}
```

### for .. of ...

```
let arr = [ ... ];  
for (let v in arr){  
    // v → valeur  
}
```

## Les structures de contrôle

### foreach

```
array.forEach((element) => {  
    // process element  
});
```

```
Object.keys(obj).forEach(key => {  
    // key → propriété  
    // obj[key] → valeur  
});
```

```
Object.values(obj).forEach(value => {  
    // value → Valeur  
});
```

```
Object.entries(obj).forEach(entry => {  
    const [key, value] = entry;  
    // key → propriété / value → valeur  
});
```

## Les fonctions :

```
function carre(i){  
    return i*i;  
}
```

JavaScript permet d'assigner des fonctions à des variables :

```
let hello = function(who){  
    console.log("Salut "+ who);  
}  
  
hello("toi"); //Salut toi
```

Notion de clôture : Fonctions imbriquées



## Gestions des erreurs

```
try {  
  // do it  
}  
catch(err){  
  // handle error  
}  
finally{  
  // exec everytime  
}
```





Dev Web / Frontend : JS : **BOM / DOM**

# Dev Web / Frontend : JS : **BOM / DOM**

Les objets BOM / DOM sont les instances des éléments du navigateur et du document qu'il est possible de manipuler avec JS

**Objets de type *BOM*** : Permet de manipuler le navigateur

- **Window, Navigator, Screen, History, Location**

**Objets de type *DOM*** : Permet de manipuler le document HTML

- **Document, Elements, Attributes, Events, ...**

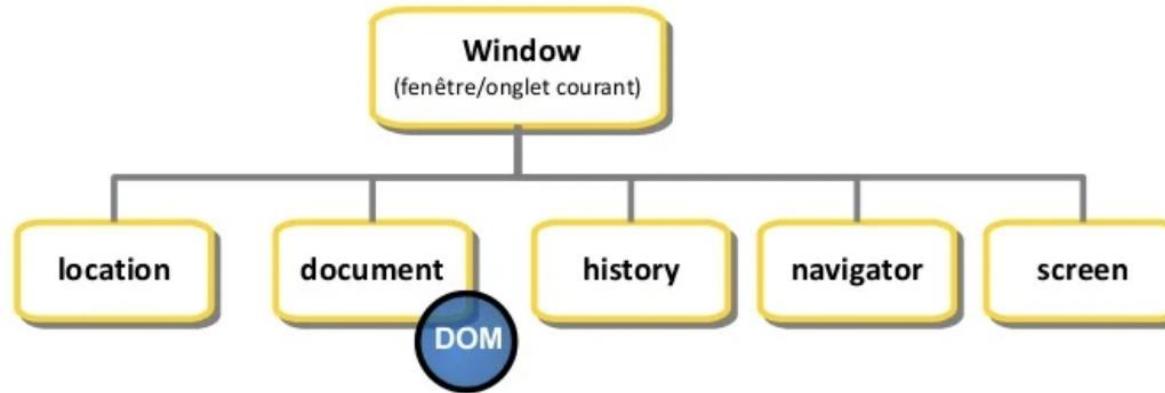
**Objets de type *HTML***

- **<a>, <area>, <canvas>, <button>, <form>, <image>, <input>, <link>, <option>, <select>, <style>, <table>, <td>, <th>, <tr>, <textarea>, ...**

# Dev Web / Frontend : JS : **BOM / DOM**

## Objets de type **BOM** : Permet de manipuler le navigateur

Ouvrir / fermer des onglets, charger une nouvelle URL, afficher des messages, ...



Chaque objet possède ses propriétés et méthodes

Entrées :

```
window.prompt("Quel nombre ?");  
window.confirm("Etes vous sûr ?");
```

Sortie :

