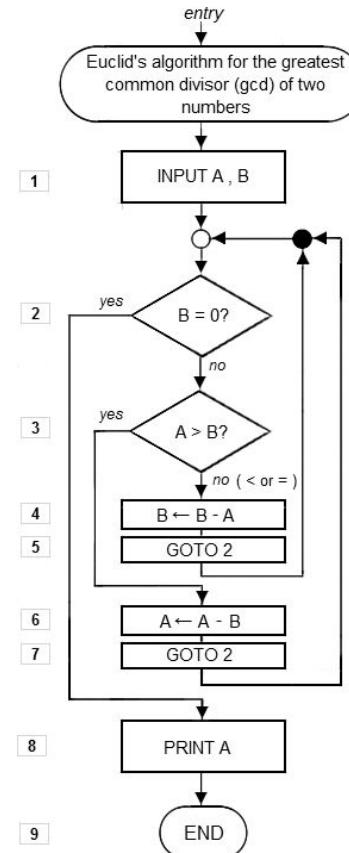


**Algorithmie,  
C'est quoi un  
algorithme ?**

**L'algorithmique** est l'étude et la production de règles et techniques qui sont impliquées dans la définition et la conception d'algorithmes, c'est-à-dire de processus systématiques de résolution d'un problème permettant de décrire précisément des étapes pour résoudre un problème algorithmique.

Source :  
<https://fr.wikipedia.org/wiki/Algorithmique>



**L'algorithme** est la capacité d'une personne à découper un problème complexe en sous parties plus faciles à comprendre afin de faire exécuter une tâche à une machine ou une personne.

Source : MOI

## **Des algorithmes que nous utilisons tous les jours :**

- La liste des courses
- Une recette de cuisine
- Suivre un itinéraire
- Les étapes pour monter un meuble
- ...

Chaque étape d'un algorithme doit être **CLAIREMENT EXPLICITE** pour ne pas faire planter le code (ou la personne) et faire en sorte que l'algorithme s'exécute correctement.

La différence entre un humain et une machine est que cette dernière n'est pas capable de comprendre un sous-entendu et va faire bêtement ce qu'il y a d'écrit. Moins on est précis, plus on laisse la machine interpréter ce qu'elle comprend, moins l'algorithme s'exécute comme on veut.



Source : <https://www.youtube.com/watch?v=FN2RM-CHkul>

INTRO  
**ALGO!**

QU'EST-CE  
QU'UN  
**ALGORITHME?**



<https://www.youtube.com/watch?v=tbmKIErjnns>

# **Le JavaScript, C'est quoi ?**

## Algorithmie - Le JavaScript, c'est quoi ?

Le JavaScript est un langage de programmation créé en 1995. Le JavaScript est aujourd’hui l’un des langages de programmation les plus populaires et il fait partie des langages web dits « standards » avec le HTML et le CSS. Son évolution est gérée par le groupe ECMA International qui se charge de publier les standards de ce langage.

- Le JavaScript est un langage dynamique ;
- Le JavaScript est un langage (principalement) côté client ;
- Le JavaScript est un langage interprété ;
- Le JavaScript est un langage orienté objet.

# Algorithmie - Le JavaScript, c'est quoi ?

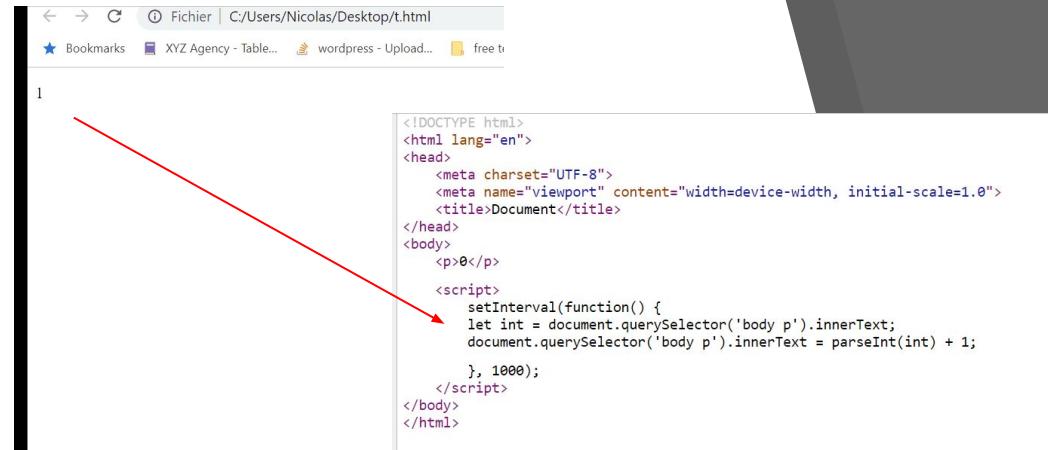
## Le JavaScript est un langage dynamique :

Le JavaScript peut modifier le contenu d'une page web (modifier, ajouter, supprimer un texte d'une page par exemple) , faire des calculs (1\*3 par exemple), aller chercher des données sur d'autres sources (via des API), structurer des données (sous forme de tableau ou objet), stocker des informations via des cookies, ....On oppose généralement les langages « dynamiques » aux langages « statiques » comme le HTML et le CSS.



```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <p>Je suis un message écrit dans le HTML de la page</p>
  <script>
    document.querySelector('body p').innerHTML = 'J\'ai le pouvoir de changer le code dynamiquement !';
  </script>
</body>
</html>
```

J'ai le pouvoir de changer le code dynamiquement !



```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <p>0</p>
  <script>
    setInterval(function() {
      let int = document.querySelector('body p').innerText;
      document.querySelector('body p').innerText = parseInt(int) + 1;
    }, 1000);
  </script>
</body>
</html>
```

# Algorithmie - Le JavaScript, c'est quoi ?

Le JavaScript est un langage (principalement) côté client :

C'est une code "front-end", c'est à dire que les navigateurs web peuvent l'interpréter.

Cependant, il peut également être comme langage serveur (nodeJS).

D'ailleurs, si vous voyez le code dans la source, c'est que c'est un code client, un code serveur est invisible depuis le navigateur.

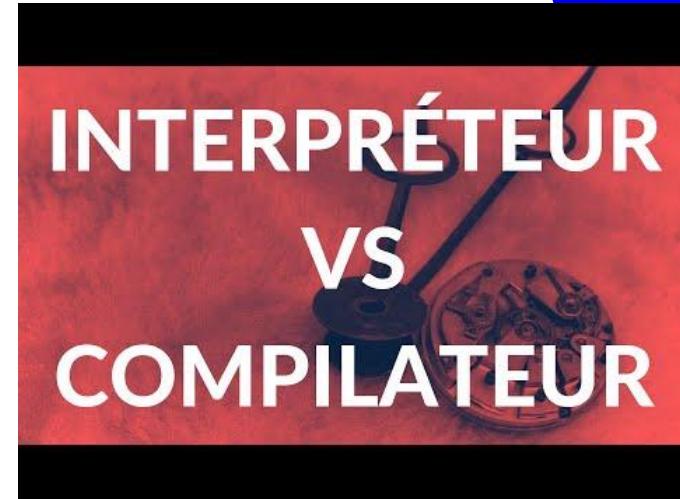
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <p>0</p>
    <script>
        setInterval(function() {
            let int = document.querySelector('body p').innerText;
            document.querySelector('body p').innerText = parseInt(int) + 1;
        }, 1000);
    </script>
</body>
</html>
```

# Algorithmie - Le JavaScript, c'est quoi ?

**Le JavaScript est un langage interprété :**

Le JavaScript est un langage interprété. Cela signifie qu'il va pouvoir être exécuté directement sous réserve qu'on possède le logiciel interpréteur. C'est à dire un navigateur web. C'est lui qui va "traduire" notre code pour lui faire faire ce que l'on veut.

L'opposé du langage interprété est le langage compilé (comme le SCSS ou le C par exemple). [https://www.youtube.com/watch?v=4IXp\\_89c3RU](https://www.youtube.com/watch?v=4IXp_89c3RU)



# Algorithmie - Le JavaScript, c'est quoi ?

Le JavaScript est un langage orienté objet :

Trop tôt pour en parler mais, pour info, l'objet du JavaScript s'appelle le JSON (JavaScript Object Notation) et qu'il sert à structurer des informations. Un peu comme les tableaux. On va dire que la différence entre un tableau et un objet est que l'objet peut stocker des fonctions. On y reviendra...

```
<script>
    var personne = {
        nom: ['Jean', 'Martin'],
        age: 32,
        sexe: 'masculin',
        interets: ['musique', 'skier'],
        bio: function() {
            alert(this.nom[0] + ' ' + this.nom[1] + ' a ' + this.age + ' ans. Il aime ' +
                this.interets[0] + ' et ' + this.interets[1] + '.');
        },
        salutation: function() {
            alert('Bonjour ! Je suis ' + this.nom[0] + '..');
        }
    };

    </script>
</body>
```

## Algorithmie - Le JavaScript, c'est quoi ?

Le JavaScript n'est pas du Java !!!

Ce sont 2 codes complètement différents !

Trop de personnes confondent ces 2 codes mais plus vous.

Si vous trouvez que JavaScript est trop long à dire, dites JS.

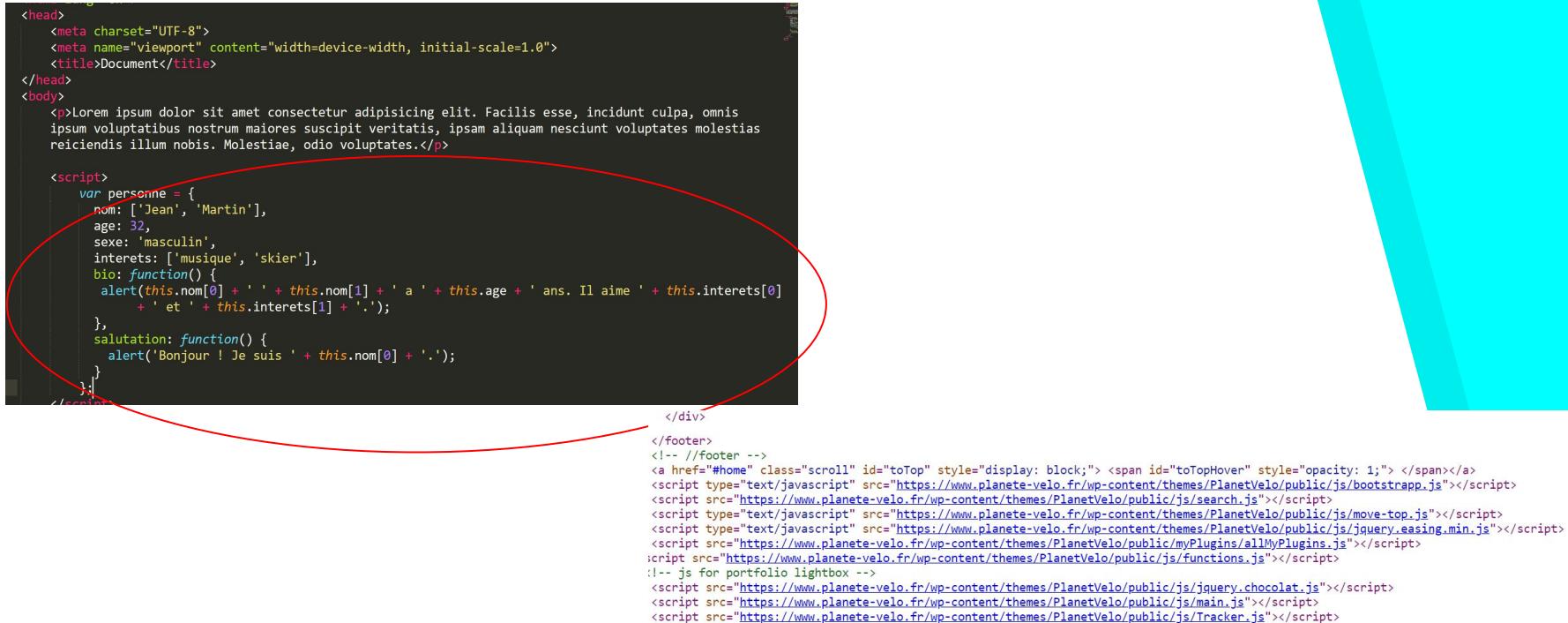




<https://www.youtube.com/watch?v=cdDPQkF7hRA>

# Algorithmie - Le JavaScript, c'est quoi ?

Le JS s'écrit soit entre 2 balises script OU directement dans un fichier avec une extension en .js (c'est plus propre)



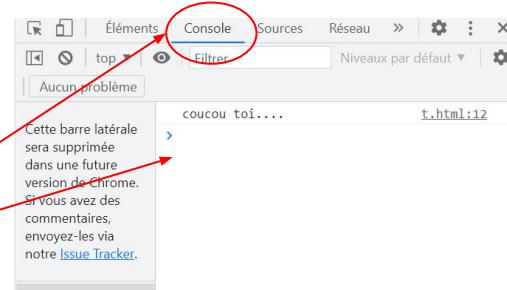
The screenshot shows a code editor displaying an HTML file. The file contains a head section with meta tags for charset and viewport, and a title. The body section contains a paragraph of placeholder text (Lorem ipsum) and a script block. The script block defines a variable 'personne' with properties: nom (an array of 'Jean' and 'Martin'), age (32), sexe ('masculin'), interets (an array of 'musique' and 'skier'), bio (a function that alerts a message combining all these properties), and salutation (a function that alerts 'Bonjour ! Je suis ' followed by the first name). A red oval highlights the entire script block. The footer section includes a 'toTop' scroll button and several external script imports.

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Facilis esse, incident culpa, omnis ipsum voluptatibus nostrum maiores suscipit veritatis, ipsam aliquam nesciunt voluptates molestias reciendis illum nobis. Molestiae, odio voluptates.</p>
    <script>
        var personne = {
            nom: ['Jean', 'Martin'],
            age: 32,
            sexe: 'masculin',
            interets: ['musique', 'skier'],
            bio: function() {
                alert(this.nom[0] + ' ' + this.nom[1] + ' a ' + this.age + ' ans. Il aime ' + this.interets[0]
                    + ' et ' + this.interets[1] + '.');
            },
            salutation: function() {
                alert('Bonjour ! Je suis ' + this.nom[0] + '.');
            }
        };
    </script>
</body>
<div>
    <!-- //header -->
    <a href="#home" class="scroll" id="toTop" style="display: block;"> <span id="toTopHover" style="opacity: 1;"></span></a>
    <script type="text/javascript" src="https://www.planete-velo.fr/wp-content/themes/PlanetVelo/public/js/bootstrap.js"></script>
    <script src="https://www.planete-velo.fr/wp-content/themes/PlanetVelo/public/js/search.js"></script>
    <script type="text/javascript" src="https://www.planete-velo.fr/wp-content/themes/PlanetVelo/public/js/move-top.js"></script>
    <script type="text/javascript" src="https://www.planete-velo.fr/wp-content/themes/PlanetVelo/public/js/jquery.easing.min.js"></script>
    <script src="https://www.planete-velo.fr/wp-content/themes/PlanetVelo/public/myPlugins/allMyPlugins.js"></script>
    <script src="https://www.planete-velo.fr/wp-content/themes/PlanetVelo/public/js/functions.js"></script>
    <!-- js for portfolio lightbox -->
    <script src="https://www.planete-velo.fr/wp-content/themes/PlanetVelo/public/js/jquery.chocolat.js"></script>
    <script src="https://www.planete-velo.fr/wp-content/themes/PlanetVelo/public/js/main.js"></script>
    <script src="https://www.planete-velo.fr/wp-content/themes/PlanetVelo/public/js/Tracker.js"></script>
```