```
window.alert("Message !");
```

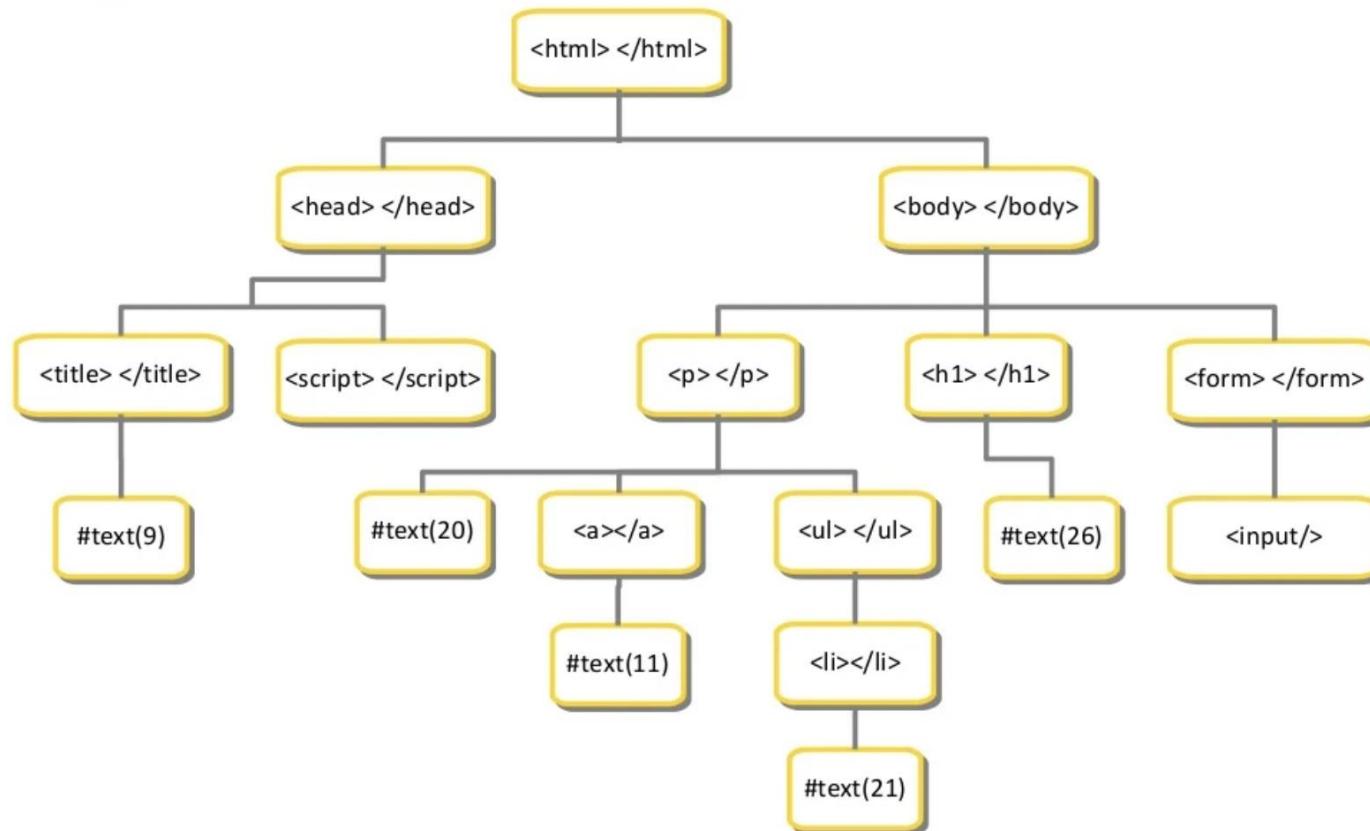
Autres exemples :

```
window.location = "https://www.google.fr";  
window.open("https://youtube.fr");
```

# Dev Web / Frontend : JS : **BOM / DOM**

## Objets de type **DOM** : Permet de manipuler le document HTML

Représentation sous forme arborescente de tous les objets du document html



# Dev Web / Frontend : JS : **BOM / DOM**

## Objets de type ***DOM*** : Permet de manipuler le document HTML

Représentation sous forme arborescente de tous les objets du document html

En HTML chaque balise est considérée comme un noeud et peut posséder des attributs

```
<a href='test.html'> Page d'accueil</a>
```

En présence de plusieurs nœuds du même type, je peux attribuer un identifiant unique pour y accéder après

```
<div id='par1'>ceci est le paragraphe 1</div>
<div id='par2'>ceci est le paragraphe 2</div>
<div id='par3'>ceci est le paragraphe 2</div>
```

# Dev Web / Frontend : JS : **BOM / DOM**

## Objets de type **DOM** : Permet de manipuler le document HTML

Représentation sous forme arborescente de tous les objets du document html

A partir de Document je peux accéder aux éléments à l'aide des méthodes

- **getElementById('valeur\_id')** : accéder à un élément via son Id  
`document.getElementById('par1');`
- **getElementsByName('nom\_element')** : accéder à tous les éléments  
`document.getElementsByName('div');`
- **getElementsByClassName('nom\_classe')** : accéder à tous les éléments sur la base de l'attribut class

# Dev Web / Frontend : JS : **BOM / DOM**

## Objets de type **DOM** : Permet de manipuler le document HTML

Représentation sous forme arborescente de tous les objets du document html

A partir de Document je peux accéder aux éléments à l'aide des méthodes

- Méthode **innerHTML** : accéder au contenu d'un élément :

```
document.getElementById('par1').innerHTML;
```

```
document.getElementById('par1').innerHTML="Bienvenue";
```

- Méthode **textContent()** : accéder au contenu texte d'un élément :

```
document.getElementById('par1').textContent();
```

- Modifier un attribut d'un élément :

```
document.getElementById('par1').style.color = "blue";
```

# Dev Web / Frontend : JS : **BOM / DOM**

## Objets de type **DOM** : Permet de manipuler le document HTML

Représentation sous forme arborescente de tous les objets du document html

### Les évènements :

Un évènement est une action de l'utilisateur prise en compte par le navigateur

*Exemple : Clic droit, touche enfoncee, position de la souris, etc.....*

**onEvenement="Action\_Javascript\_ou\_Fonction();"**

```
<img src='une_image.jpg' onclick='nom_de_la_fonction(parametre1,parametre2)' />
```

```
<img src='une_image.jpg' onmouseover='fonction1() ; fonction2() ; fonction3()' />
```

### Dans le DOM :

```
document.getElementById('mon_element').onclick=nom_de_la_fonction;
```

```
document.getElementById('mon_element').onclick=function(un_parametre){  
du code du code...etc}
```

```
element.onclick = function(e) {  
    // L'argument « e » va récupérer une référence vers l'objet « Event »  
    alert(e.type); // Ceci affiche le type de l'événement (click, mouseover, etc.)  
};
```

## Objets de type ***DOM*** : Permet de manipuler le document HTML

Représentation sous forme arborescente de tous les objets du document html

### L'événement **onload**

Un évènement spécial va vous permettre d'exécuter du code en vous assurant que toute la structure de la page HTML est bien chargée... ☺

```
<head>
<script>
  function initialize(){
    //some process
  }
</script>
</head>