# Algorithmie - Le JavaScript, c'est quoi ?

  Lorem ipsum dolor sit amet consectetur adipisicing elit. Facilis esse, incident culpa, omnis ipsum voluptatibus nostrum maiores suscipit veritatis, ipsam aliquam nesciunt voluptates molestias reiciendis illum nobis. Molestiae, odio voluptates.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Facilis esse, incident culpa, omnis ipsum voluptatibus nostrum maiores suscipit veritatis, ipsam aliquam nesciunt voluptates molestias reiciendis illum nobis. Molestiae, odio voluptates.</p>
    <script>
        console.log('coucou toi....')
    </script>
</body>
</html>
```



Quand on code en JS, on ouvre la console (clique droit > inspecter > onglet "console") dans le navigateur.

C'est dans cet endroit où on va pouvoir constater les éventuelles Erreurs. De base, le JS est invisible...

# **Algorithmie, Les Variables**

# Algorithmie - Les variables

Quelques définitions :

String : chaîne de caractère (une phrase en gros, entre guillemet)

Number : un chiffre/nombre entier

Boolean : true / false - 0 / 1 - ON / OFF

Var : initialisation d'une variable qui peut changer (va devenir obsolète)

Const : initialisation d'une variable qui ne peut pas changer

Let : initialisation d'une variable qui peut changer

Array : c'est un tableau (indiqué, associatif, multi-dimensionnel)

Concaténation : le fait de mettre une variable dans un string

## Algorithmie - Les variables

```
<script>
    var variable1 = 1;
    let variable_2 = 'coucou';
    const variableTrois = [];
</script>
```

Une variable est une boite servant à stocker des informations de manière temporaire, comme une chaîne de caractères (variable\_2), un nombre (variable1), un tableau (variableTrois) par exemple. Les variables sont l'une des constructions de base du JavaScript et vont être des éléments qu'on va énormément utiliser. Nous allons illustrer leur utilité par la suite.

# Algorithmie - Les variables

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.
<title>Document</title>
</head>
<body>
  <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Facilis
  ipsum voluptatibus nostrum maiores suscipit veritatis, ipsam aliquam nesciunt voluptates molestias reiciendis illum
  nobis. Molestiae, odio voluptates.</p>

  <script>
    var variable1 = 1;
    let variable_2 = 'coucou';
    const variableTrois = [];

    console.log(variable1);
  </script>
</body>
</html>
```

Et voici le résultat dans la console :

1

Cette barre latérale sera supprimée dans une future version de Chrome. Si vous avez des commentaires, envoyez-les via notre [Issue Tracker](#).

▶ 1 message

▶ 1 message ...

✖ Aucune erre...

⚠ Aucun avert

On voit bien que si je souhaite afficher la variable “variable1” dans la console,  
On voit la valeur 1, celle qu'on lui a attribué.  
console.log('...') affiche dans la console le contenu entre parenthèses.

## Algorithmie - Les variables

```
<script>
    var variable1 = 1;
    let variable_2 = 'coucou';
    const variableTrois = [];
</script>
```

- Le nom d'une variable doit obligatoirement commencer par une lettre ou un underscore (\_) et ne doit pas commencer par un chiffre ;
- Le nom d'une variable ne doit contenir que des lettres, des chiffres et des underscores mais pas de caractères spéciaux ;
  - Le nom d'une variable ne doit pas contenir d'espace.

## Algorithmie - Les variables

```
<script>
  var variable1 = 1;
  let variable_2 = 'coucou';
  const variableTrois = [];
</script>
```

Pourquoi ces petits mots devant les variables ?

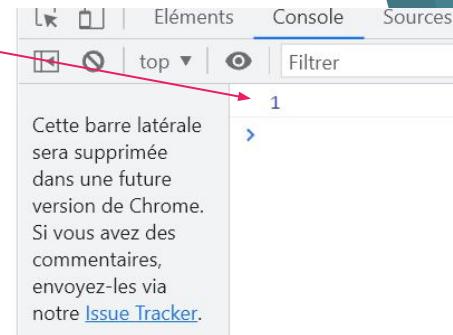
Quand une variable est utilisée pour la première fois, il faut l'initialiser. Et pour ce faire, on utilise soit **var**, **let** ou encore **const**. Cela permet aussi de mieux organiser ses infos et son code.

# Algorithmie - Les variables

```
<script>
    variable1 = 1;
    Let variable_2 = 'coucou';
    const variableTrois = [];

    console.log(variable1);
</script>
```

  Lorem ipsum dolor sit amet consectetur adipisciing  
  elit. Facilis esse, incidunt culpa, omnis ipsum  
  voluptatibus nostrum maiores suscipit veritatis, ipsam  
  aliquam nesciunt voluptates molestias reiciendis illum  
  nobis. Molestiae, odio voluptates.

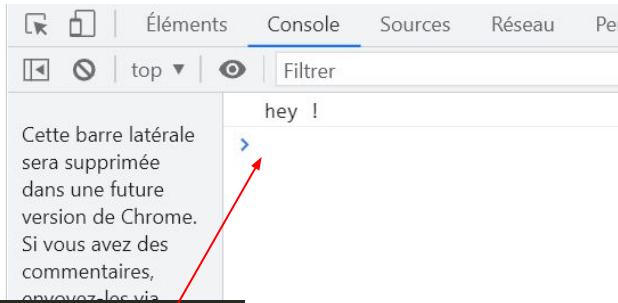


Après, on peut voir que JS fonctionne normalement si on ne l'initialise pas correctement. Mais ça se fait pas.

# Algorithmie - Les variables

Lorem ipsum dolor sit amet consectetur adipisicing elit. Facilis esse, incidunt culpa, omnis ipsum voluptatibus nostrum maiores suscipit veritatis, ipsam aliquam nesciunt voluptates molestias reiciendis illum nobis. Molestiae, odio voluptates.

```
<script>
    var variable1 = 1;
    variable1 = 'belette';
    variable1 = 'hey !';
    console.log(variable1);
</script>
```



Le code est lu de haut en bas. De ce fait, une variable peut être réécrite (pas toute) et changer de valeur. C'est la dernière valeur qui écrase les autres...

## Algorithmie - Les variables

```
<script>
    var variable1 = 1;
    let variable_2 = 'coucou';
    const variableTrois = [];
</script>
```

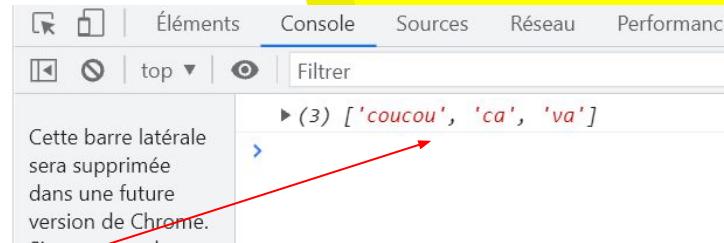
Ok, mais c'est quoi la différence entre var, let et const ????

Il faut savoir que ‘var’ est en passe d’être obsolète. Il ne faut pas l’utiliser. On peut encore la voir dans certains codes. Au début de JS, il n’y avait que ‘var’ pour initialiser les variables.

# Algorithmie - Les variables

Lorem ipsum dolor sit amet consectetur adipisicing elit. Facilis esse, incidunt culpa, omnis ipsum voluptatibus nostrum maiores suscipit veritatis, ipsam aliquam nesciunt voluptates molestias reiciendis illum nobis. Molestiae, odio voluptates.

```
<script>
    let variable1;
    variable1 = 20;
    variable1 = ['coucou', 'ca', 'va'];
    console.log(variable1);
</script>
```

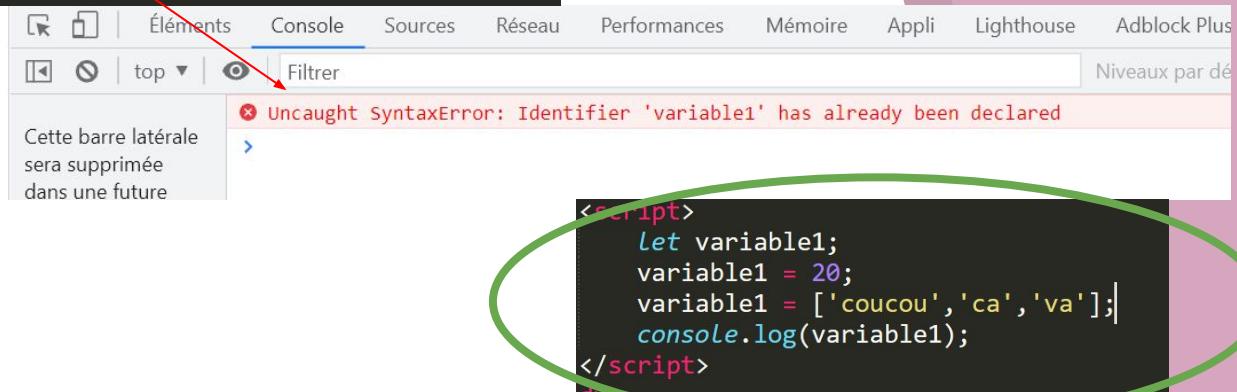


“Let” permet de pouvoir réécrire les variables à l’infini (comme “var”)

# Algorithmie - Les variables

```
<script>
    let variable1;
    let variable1 = 20;
    let variable1 = ['coucou', 'ca', 'va'];
    console.log(variable1);
</script>
```

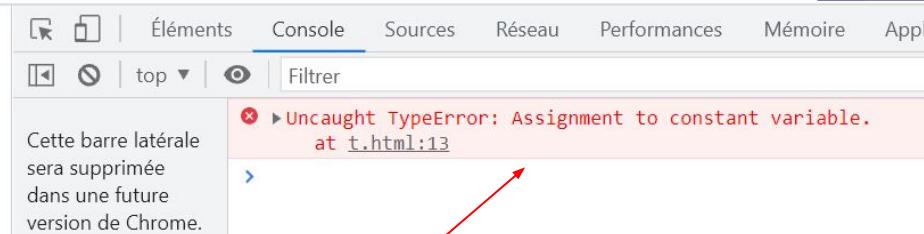
Lorem ipsum dolor sit amet consectetur adipisicing elit. Facilis esse, incidunt culpa, omnis ipsum voluptatibus nostrum maiores suscipit veritatis, ipsam aliquam nesciunt voluptates molestias reiciendis illum nobis. Molestiae, odio voluptates.