<body onload="initialize()">
...
</body>
```

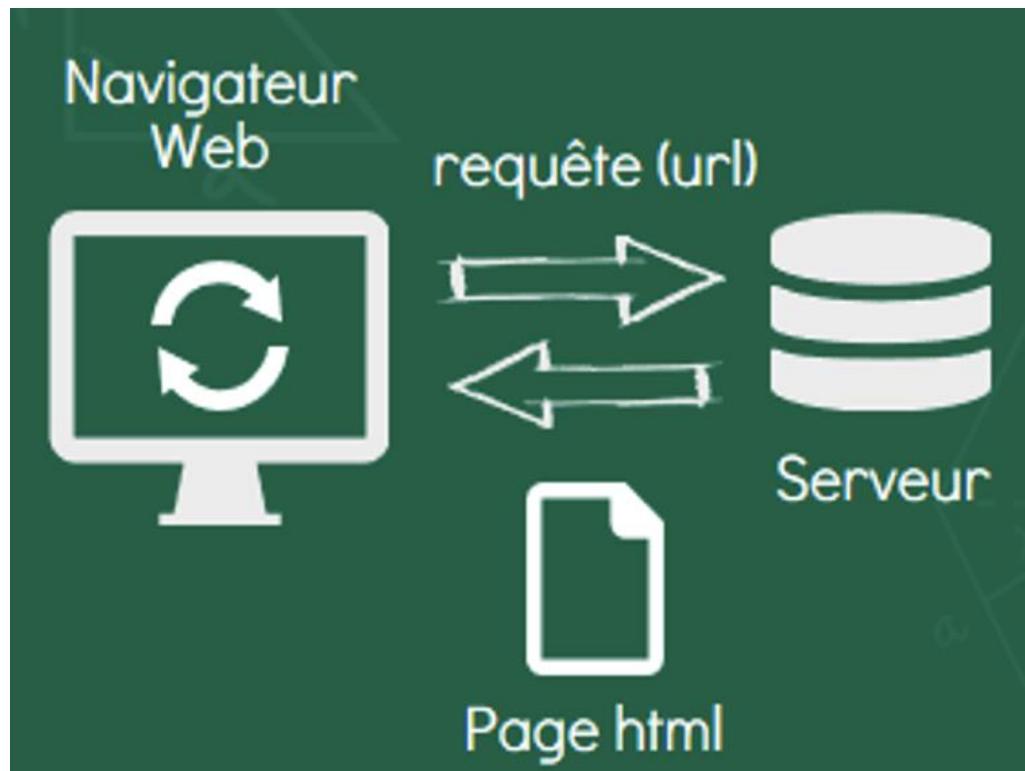


Dev Web / Frontend : JS : **AJAX**

## Avant AJAX (Web Historique)

**La moindre action de l'utilisateur (envoi ou demande de données)**

→ *Chargement d'une nouvelle page envoyée par le serveur.  
(processus inefficace, lent, et peu agréable pour l'utilisateur).*



*Cela signifie qu'une fois la page affichée à l'écran, celle-ci est « figée » : L'utilisateur ne peut pas interagir directement avec la page. Il peut en revanche cliquer sur des liens, ce qui ouvrira d'autres pages, ou remplir des formulaires. Mais ces actions vont entraîner l'envoi d'une nouvelle requête au serveur, qui répondra avec une nouvelle page web.*

# Dev Web / Frontend : JS : **AJAX**

## Arrivée d'AJAX

: Formalisé en 2005

**A**synchronous **J**avascript **A**nd **X**ML

**AJAX** permet d' envoyer et récupérer des données d'un serveur de manière **asynchrone** (en arrière-plan) sans interférer avec l'affichage et le comportement de la page existante.



Grosso-modo, l'AJAX nous permet de modifier de manière dynamique le contenu d'une page, c'est-à-dire sans qu'il soit nécessaire de recharger l'intégralité de la page.

## Arrivée d'AJAX

: Formalisé en 2005

**A**synchronous **J**avascript **A**nd **X**ML

A sa création, l'AJAX utilisait les technologies suivantes qui lui ont donné son nom :

</>  
XML pour l'échange de données avec le serveur

L'objet **XMLHttpRequest** pour la communication asynchrone

Le **JavaScript** pour afficher les données de manière dynamique et permettre à l'utilisateur d'interagir avec les nouvelles informations

Le **HTML** et le **CSS** pour la présentation des données.



# Dev Web / Frontend : JS : **AJAX**

## Arrivée d'AJAX

: Formalisé en 2005

**A**synchronous **J**avascript **A**nd **X**ML

A sa création, l'AJAX utilisait les technologies suivantes qui lui ont donné son nom :



Aujourd'hui, le XML a été largement délaissé au profit du JSON (JavaScript Object Notation) qui est une notation qui permet d'échanger des données relativement simplement tandis que l'objet XMLHttpRequest est lentement en train de laisser sa place à la nouvelle API Fetch.



**"L'AJAX"** ou plutôt **"l'Ajax"** est aujourd'hui un terme générique utilisé pour désigner toute **technique côté client** (côté navigateur) permettant **d'envoyer et de récupérer des données** depuis un serveur et de **mettre à jour dynamiquement le DOM** sans nécessiter l'actualisation complète de la page.



Requête Cross Domain : “Cross-origin resource sharing” (CORS)

# Dev Web / Frontend : JS : **AJAX**

## Exemples avec JavaScript natif :

### Requête synchrone

```
let xmlhttp = new XMLHttpRequest();
xmlhttp.open("GET", url, true);
xmlhttp.send();

window.alert(xmlhttp.responseText);
// or doing some DOM manipulation
// with this response
```



**Création de l'objet requête :**

**True : Pour autoriser les requêtes CORS**

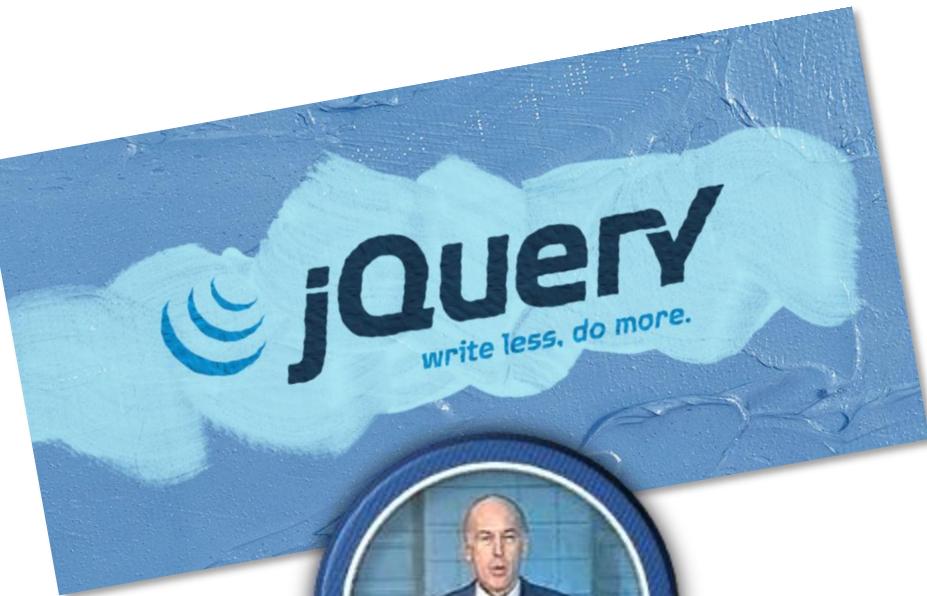
**Bloqué dans l'attente de la réponse**

### Requête Asynchrone

```
let xmlhttp = new XMLHttpRequest();
xmlhttp.open("GET", url, true);
xmlhttp.onreadystatechange = function () {
  if (xmlhttp.readyState == 4 && xmlhttp.status == 200){
    window.alert(xmlhttp.responseText);
    // or doing some DOM manipulation with this response
  }
};

xmlhttp.send();
```

**Utilisation d'un Callback**



# Les facilités d'**ES6**

# Historiquement

JavaScript était assez difficile à manipuler

Il n'était pas « cross navigateurs »

Requêtes CORS difficile à mettre en œuvre



- Ecrit par John Resig en 2006
- Open-source et cross navigateurs
- Manipulation DOM facile
- Syntaxe simplifiée
- ... notamment pour requêtes AJAX
- En 2019 : 70% des sites web sont en JQuery



## PROGRAMMING JAVASCRIPT



- Tous les navigateurs suivent la norme ECMA
- API simplifiée de manip. de DOM + AJAX
- Apparition de nombreux Frameworks JS
- JQuery devenu inutile
- De moins en moins populaire

# ES6

# Historiquement



CommitStrip.com

Tous les navigateurs suivent la norme **ECMA**

API simplifiée de manip. de DOM + AJAX

Apparition de nombreux Frameworks JS

JQuery devenu inutile

→ De moins en moins populaire

→ De moins en moins populaire

**ES6**

## Le Sélecteur Universel

`$('anything')`



`document.querySelector('anything')`

`document.querySelectorAll('anything')`

accepte un sélecteur **CSS** en argument

accepte des ID :

`$("#nomID")`



`document.querySelector("#nomID")`

`document.getElementById("nomID")`

accepte des classes (et même plusieurs):

`$(".nomClasse")`

`document.querySelector('.nomClasse')`

`$(".article, .nouvelles, .edito")`



`document.querySelectorAll('.article, .nouvelles, .edito')`

# Dev Web / Frontend : JS : ES6 vs JQuery

accepte des sélecteurs spécifiques :

```
$(":radio")  
$(":header"),  
$(":first-child")
```



```
document.querySelector("[type='radio'])  
document.querySelector("h1, h2, h3, h4, h5, h6")  
document.querySelector(":first-child")
```

\$ accepte des sélecteurs d'attributs :

```
$( "a[href]" )  
$( "a[href^='http://']" )  
$( "img[src$='.png']" )
```



```
document.querySelectorAll("a[href]")  
document.querySelectorAll("a[href^='http://']")  
document.querySelectorAll("img[src$='.png']")
```

**DOCUMENTATION IS LIKE  
SEX WHEN IT'S GOOD, IT'S VERY GOOD  
WHEN IT'S BAD, IT'S  
BETTER THAN NOTHING**

Doc Doc Doc ...

[https://developer.mozilla.org/en-US/docs/Web/API/Document\\_Object\\_Model](https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model)



<https://tobiasahlin.com/blog/move-from-jquery-to-vanilla-javascript/>

<https://aarontgrogg.com/blog/2021/09/29/replacing-jquery-with-vanilla-es6/>

<https://youmightnotneedjquery.com>

# Dev Web / Frontend : JS : ES6 vs JQuery

EXAMPLE TIME



## Exemple de manipulation du DOM