“Let” ne peut être déclarée qu’une seule fois (pas comme “var”)

# Algorithmie - Les variables

Lorem ipsum dolor sit amet consectetur adipisicing elit. Facilis esse, incidunt culpa, omnis ipsum voluptatibus nostrum maiores suscipit veritatis, ipsam aliquam nesciunt voluptates molestias reiciendis illum nobis. Molestiae, odio voluptates.

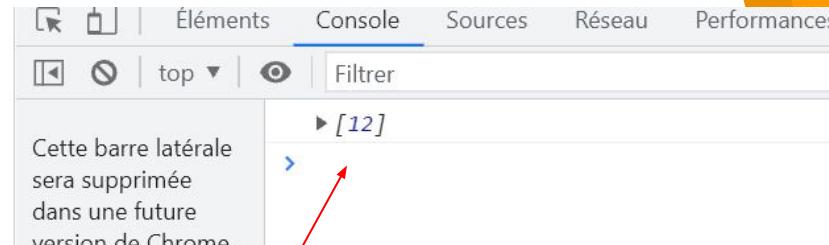


```
<script>
    const variable = 12;
    variable = 13;
    console.log(variable);
</script>
</body>
```

“Const” (qui veut dire “constante”) ne peut pas être réassignée (modifiée).  
Elle garde la valeur qu’elle a reçu lors de son initialisation.  
En gros, c’est une variable qui ne change jamais.

# Algorithmie - Les variables

Lorem ipsum dolor sit amet consectetur adipisicing elit. Facilis esse, incidunt culpa, omnis ipsum voluptatibus nostrum maiores suscipit veritatis, ipsam aliquam nesciunt voluptates molestias reiciendis illum nobis. Molestiae, odio voluptates.



```
<script>
    const variable = [];
    variable.push(12);
    console.log(variable);
</script>
```

En revanche, si “const” est un tableau, on peut modifier le contenu du tableau,  
Car techniquement, on a pas modifier sa valeur qui est “tableau”....  
La fonction .push() va ajouter la valeur entre parenthèse au tableau.

## Algorithmie - Les variables

Une autre notion importante, est celle de la portée d'une variable...

La « portée » d'une variable désigne l'espace du script dans laquelle elle va être accessible. Toutes nos variables ne sont pas automatiquement disponibles à n'importe quel endroit dans un script et on ne va donc pas toujours pouvoir les utiliser. Cela dépend d'où la variable a été initialisée.

# Algorithmie - Les variables

```
<script>
    //Variable globale car disponible dans tout le document
    let variable="hey !";

    //Bloc conditionnel
    if(1==1){
        //variable locale, disponible UNIQUEMENT dans le bloc conditionnel
        let variable2="oh ! ";

        console.log('Disponible dans le bloc conditionnel : '+variable);
        console.log('Disponible dans le bloc conditionnel : '+variable2);
    }

    console.log('Disponible dans la page : '+variable);
    console.log('Disponible dans la page : '+variable2);
</script>
//dehors
```

La variable “variable” est disponible partout dans le script, car initialisée hors bloc (on verra plus en détail plus tard) et en haut de page. (Je ne peux pas utiliser cette variable au dessus de son initialisation). On dit qu’elle est **globale**.

La variable “variable2” n’est disponible uniquement dans le bloc conditionnel (entre les accolades), car elle a été initialisée dans ce bloc. Je ne peux pas l’afficher en dehors. On dit qu’elle est **locale**.

# Algorithmie - Les variables

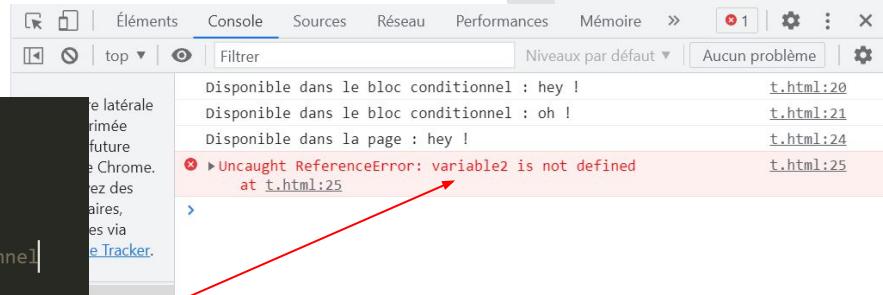
Ipsum dolor sit amet consectetur adipisicing elit. Facilis esse, incidunt culpa, omnis ipsum voluptatibus nostrum maiores suscipit veritatis, ipsam aliquam nesciunt voluptates molestias reiciendis illum nobis. Molestiae, odio voluptates.

```
<script>
    //Variable globale car disponible dans tout le document
    let variable="hey !";

    //Bloc conditionnel
    if(1==1){
        //variable locale, disponible UNIQUEMENT dans le bloc conditionnel
        let variable2="oh ! ";

        console.log('Disponible dans le bloc conditionnel : '+variable);
        console.log('Disponible dans le bloc conditionnel : '+variable2);
    }

    console.log('Disponible dans la page : '+variable);
    console.log('Disponible dans la page : '+variable2);
</script>
```



Normalement, je devrais avoir 4 messages. 2 commençant par “Disponible dans le bloc conditionnel” et 2 commençant par “Disponible dans la page”.

Mais comme je ne peux pas utiliser “variable2” en dehors du bloc conditionnel (et que j’essaie de l’afficher quand même), le JS me génère une erreur (en me disant que “variable2” n’est pas définie).

# Algorithmie - Les variables

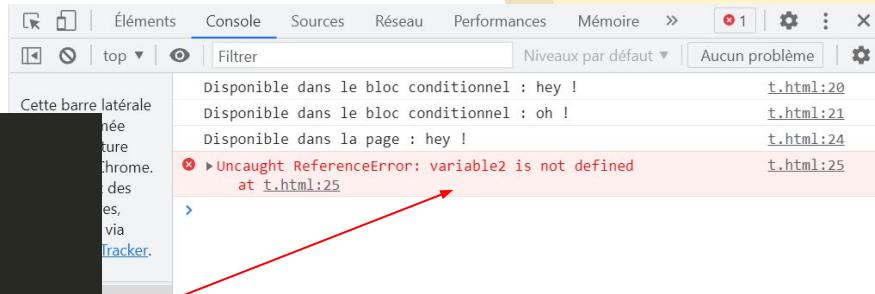
Ipsum dolor sit amet consectetur adipisicing elit. Facilis esse, incidunt culpa, omnis ipsum voluptatibus nostrum maiores suscipit veritatis, ipsam aliquam nesciunt voluptates molestias reiciendis illum nobis. Molestiae, odio voluptates.

```
<script>
    //Variable globale car disponible dans tout le document
    const variable="hey !";

    //Bloc conditionnel
    if(1==1){
        //variable locale, disponible UNIQUEMENT dans le bloc conditionnel
        const variable2="oh ! ";

        console.log('Disponible dans le bloc conditionnel : '+variable);
        console.log('Disponible dans le bloc conditionnel : '+variable2);
    }

    console.log('Disponible dans la page : '+variable);
    console.log('Disponible dans la page : '+variable2);
</script>
```



Ce qui est valable pour "let" l'est aussi pour "const"

## Algorithmie - Les variables

Autre notion, la concaténation. Concaténer signifie littéralement « mettre bout à bout ». La concaténation est un mot généralement utilisé pour désigner le fait de rassembler deux chaînes de caractères pour en former une nouvelle. La concaténation va nous permettre de mettre bout-à-bout une chaîne de caractères et la valeur d'une variable par exemple.

# Algorithmie - Les variables

Lorem ipsum dolor sit amet consectetur adipisicing elit. Facilis esse, incidunt culpa, omnis ipsum voluptatibus nostrum maiores suscipit veritatis, ipsam aliquam nesciunt voluptates molestias reiciendis illum nobis. Molestiae, odio voluptates.

```
<script>
    //Variable globale car disponible dans tout le document
    const variable="hey !";

    //Bloc conditionnel
    if(1==1){
        //variable locale, disponible UNIQUEMENT dans le bloc conditionnel
        const variable2="oh ! ";

        console.log('Disponible dans le bloc conditionnel : '+variable);
        console.log('Disponible dans le bloc conditionnel : '+variable2);
    }

    console.log('Disponible dans la page : '+variable);
    console.log('Disponible dans la page : '+variable2);
</script>
```



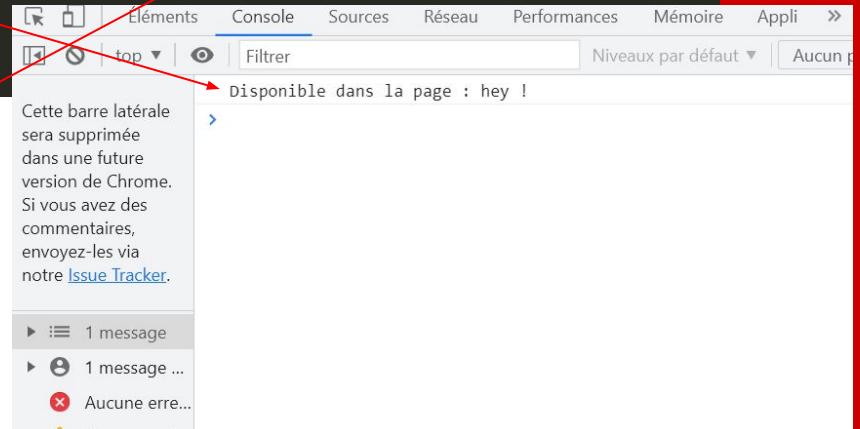
D'ailleurs, nous l'avons vu dans les exercices précédents...

En JavaScript, l'opérateur de concaténation est le signe +. Faites bien attention ici : lorsque le signe + est utilisé avec deux nombres, il sert à les additionner. Lorsqu'il est utilisé avec autre chose que deux nombres, il sert d'opérateur de concaténation.