```
<html>
<script type="text/javascript" src= "...">
</script>
<body>
  <div id="monDiv">Bonjour</div>
  <a href="#" »
    onClick="document.querySelector('#monDiv').style.display='none';">
      disparition
    </a>
</body>
</html>
```

# Dev Web / Frontend : JS : ES6 vs JQuery

## Pour jouer avec les attributs CSS

Appliquer à tous les éléments sélectionnés :

```
$(".classe").hide();  
$(".classe").show();
```

```
document.querySelectorAll(".classe").foreach(  
  e => e.style.display = 'none');  
document.querySelectorAll(".classe").foreach(  
  e => e.style.display = 'block');
```

## Accéder à des sous éléments

```
let cont = $(".container");  
cont.find(".box");
```

```
let cont =  
  document.querySelector(".container");  
cont.querySelector(".box");
```

## Parcourir le DOM

```
$(".box").next();  
$(".box").prev();  
$(".box").parent();
```

```
let box = document.querySelector(".box");  
box.nextElementSibling;  
box.previousElementSibling;  
box.parentElement;
```

# Dev Web / Frontend : JS : ES6 vs JQuery

Vu que la plupart des méthodes retourne l'objet, on peut chaîner ☺

```
$( 'anything' )  
  .parent()  
  .find( 'still anything' )  
  .show();
```

```
document.querySelector( 'anything' )  
  .parentElement  
  .querySelector('still anything')  
  .style.display = 'block';
```

On peut jouer avec les attributs (checked, innerText, innerHTML) :

```
elem.attr('attributeName');
```

```
elem.getAttribute('attributeName');  
ou elem.attributeName;
```

```
elem.attr('attributeName',  
  'newValue');
```

```
elem.setAttribute('attributeName',  
  'newValue');  
ou elem.setAttributeName = 'newValue';
```

```
elem.text('Something');
```

```
elem.innerText = 'Something';
```

```
elem.html('<h1>Something</h1>');
```

```
elem.innerHTML = '<h1>Something</h1>';
```

# Dev Web / Frontend : JS : ES6 vs JQuery

... avec les styles :

```
$(".box").css({  
  "color": "#000",  
  "background-color": "red"  
});
```

```
let box =  
  document.querySelector(".box");  
  
box.style.color = "#000";  
box.style.backgroundColor = "red";  
ou  
box.style.cssText =  
  "color: #000; background-color: red";
```

```
$(".box").addClass("focus");  
$(".box").removeClass("focus");  
$(".box").toggleClass("focus");
```

```
var box =  
  document.querySelector(".box");  
box.classList.add("focus");  
box.classList.remove("focus");  
box.classList.toggle("focus");
```

# Dev Web / Frontend : JS : ES6 vs JQuery

, Créer des éléments :

```
$('#div1').append(  
  "Text : <a  
  href='#'>Lien</a>");
```



```
let element =  
  document.createElement("div");  
  
element.innerHTML =  
  "Text : <a href='#'>Lien</a>"  
  
document.querySelector("#div1")  
  .appendChild(element)
```

```
var html = '<tr>';  
  
html += '<td>' + data.one + '</td>';  
html += '<td>' + data.two + '</td>';  
html += '<td>' + data.three + '</td>';  
html += '</tr>';  
  
tbody.innerHTML += html;  
ou tbody.insertAdjacentHTML('beforeend', html);
```

# Dev Web / Frontend : JS : ES6 vs JQuery

## Et agir sur les événements

```
<html><body><div id="container">
<ul id="list">
<li><a href="http://domain1.com">Item #1</a></li>
<li><a href="/local/path/1">Item #2</a></li>
<li><a href="/local/path/2">Item #3</a></li>
</ul>
</div></body></html>
```

→ Attach a directly bound event handler

```
$( "#list a" ).on("click",
  function(event) {
    event.preventDefault();
    console.log($(this).text());
  });

```



```
document.querySelectorAll("#list a")
.forEach(a =>
  a.addEventListener("click",
    function(event) {
      event.preventDefault();
      console.log(event.target);
    })
)
```

Et si on ajoute un élément dynamiquement ?

# Dev Web / Frontend : JS : ES6 vs JQuery



Et si on ajoute un élément dynamiquement ?

```
<html><body><div id="container">
<ul id="list">
  <li><a href="http://domain1.com">Item #1</a></li>
  <li><a href="/local/path/1">Item #2</a></li>
  <li><a href="/local/path/2">Item #3</a></li>
</ul>
</div></body></html>
```

```
$("#list").append(
  "<li><a href='#test'>Item
#4</a></li>");
```

```
document.querySelector("#list").innerHTML +=
  "<li><a href='#test'>Item #4</a></li>";
```

→ Utiliser la délégation d'événements :

```
$( "#list" ).on("click", "a",
  function(event) {
    event.preventDefault();
    console.log($(this).text());
 });
```

```
document.querySelector("#list")
  .addEventListener("click",
    function(event) {
      if (event.target.matches('a')){
        event.preventDefault();
        console.log(event.target);
      }
    })
});
```

# Dev Web / Frontend : JS : ES6 vs JQuery

## Et agir sur les événements

→ "Trigger" un événement

```
$(document).trigger("myEvent");
$(".box").trigger("myEvent");
```

```
document.dispatchEvent(new Event("myEvent"));
document.querySelector(".box")
  .dispatchEvent(new Event("myEvent"));
```

→ Et interagir selon l'état de la page

```
$(document).ready(function() {
  /* Do things after
  DOM has fully loaded */
});
```

```
var ready = (callback) => {
  if (document.readyState != "loading") {
    callback();
  }
  else {
    document.addEventListener("DOMContentLoaded",
      callback);
  }
}

ready(() => {
  /* Do things after DOM
  has fully loaded */
});
```

# Dev Web / Frontend : JS : ES6 vs JQuery

EXAMPLE TIME



Intercepter le bouton submit d'un formulaire :

```
var ready = (callback) => {
  if (document.readyState != "loading") {callback();}
  else {document.addEventListener("DOMContentLoaded",callback);}
}

ready(() => {
  Let form = document.querySelector("#form1");
  form.onsubmit = function(event) {
    event.preventDefault();
    if (form.elements['login'].value == "") {
      alert('Entrer un login');
    }
    return false;
  });
});
```

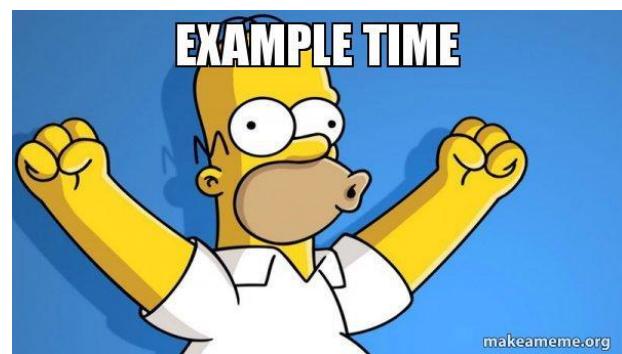
## Exemples :

Determiner si une checkbox est cochée

```
if (document.querySelector('#checkbox1').checked) {
  //Traitement si cochée
}
else {
  //Traitement si non cochée
}
```

# Dev Web / Frontend : JS : ES6 vs JQuery

EXAMPLE TIME



## Exemples :

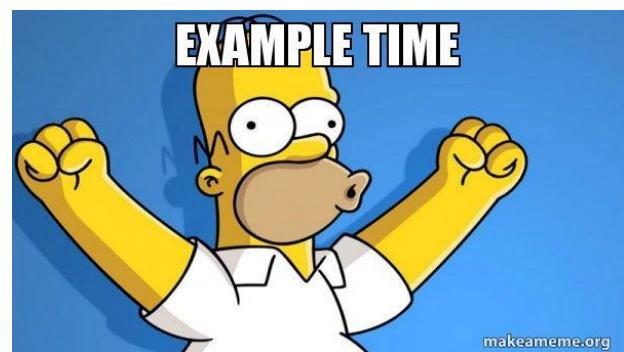
Effacer le contenu d'un champs de texte lorsqu'il a le focus

```
<input name="nom" type="text" id="nom" value="Entrez votre nom">
```

```
document.querySelector("#nom").onfocus = function(event) {  
    event.target.value = "";  
}
```

# Dev Web / Frontend : JS : ES6 vs JQuery

EXAMPLE TIME



## Exemples :

Fonction qui cache ou affiche les lignes d'un tableau