# Algorithmie - Les variables

```
<script>  
  
    const variable="hey !";  
  
    console.log('Disponible dans la page : '+variable);  
  
</script>  
dy>
```

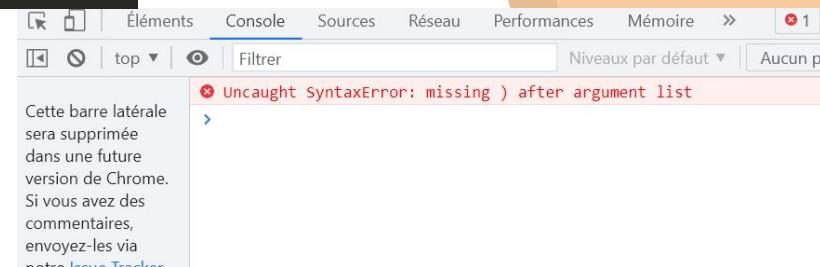
J'ai allégé un peu le code. On voit que la chaîne de caractère est entre les guillemets et que la variable est après le signe +



# Algorithmie - Les variables

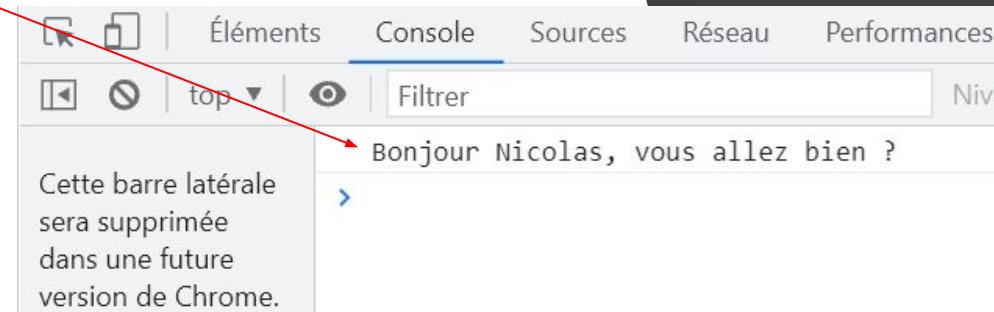
```
<script>  
  
    const variable="hey !";  
  
    console.log('Disponible dans la page : ' variable);  
  
</script>
```

Si j'oublie le signe +, nous avons une erreur...



# Algorithmie - Les variables

```
<script>  
  
    const variable="Nicolas";  
  
    console.log('Bonjour '+variable+', vous allez bien ?');  
  
</script>  
ody>
```



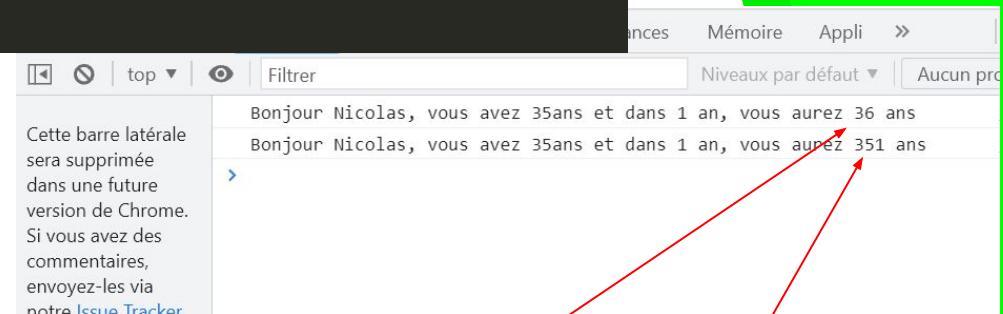
On peut également intercaler une variable entre 2 chaînes de caractère

# Algorithmie - Les variables

```
const age = 35;
const nom ="Nicolas";
const text = 'Bonjour '+nom+', vous avez '+ age +'ans et dans 1 an, vous aurez '+ (age+1) +' ans';
const text2 = 'Bonjour '+nom+', vous avez '+ age +'ans et dans 1 an, vous aurez '+ age+1 +' ans';

console.log(text);
console.log(text2);

script>
```

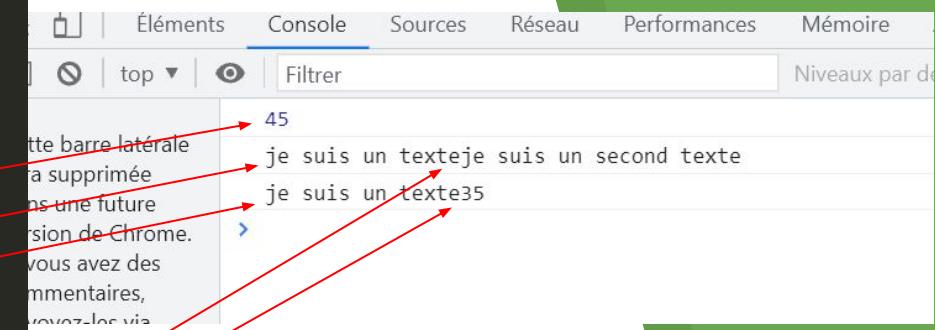


Dans l'exemple ci-dessus, nous voyons que JS peut soit concaténer (text2, il a collé 35 et 1) soit additionner (text). Le fait d'avoir écrit (age+1), JS comprend qu'il doit les additionner car "age" est un chiffre comme 1.

Dans "text2" le JS a "chaine de caractère (...vous aurez) + chiffre (35) + chiffre (1) + chaine de caractère (an)" . Comme il ne peut pas additionner une chaîne de caractère avec des nombres, il les concatènent

# Algorithmie - Les variables

```
const var1 = 35;  
const var2 = 10;  
let var3 ="je suis un texte";  
let var4 = "je suis un second texte";  
  
console.log(var1 + var2);  
console.log(var3 + var4);  
console.log(var3 + var1);  
/script>
```



Dans le premier cas, JS additionne les 2 variables car ce sont 2 nombres

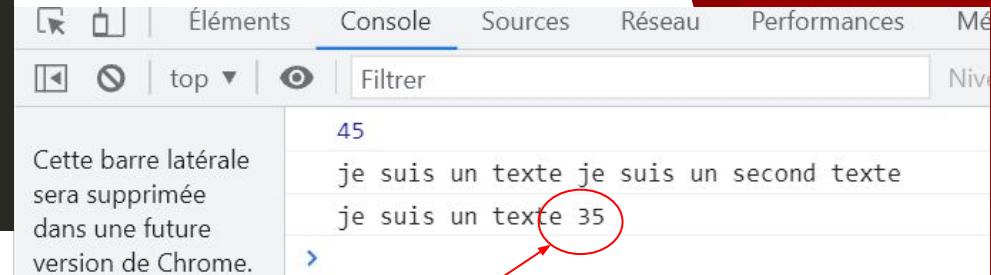
Dans le deuxième cas, JS les concatène car ce sont 2 chaînes de caractères

Dans le derniers cas, JS les concatène car on ne peut pas additionner un chiffre et une chaîne de caractères.

Petite parenthèse, on voit qu'il n'y a pas d'espace entre les 2 variables concaténées.

# Algorithmie - Les variables

```
const var1 = 35;  
const var2 = 10;  
let var3 ="je suis un texte";  
let var4 = "je suis un second texte";  
  
console.log(var1 + var2);  
console.log(var3 +' '+ var4);  
console.log(var3 +' '+ var1);  
script>
```



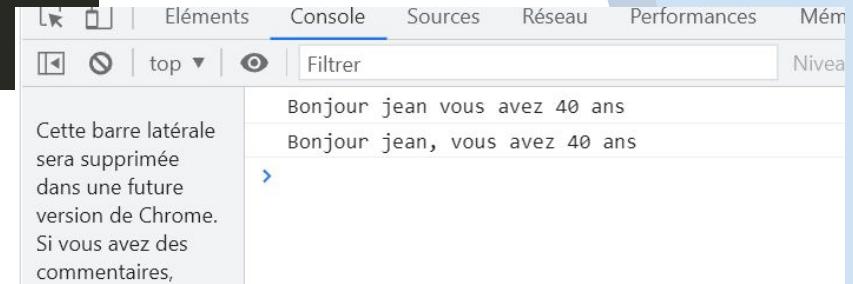
Pour mettre un espace entre 2 variables, il faut mettre une chaîne de caractère vide (un espace entre 2 guillemets)

## Algorithmie - Les variables

Et maintenant voyons....les littéraux de gabarits !

# Algorithmie - Les variables

```
<script>  
  
    const nom = 'jean';  
    const age = 40;  
  
    console.log('Bonjour '+nom+ ' vous avez '+age+ ' ans');  
  
    console.log(`Bonjour ${nom}, vous avez ${age} ans`);  
  
</script>  
body>
```



Il existe une méthode plus récente pour la mise en forme d'un texte et la concaténation.

Pour ce faire, on ne met pas le texte en guillemets (simple ou double), mais entre "accents graves". Et de faire passer les variable dans une syntaxe particulière \${ma\_variable} mais plus besoin du signe +. La variable est directement intégrée dans la chaîne de caractère.

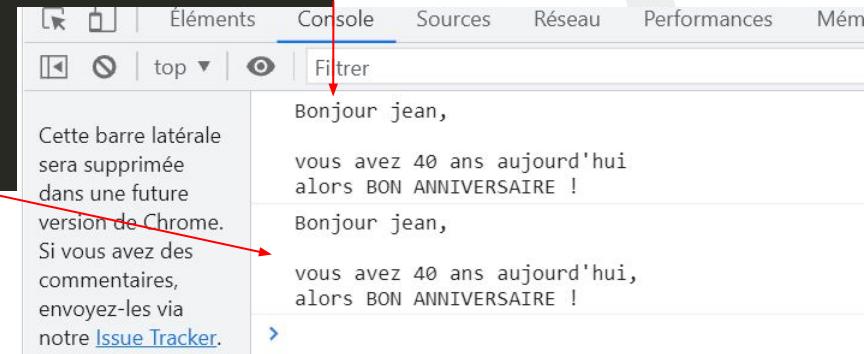
# Algorithmie - Les variables

```
const nom = 'jean';
const age = 40;

const message = 'Bonjour ' + nom + ',\n\nvous avez ' + age + ' ans aujourd\'hui\nalors BON ANNIVERSAIRE !';

const message2 =
`Bonjour ${nom},
vous avez ${age} ans aujourd'hui,
alors BON ANNIVERSAIRE !`;

console.log(message);
console.log(message2);
```



De plus, les littéraux de gabarits gèrent les sauts de ligne, alors que sans cette technique, j'ai besoin d'insérer \n (qui signifie saut de ligne).

Du coup, dans mon code, la variable "message2" est plus facilement lisible que "message"...

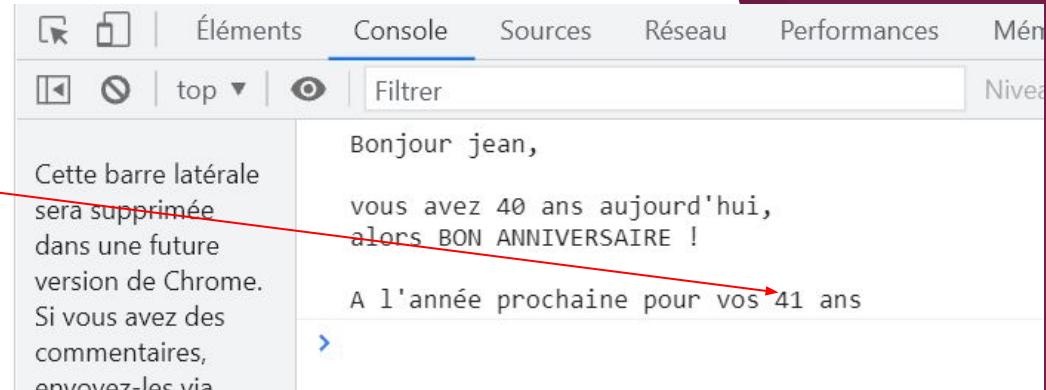
# Algorithmie - Les variables

```
const nom = 'jean';
const age = 40;

const message2 =
`Bonjour ${nom},
vous avez ${age} ans aujourd'hui,
alors BON ANNIVERSAIRE !

A l'année prochaine pour vos ${age + 1} ans`;

console.log(message2);
```



Bien entendu, on peut également passer des calculs mathématique à l'intérieur des accolades....

# **Algorithmie, Les Opérateurs**

## Algorithmie - Les Opérateurs

Un opérateur est un symbole qui va être utilisé pour effectuer certaines actions notamment sur les variables et leurs valeurs.

Il existe différents types d'opérateurs qui vont nous servir à réaliser des opérations de types différents. Les plus fréquemment utilisés sont :

- Les opérateurs arithmétiques ;
- Les opérateurs d'affectation / d'assignation ;
- Les opérateurs de comparaison ;
- Les opérateurs d'incrémentation et décrémentation ;
- Les opérateurs logiques ;
- L'opérateur de concaténation (vu plus haut) ;
- L'opérateur ternaire ;
- ...