```
function toggleTableRows(display) {  
    document.querySelectorAll("#mytable1 tbody tr").forEach((tr)=>{  
        tr.style.display = display?"none';  
    });  
}
```

# Dev Web / Frontend : JS : ES6 vs JQuery

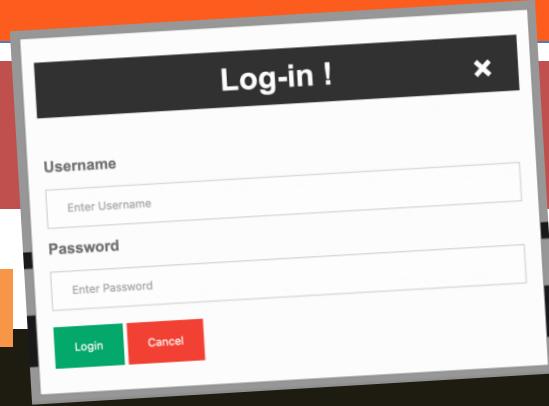
EXAMPLE TIME



## Exemples :

Afficher un dialog 'modal'

HTML / JS



```
<!-- DIV contenant le formulaire d'authentification -->
<div id="loginModal" class="modal">
<form id="loginForm" class="modal-content"
      action=<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?> method="post">
  <div class="dlgheadcontainer">
    <span onclick="document.getElementById('loginModal').style.display='none'">
      <span>&times;</span>
      <h1>Log-in !</h1>
    </div>

    <div class="dlgcontainer">
      <label for="uname"><b>Username</b></label>
      <input type="text" placeholder="Enter Username" name="login" id="login" required>

      <label for="psw"><b>Password</b></label>
      <input type="password" placeholder="Enter Password" name="password" id="password" required>
      <button type="submit" class="okbtn">Login</button>
      <button type="button" onclick="document.getElementById('loginModal').style.display='none'">
        <span>Cancel</span>
      </button>
    </div>
  </form>
</div>
<a href="#" onclick="document.getElementById('loginModal').style.display='block';">CONNECT</a>
```

# Dev Web / Frontend : JS : ES6 vs JQuery

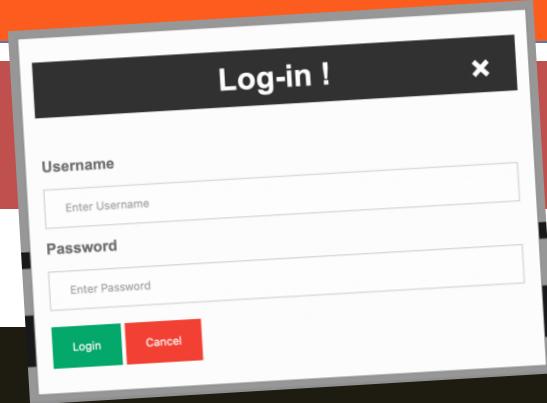
EXAMPLE TIME



## Exemples :

Afficher un dialog 'modal'

CSS



```
.modal {  
    display: none; /* Hidden by default */  
    position: fixed; /* Stay in place */  
    z-index: 200; /* Sit on top */  
  
    left: 0;  
    top: 0;  
    width: 100%; /* Full width */  
    height: 100%; /* Full height */  
    overflow: auto; /* Enable scroll if needed */  
    /* Fallback color */  
    background-color: rgb(0, 0, 0);  
    /* Black w/ opacity */  
    background-color: rgba(0, 0, 0, 0.4);  
    padding-top: 60px;  
}  
  
/* Modal Content/Box */  
.modal-content {  
    background-color: #fefefe;  
    /* 5% from the top, 15% from the bottom and centered */  
    margin: 5% auto 15% auto;  
    border: 1px solid #888;  
    width: 500px;  
}
```

```
/* The Close Button (x) */  
.close {  
    position: absolute;  
    right: 25px;  
    top: 0;  
    color: white;  
    font-size: 35px;  
    font-weight: bold;  
}
```

# Dev Web / Frontend : JS : ES6 vs JQuery



## \$.ajax VS fetch ...

```
$.ajax({  
  url: 'https://example.com',  
  method: 'post',  
  dataType: 'json',  
  contentType: 'application/json',  
  data: JSON.stringify(data),  
  success: function( response ){  
    //...  
  },  
  error: function( error ){  
    //...  
  }  
});
```

```
fetch( 'https://example.com', {  
  method: 'post',  
  headers: {  
    'Accept': 'application/json',  
    'Content-Type': 'application/json'  
  },  
  body: JSON.stringify(data)  
})  
.then( response => {  
  //...  
})  
.catch( error => {  
  //...  
});
```



Th-Th-Th-That's All Folks

Dev Web / frontend : JS : **FIN**