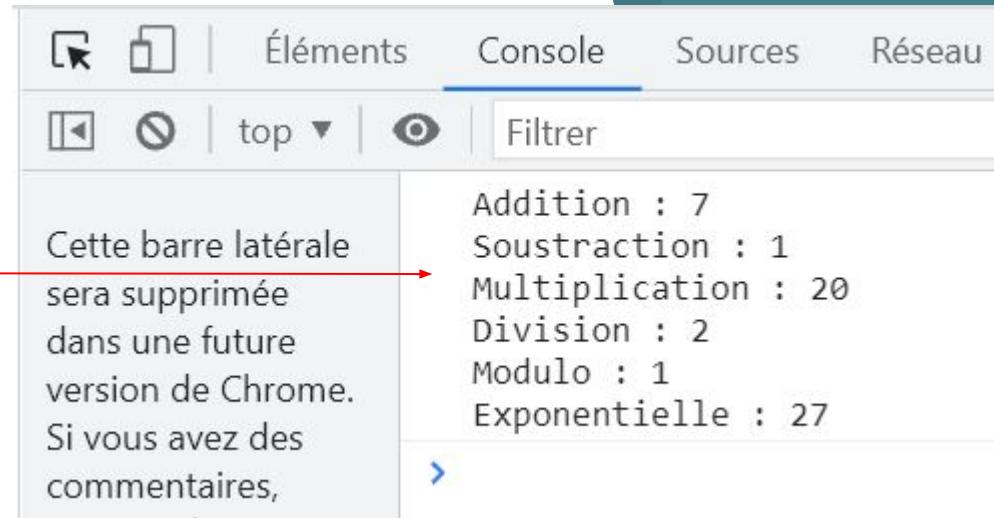
# Algorithmie - Les Opérateurs

Opérateur	Nom de l'opération associée
+	Addition
-	Soustraction
*	Multiplication
/	Division
%	Modulo (reste d'une division euclidienne)
**	Exponentielle (élévation à la puissance d'un nombre par un autre)

Voici les opérateurs arithmétiques...

# Algorithmie - Les Opérateurs

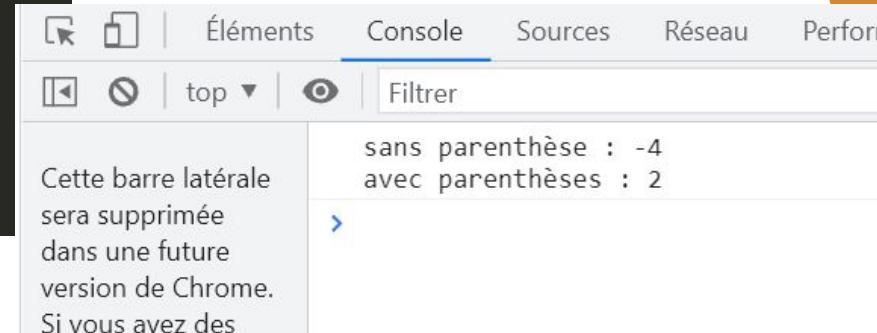
```
const num = 2;  
const num2 = 3;  
const num3 = 4;  
  
const result =  
`Addition : ${num3 + num2}  
Soustraction : ${num2 - num}  
Multiplication : ${num * 10}  
Division : ${num3 / num}  
Modulo : ${num3 % num2}  
Exponentielle : ${num2**3}  
`;
```



En action... Pour rappel, le modulo c'est le reste d'une division.

# Algorithmie - Les Opérateurs

```
const result =  
`sans parenthèse : ${1 - 2 - 3}  
avec parenthèses : ${1 -(2 - 3)}  
`;  
  
console.log(result);
```



Il est possible de faire passer des priorités de calcul via des parenthèses.  
Sans parenthèse, JS exécute le calcul de gauche à droite ( $1 - 2 = -1$  puis  $-1 - 3 = -4$ )  
Avec parenthèse, JS exécute le calcul entre les parenthèses d'abord ( $2 - 3 = -1$  puis  $1 - -1 = 2$ )

# Algorithmie - Les Opérateurs

Opérateur	Définition
<code>+=</code>	Additionne puis affecte le résultat
<code>-=</code>	Soustrait puis affecte le résultat
<code>*=</code>	Multiplie puis affecte le résultat
<code>/=</code>	Divise puis affecte le résultat
<code>%=</code>	Calcule le modulo puis affecte le résultat

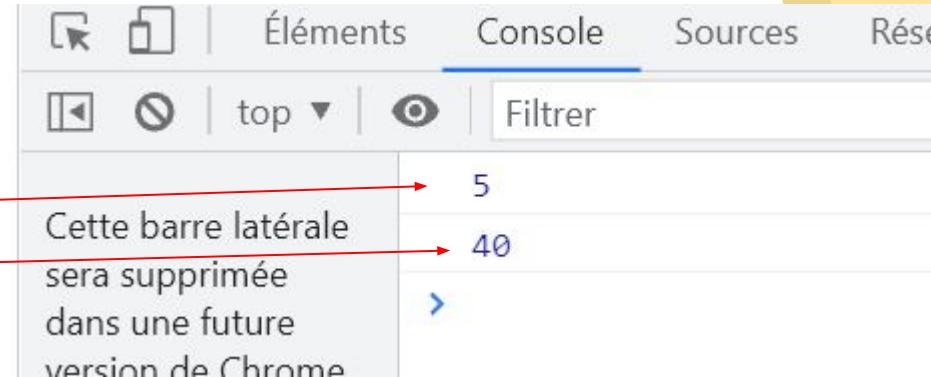
Voici les opérateurs d'affectation / d'assignation

# Algorithmie - Les Opérateurs

```
Let num = 2;  
Let num2 = 3;  
Let num3 = 4;
```

```
num += num2;  
num3 *= 10;
```

```
console.log(num);  
console.log(num3);
```



Il faut les voir comme un raccourci :

“num” est égal à 2, puis on additionne “num” (donc 2) à “num2” (soit 3) et on stock le résultat dans “num”.

“num += num2” c'est comme “num = num + num2”

“num3 \*= 10” c'est comme “num3 = num3 \* 10”

D'ailleurs le signe = est également un opérateur d'assignation.

# Algorithmie - Les Opérateurs

Opérateur	Définition
<code>==</code>	Permet de tester l'égalité sur les valeurs
<code>====</code>	Permet de tester l'égalité en termes de valeurs et de types
<code>!=</code>	Permet de tester la différence en valeurs
<code>&lt;&gt;</code>	Permet également de tester la différence en valeurs
<code>!==</code>	Permet de tester la différence en valeurs ou en types
<code>&lt;</code>	Permet de tester si une valeur est strictement inférieure à une autre
<code>&gt;</code>	Permet de tester si une valeur est strictement supérieure à une autre
<code>&lt;=</code>	Permet de tester si une valeur est inférieure ou égale à une autre
<code>&gt;=</code>	Permet de tester si une valeur est supérieure ou égale à une autre

Voici les opérateurs de comparaison

# Algorithmie - Les Opérateurs

Opérateur	Définition
<code>==</code>	Permet de tester l'égalité sur les valeurs
<code>====</code>	Permet de tester l'égalité en termes de valeurs et de types
<code>!=</code>	Permet de tester la différence en valeurs
<code>&lt;&gt;</code>	Permet également de tester la différence en valeurs
<code>!==</code>	Permet de tester la différence en valeurs ou en types
<code>&lt;</code>	Permet de tester si une valeur est strictement inférieure à une autre
<code>&gt;</code>	Permet de tester si une valeur est strictement supérieure à une autre
<code>&lt;=</code>	Permet de tester si une valeur est inférieure ou égale à une autre
<code>&gt;=</code>	Permet de tester si une valeur est supérieure ou égale à une autre

Certains opérateurs vous sont déjà familier, on ne va pas revenir dessus...  
Nous en reparlerons au moment des conditions.

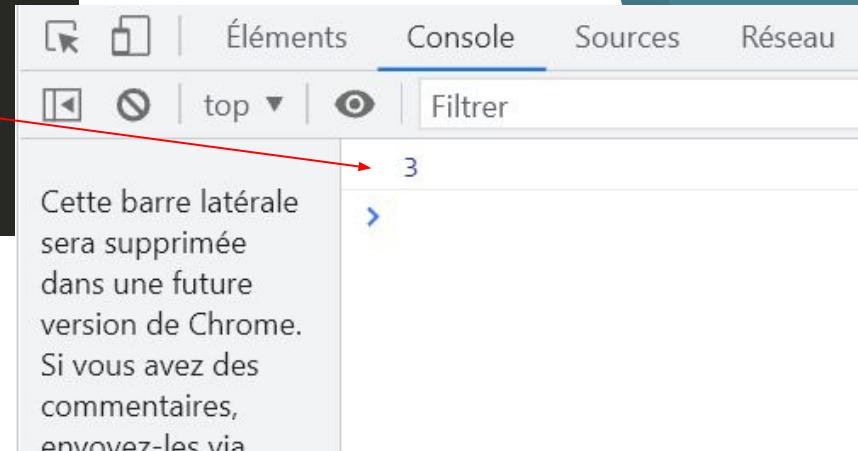
# Algorithmie - Les Opérateurs

Exemple (opérateur + variable)	Résultat
<code>++x</code>	Pré-incrémantation : incrémente la valeur contenue dans la variable x, puis retourne la valeur incrémentée
<code>x++</code>	Post-incrémantation : retourne la valeur contenue dans x avant incrémantation, puis incrémente la valeur de \$x
<code>--x</code>	Pré-décrémantation : décrémente la valeur contenue dans la variable x, puis retourne la valeur décrémentée
<code>x--</code>	Post-décrémantation : retourne la valeur contenue dans x avant décrémantation, puis décrémente la valeur de \$x

## Les opérateurs d'incrémantation et décrémantation

# Algorithmie - Les Opérateurs

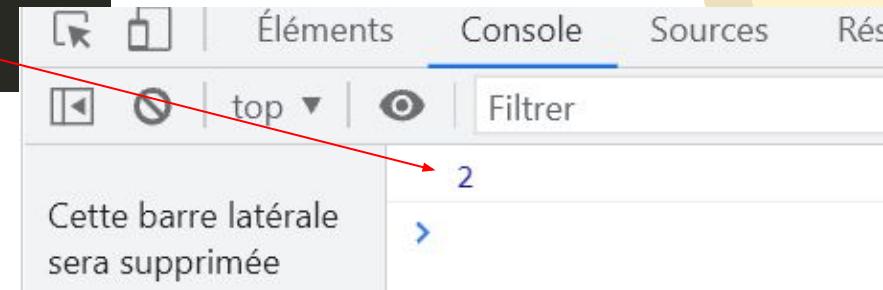
```
Let num = 2;  
Let num2 = 3;  
Let num3 = 4;  
  
console.log(++num); // num = num + 1;
```



Le fait de mettre ++ avant le chiffre on dit à JS “d’abord tu incrémentes de 1 et ensuite tu m’affiches le résultat).

# Algorithmie - Les Opérateurs

```
Let num = 2;  
Let num2 = 3;  
Let num3 = 4;  
  
console.log(num++); // num = num + 1;
```

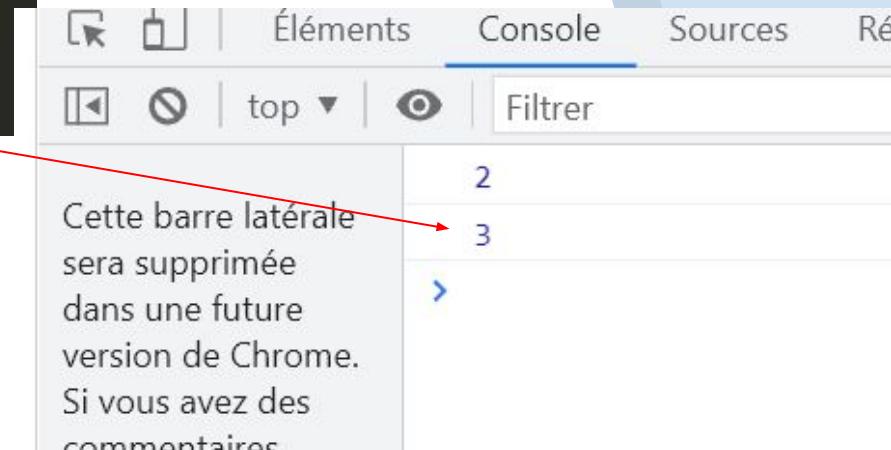


Le fait de mettre ++ après le chiffre on dit à JS “d’abord tu m’affiches le résultat PUIS tu incrémentes de 1).

# Algorithmie - Les Opérateurs

```
Let num = 2;  
Let num2 = 3;  
Let num3 = 4;
```

```
console.log(num++); // num = num + 1;  
console.log(num);|
```



D'ailleurs, si je décide de ré-afficher la variable "num" après l'incrémentation, on voit qu'elle a bien augmenté de 1. Pas de slide pour le -- (décrémentation) ou -1 mais c'est le même combat que ++.

# Algorithmie - Les Opérateurs

Opérateur (nom)	Opérateur (symbole)	Description
AND (ET)	&&	Lorsqu'il est utilisé avec des valeurs booléennes, renvoie <b>true</b> si toutes les comparaisons sont évaluées à <b>true</b> ou <b>false</b> sinon
OR (OU)		Lorsqu'il est utilisé avec des valeurs booléennes, renvoie <b>true</b> si au moins l'une des comparaisons est évaluée à <b>true</b> ou <b>false</b> sinon
NO (NON)	!	Renvoie <b>false</b> si une comparaison est évaluée à <b>true</b> ou renvoie <b>true</b> dans le cas contraire

Les opérateurs de logique (que nous verrons plus en détails dans la partie des conditions) comme les opérateurs ternaires...(var ? 'un truc' : 'un autre truc')

# **Algorithmie, Les Conditions**

## Algorithmie - Les Conditions

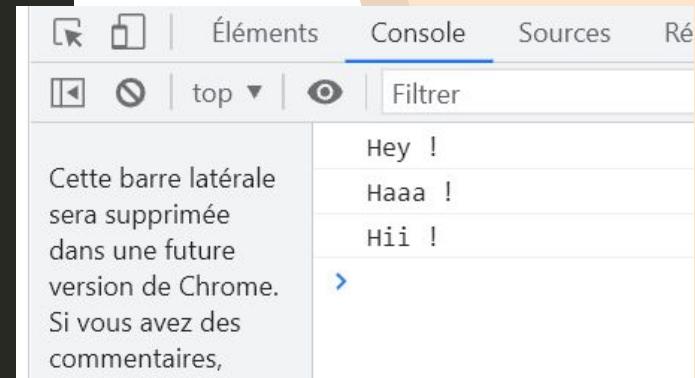
```
if(ma_condition_est_vraie){  
    console.log(alors_jaffiche_ce_message);  
}  
:  
:
```

La condition **if** est l'une des conditions les plus utilisées et est également la plus simple à appréhender puisqu'elle va juste nous permettre d'exécuter un bloc de code si et seulement si le résultat d'un test vaut **true**.

# Algorithmie - Les Conditions

```
Let num = 2;
Let num2 = 3;
Let num3 = 4;

//Si num est égal à 2
//VRAI
if(num == 2){
    console.log('Hey !');
}
//Si num est égal à num2 (si 2 est égal à 3)
//FAUX
if(num == num2){
    console.log('Hooo !');
}
//Si num est plus petit que num2 (si 2 est plus petit que 3)
//VRAI
if(num < num2){
    console.log('Haaa !');
}
//Si num est plus grand OU égal à num2 (si 2 est plus grand OU égal à 3)
//VRAI
if(num >= 2){
    console.log('Hii !');
}
```



Voici des exemples simples de conditions. A chaque fois que ce qu'il y a entre parenthèses est vrai, ce qu'il y a entre les accolades sera exécuté ! Dans l'exemple ci-dessus la seule condition fausse est la deuxième, "Hooo !" n'est pas affiché.

# Algorithmie - Les Conditions

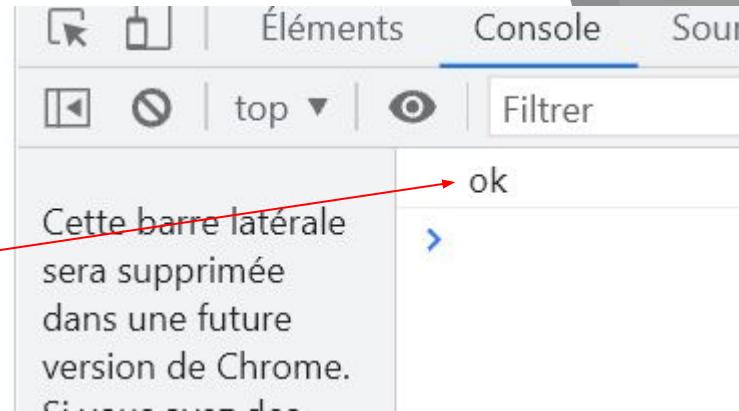
```
Let num = 2;
Let num2 = 3;
Let num3 = 4;

//Si num est égal à 2
//VRAI
if(num == 2){
    console.log('Hey !');
}
//Si num est égal à num2 (si 2 est égal à 3)
//FAUX
if(num == num2){
    console.log('Hooo !');
}
//Si num est plus petit que num2 (si 2 est plus petit que 3)
//VRAI
if(num < num2){
    console.log('Haaa !');
}
//Si num est plus grand OU égal à num2 (si 2 est plus grand OU égal à 3)
//VRAI
if(num >= 2){
    console.log('Hii !');
}
```

Notons que pour comparer, on utilise `==` et non `=`. Car `=` est un opérateur d'assignation et non de comparaison (ex : `let variable = 12;` On assigne la valeur de 12 à la variable “variable”).

# Algorithmie - Les Conditions

```
let num = 2;  
let num2 = 3;  
let num3 = 4;  
let var1 ="";  
  
if(num){  
    console.log('ok');  
}  
  
if(var1){  
    console.log('coucou');  
}
```

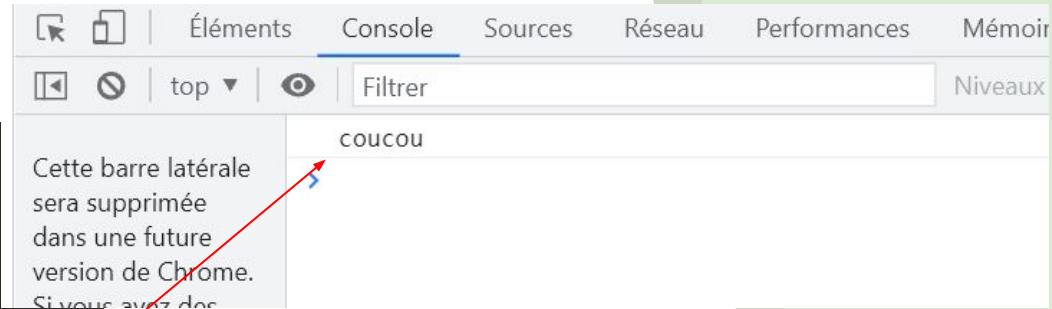


Si aucun opérateur (==,>,>+,...) n'est présent dans la condition "if", JS va traduire ça par "est-ce que "num" existe.... Comme "num" est égal à 2, elle existe.

En revanche la variable "var1" est égal à "" (soit rien), JS considère qu'elle n'existe pas (ce qui est logique car par définition "rien" n'existe pas).

# Algorithmie - Les Conditions

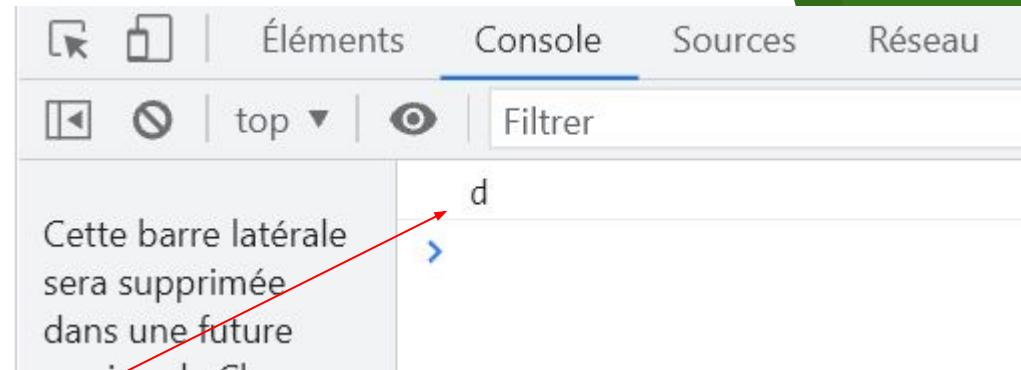
```
let var1 = " ";  
  
if(var1){  
    console.log('coucou');  
}  
....
```



Par contre, si "var1" est égal à un espace, JS considère que ce n'est pas "rien", donc le code s'exécute

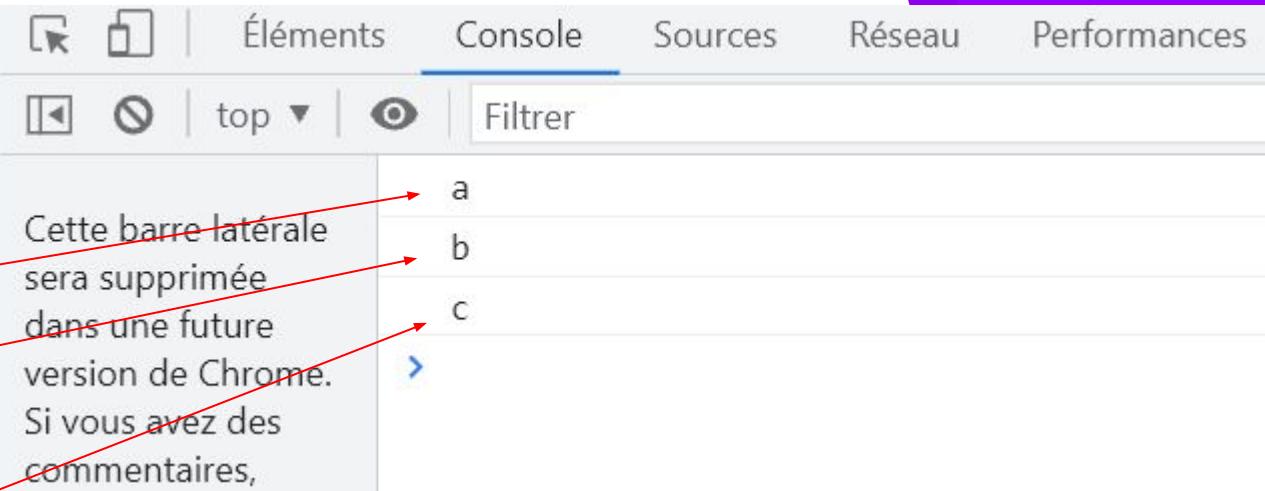
# Algorithmie - Les Conditions

```
let a = false;  
let b = 0;  
let c = '';  
let d = '.';  
  
if(a){  
    console.log('a');  
}  
if(b){  
    console.log('b');  
}  
if(c){  
    console.log('c');  
}  
if(d){  
    console.log('d');  
}
```



Considérer comme vide / nul : False; 0; null; undefined; NaN (« Not a Number ») = « n'est pas un nombre »); " (chaine de caractère vide)

# Algorithmie - Les Conditions



The screenshot shows a browser's developer tools open to the 'Console' tab. On the left, there is a code editor window containing the following JavaScript code:

```
let a = false;
let b = 0;
let c = '';
let d = '..'

if(!a){
    console.log('a');
}
if(!b){
    console.log('b');
}
if(!c){
    console.log('c');
}
if(!d){
    console.log('d');
}
```

The console output on the right shows the values of variables a, b, c, and d:

- a
- b
- c

A tooltip is displayed over the first three items (a, b, c) in the list, containing the following text:

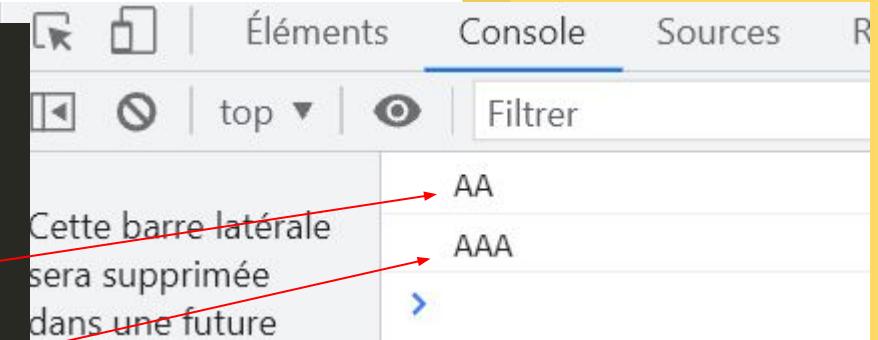
Cette barre latérale sera supprimée dans une future version de Chrome. Si vous avez des commentaires,

Red arrows from the text "Cette barre latérale sera supprimée dans une future version de Chrome. Si vous avez des commentaires," point to the variable names a, b, and c in the list.

L'opérateur “!” signifie “pas”. JS traduit ça par “si la variable n'existe pas, alors j'exécute le code. C'est le comportement inverse.

# Algorithmie - Les Conditions

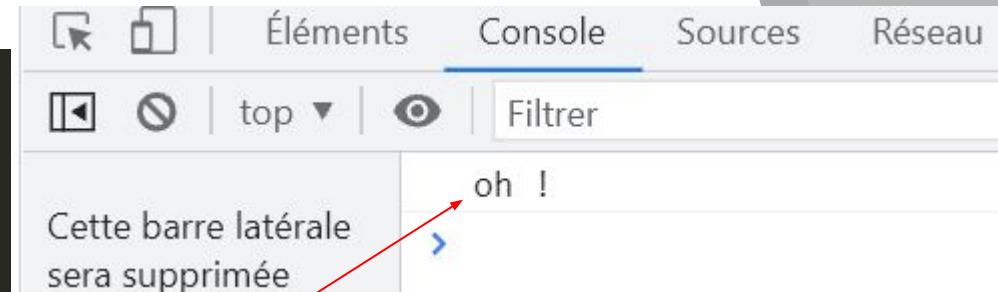
```
let a = '';  
  
//3 n'est pas égal à 4 donc VRAI puis vrai est égal à faux donc FAUX  
//FAUX  
if((3 != 4) == false){  
    console.log('A');  
}  
//2 n'est pas égal à 2 c'est FAUX puis faux est égal à faux c'est VRAI  
//VRAI  
if((2!=2) == false){  
    console.log('AA');  
}  
//a est égal à rien (donc faux) donc faux est égal à faux, c'est VRAI  
//VRAI  
if(a==false){  
    console.log('AAA');  
}
```



Comme pour l'arithmétique, on peut utiliser les parenthèses pour renverser la valeur logique (! fait très bien l'affaire mais c'est pour exemple)

# Algorithmie - Les Conditions

```
Let num = 2;  
Let num2 = 3;  
Let num3 = 4;  
  
if(num != 3){  
    console.log('oh !');  
}  
|
```



Dans cet exemple, on dit au JS, “si la variable ‘num’ n’est pas égal à 3, alors exécute le code”. (donc si 2 n’est pas égal à 3). Attention, cela ne fonctionne pas avec un autre opérateur que = (ex : !=)

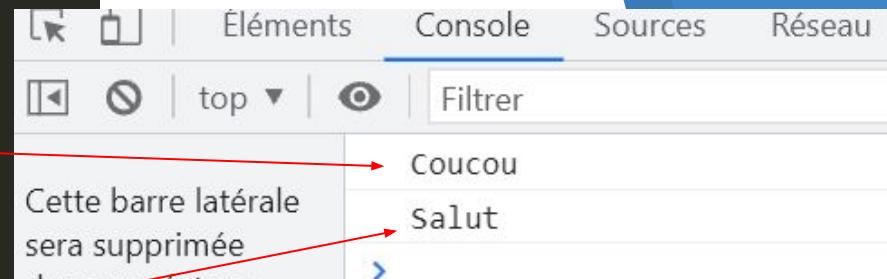
# Algorithmie - Les Conditions

Opérateur	Définition
<code>==</code>	Permet de tester l'égalité sur les valeurs
<code>====</code>	Permet de tester l'égalité en termes de valeurs et de types
<code>!=</code>	Permet de tester la différence en valeurs
<code>&lt;&gt;</code>	Permet également de tester la différence en valeurs
<code>!==</code>	Permet de tester la différence en valeurs ou en types
<code>&lt;</code>	Permet de tester si une valeur est strictement inférieure à une autre
<code>&gt;</code>	Permet de tester si une valeur est strictement supérieure à une autre
<code>&lt;=</code>	Permet de tester si une valeur est inférieure ou égale à une autre
<code>&gt;=</code>	Permet de tester si une valeur est supérieure ou égale à une autre

Il nous reste 3 opérateurs de comparaison à voir

# Algorithmie - Les Conditions

```
//La variable 'a' est une chaîne de caractère car entre guillemets  
//Ce n'est pas un nombre !  
let a = '12';  
  
//a est-il égal à 12  
//VRAI  
if(a == 12){  
    console.log('Coucou');  
}  
//a 12 est-il égal à 12 ET de type nombre ??  
//FAUX  
if(a === 12){  
    console.log('Bonsoir');  
}  
//a 12 est-il égal à 12 ET de est-il une chaîne de caractère ??  
//VRAI  
if(a === '12'){  
    console.log('Salut');  
}
```

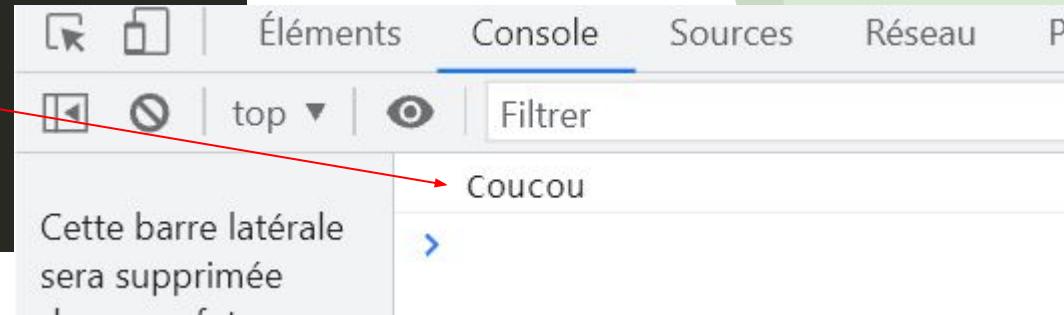


Le triple égal (==>) vérifie la valeur ET le type de variable.

Il est plus strict que == qui ne vérifie que la valeur mais peut éviter certaines erreurs. Pour JS 12 et '12' ont la même valeur, cependant pas le même type.

# Algorithmie - Les Conditions

```
//La variable 'a' est une chaîne de caractère car entre guillemets  
//Ce n'est pas un nombre !  
Let a = '12';  
  
if(a !== 12){  
    console.log('Coucou');  
}  
  
if(a !== '12'){  
    console.log('Salut');  
}
```



Du coup le `!==` fait l'opposé de l'opérateur `==`, il vérifie que la valeur ET le type sont bien différents.

# Algorithmie - Les Conditions

Opérateur	Définition
==	Permet de tester l'égalité sur les valeurs
====	Permet de tester l'égalité en termes de valeurs et de types
!=	Permet de tester la différence en valeurs
<>	Permet également de tester la différence en valeurs
!==	Permet de tester la différence en valeurs ou en types
<	Permet de tester si une valeur est strictement inférieure à une autre
>	Permet de tester si une valeur est strictement supérieure à une autre
<=	Permet de tester si une valeur est inférieure ou égale à une autre
>=	Permet de tester si une valeur est supérieure ou égale à une autre

Cet opérateur ne fonctionne plus et je n'ai pas trouver de documentation le concernant. Lorsqu'il est utilisé, il provoque une erreur. C'est l'équivalent de l'opérateur !=

## Algorithmie - Les variables

La condition if est une structure conditionnelle limitée par définition puisqu'elle ne nous permet d'exécuter un bloc de code que dans le cas où le résultat d'un test est évalué à true mais elle ne nous offre aucun support dans le cas contraire.

La structure conditionnelle if...else (« si... sinon » en français) va être plus complète que la condition if puisqu'elle va nous permettre d'exécuter un premier bloc de code si un test renvoie true ou un autre bloc de code dans le cas contraire.

## Algorithmie - Les Conditions

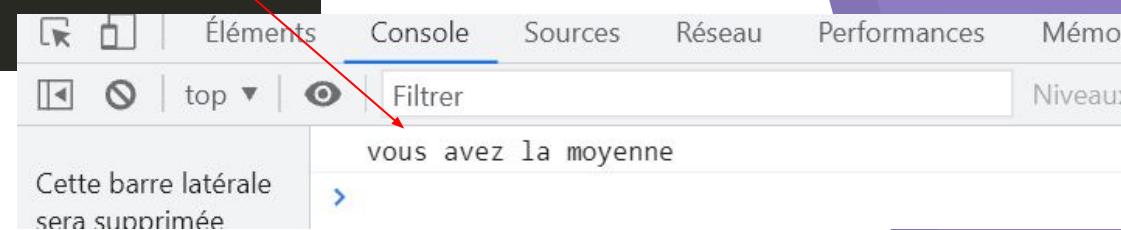
```
if(condition_est_vraie){  
    console.log('ce code est exécuté');  
}else{  
    console.log('sinon, ce code est exécuté');  
}
```

Si la condition entre les parenthèse est vraie, le premier code sera exécuté, sinon, ce sera le deuxième.

# Algorithmie - Les Conditions

```
let note = 12;

if(note >= 10){
    console.log('vous avez la moyenne');
}else{
    console.log('vous n\'avez pas la moyenne');
}
```



Si la variable “note” est supérieur ou égal à 10, le code affichera ‘vous avez la moyenne’ sinon le code ‘affichera ‘vous n’avez pas la moyenne’ . Le “else” est un filet de sécurité qui sera forcément exécuter si la condition est fausse.

# Algorithmie - Les Conditions

```
let note = 12;

if(note > 10){

    if(note === 10){
        console.log('Vous avez tout juste la moyenne');
    }else{
        console.log('Vous avez plus que la moyenne');
    }

}else{
    console.log('vous n\'avez pas la moyenne');
}
```



Il est possible d'imbriquer des conditions dans des conditions mais cette syntaxe n'est pas vraiment facile à lire....

# Algorithmie - Les Conditions

```
let note = 12;

if(note > 10){

    if(note === 10){
        console.log('Vous avez tout juste la moyenne');
    }else{
        console.log('Vous avez plus que la moyenne');
    }

}else{
    console.log('vous n\'avez pas la moyenne');
}
```

```
let note = 12;

if(note == 10){
    console.log('vous avez tout juste la moyenne');
}else if(note > 10){
    console.log('vous avez plus que la moyenne');
}else{
    console.log('vous n\'avez pas la moyenne');
}
```

Pour faciliter la lecture du code, il existe le  
“if(condition 1){....}else if(condition 2){...}else{...}”

# Algorithmie - Les Conditions

```
let note = 12;

if(note === 10){
    console.log('vous avez tout juste la moyenne');
}else if(note === 11){
    console.log(`Vous avez ${note}`);
}else if(note === 12){
    console.log(`Vous avez ${note}`);
}else{
    console.log('Vous avez soit moins de 10, soit plus de 12');
}
```

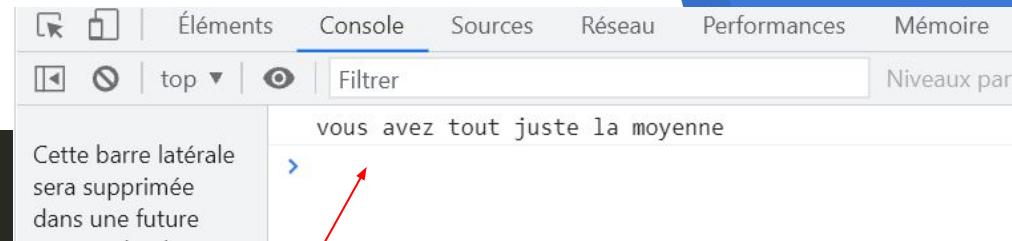


Vous pouvez chaîner autant de conditions que vous le souhaitez.

# Algorithmie - Les Conditions

```
let note = 12;

if(note > 10){
    console.log('vous avez tout juste la moyenne');
}else if(note === 11){
    console.log(`Vous avez ${note}`);
}else if(note === 12){
    console.log(`Vous avez ${note}`);
}else{
    console.log('Vous avez soit moins de 10, soit plus de 12');
}
```



Attention : Si vous avez plusieurs conditions vraies, JS s'arrêtera à la première !  
Dans l'exemple ci-dessus, la première et la troisième sont vraies. JS va quand même s'arrêter à la première.

# Algorithmie - Les Conditions

## Notion du SWITCH :

Le switch va nous permettre d'exécuter un code en fonction de la valeur d'une variable. On va pouvoir gérer autant de situations ou de « cas » que l'on souhaite.

Le switch représente une alternative à l'utilisation d'un if...else if...else.

Cependant, ces deux types d'instructions ne sont pas strictement équivalentes puisque dans un switch chaque cas va être lié à une valeur précise. En effet, l'instruction switch ne supporte pas l'utilisation des opérateurs de supériorité ou d'infériorité.

Dans certaines (rares) situations, il va pouvoir être intéressant d'utiliser un switch plutôt qu'un if...else if...else car cette instruction peut rendre le code plus clair et légèrement plus rapide dans son exécution.

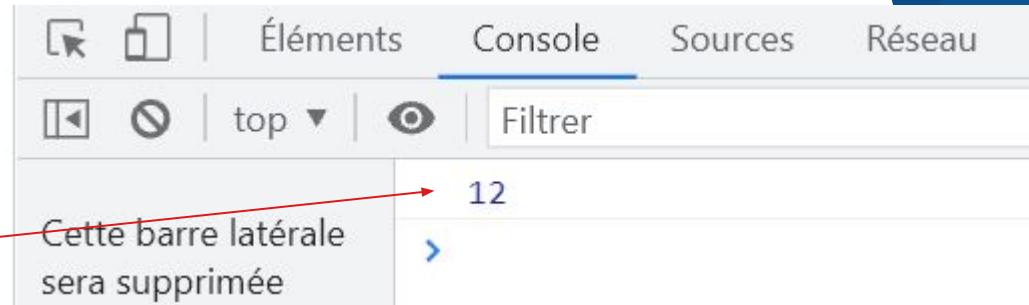
```
let note = 12;

switch (note) {
  case 10:
    console.log(10);
    break;
  case 11:
    console.log(11);
    break;
  case 12:
    console.log(12);
    break;
  case 13:
    console.log(13);
    break;
  default:
    console.log('nope');
    break;
}
```

# Algorithmie - Les Conditions

```
let note = 12;

switch (note) {
    case 10:
        console.log(10);
        break;
    case 11:
        console.log(11);
        break;
    case 12:
        console.log(12); // Line 12
        break;
    case 13:
        console.log(13);
        break;
    default:
        console.log('nope');
        break;
}
```



On vérifie la valeur entre les parenthèses du “switch”, puis, en fonction des cas (case), on exécute le code correspondant. A la fin de chaque cas (case), il y a un “break” pour stopper l’exécution du “switch”. A la fin de chaque “switch”, il y a un “default” (qui équivaut au “else”) pour afficher quelque chose au cas où aucun cas ne correspondent à la valeur du “switch”.

# Algorithmie - Les Conditions

## Les opérateurs logiques :

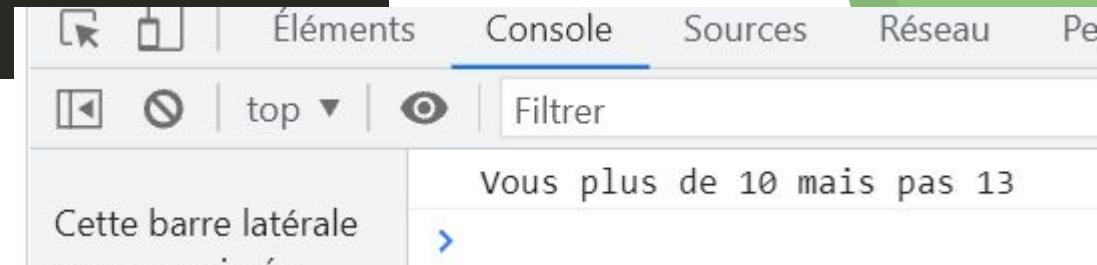
Les opérateurs logiques sont des opérateurs qui vont principalement être utilisés avec des valeurs booléennes et au sein de conditions.

Le JavaScript supporte trois opérateurs logiques : l'opérateur logique « ET », l'opérateur logique « OU » et l'opérateur logique « NON ».

Les opérateurs logiques « ET » et « OU » vont nous permettre d'effectuer plusieurs comparaisons dans une condition. Si on utilise l'opérateur « ET », toutes les comparaisons devront être évaluées à true pour que le test global de la condition retourne true. Dans le cas où n'utilise l'opérateur logique « OU », il suffira qu'une seule des comparaisons soit évaluée à true pour exécuter le code dans la condition.

# Algorithmie - Les Conditions

```
Let note = 12;  
  
if( note > 10 && note != 13){  
    console.log('Vous avez plus de 10 mais pas 13');  
}
```



La condition fonctionne uniquement si la note est strictement plus grande que 10 et n'est pas égale à 13...

## Algorithmie - Les Conditions

The screenshot shows a browser's developer tools open to the 'Console' tab. The code in the console is:

```
let note = 7;

if( !(note > 10) && note > 5){
    console.log('Vous avez mois de 10 et plus de 5');
}
```

The output in the console is:

Cette barre latérale > Vous avez mois de 10 et plus de 5

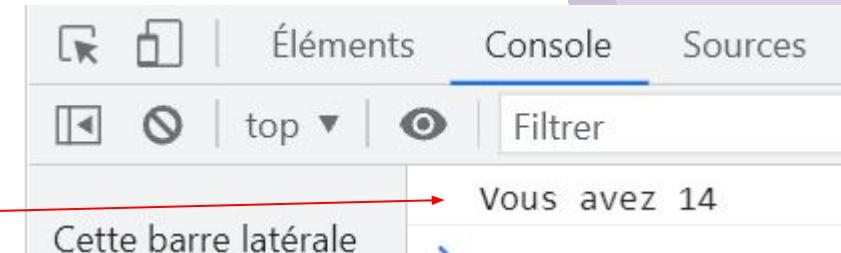
Si nous mettons des parenthèses autour d'une logique et un ! avant “!(logique)”, on inverse cette logique. En clair, on demande au JS “si la note est plus petite que 10...”

# Algorithmie - Les Conditions

```
let note = 7;

if( !(note > 10) || note == 14){
    console.log('Vous avez 14');
}

if( !(note > 10) && note == 14){
    console.log('Ooops !');
}
```



Dans la première condition, je demande au JS si ma note est plus petite que 10 OU si elle est égale à 14. Comme elle est inférieure à 10, le code s'exécute. Dans la deuxième, je lui dit "si la note est plus petite que 10 ET égale à 14..." ce qui est faux donc rien ne s'exécute...

# Table de vérité

Vrai && Vrai = Vrai

Vrai && Faux = Faux

Faux && Faux = Faux

Vrai || Vrai = Vrai

Vrai || Faux = Vrai

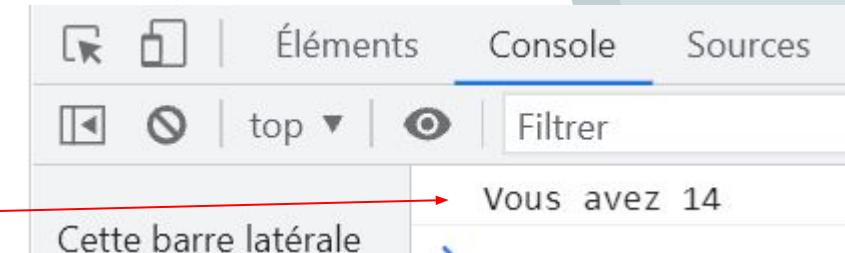
Faux || Faux = Faux

# Algorithmie - Les Conditions

```
let note = 7;

if( !(note > 10) || note == 14){
    console.log('Vous avez 14');
}

if( !(note > 10) && note == 14){
    console.log('Ooops !');
}
```



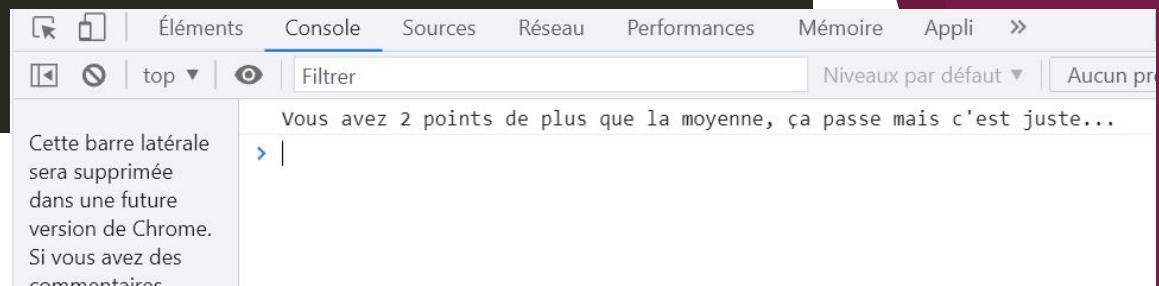
Première condition Vrai || Faux = Vrai

Deuxième condition Vrai && Faux = Faux

# Algorithmie - Les Conditions

```
let note = 12;

//Si la note est plus petite que 10 ET plus grande que 5
if(note < 10 && note > 7){
    console.log(`Vous avez ${note}, vous êtes bon pour les rattrapages`);
//Si la note est strictement égal à 10
}else if(note==10){
    console.log(`Vous avez tout juste 10, ne vous ratez au prochain test !`);
//Si la note est plus grande que 10 ET plus petite que 13 (il a 11 ou 12 en fait...)
}else if(note > 10 && note < 13){
    console.log(`Vous avez ${note - 10} points de plus que la moyenne, ça passe mais c'est juste...`);
//Si la note est de 13 ou plus...
}else if(note > 12){
    console.log(`Vous avez géré ;));
}else{
    console.log('Vous allez redoubler :(');
}
```



On peut cumuler “else if” et opérateurs logiques...

## Algorithmie - Les Conditions

```
let note = 11;

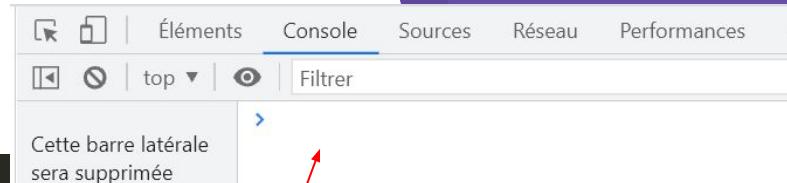
if( !(note > 10) && ( note % 2 == 0 ) ) {
    console.log('Serais-je exécuté ?');
}
```

D'après vous ??

# Algorithmie - Les Conditions

```
let note = 11;

if( !(note > 10) && ( note % 2 == 0 ) ) {
    console.log('Serais-je exécuté ?');
}
```



Et bien non...

Est-ce que note est plus petit que 10... non (Faux)

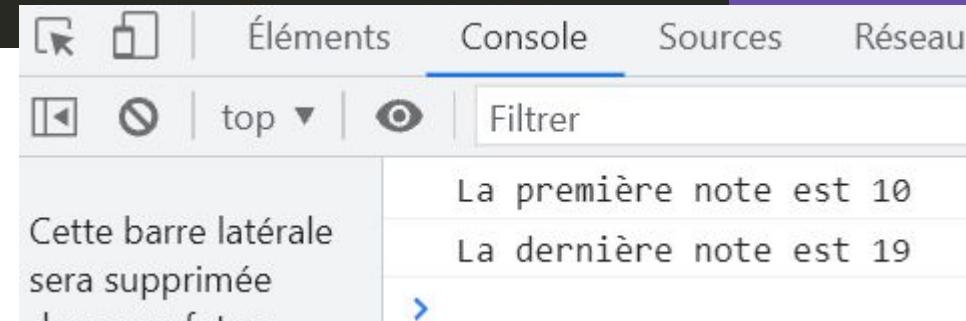
Est-ce que je peux diviser note par 2 et qu'il reste 0 (en gros, note est-il un nombre paire)...non (Faux)

Donc Faux && Faux = Faux

# **Algorithmie, Les Tableaux**

## Algorithmie - Les Tableaux

```
const notes = [10, 5, 14, 19];
console.log(`La première note est ${notes[0]}`);
console.log(`La dernière note est ${notes[3]}`);
```



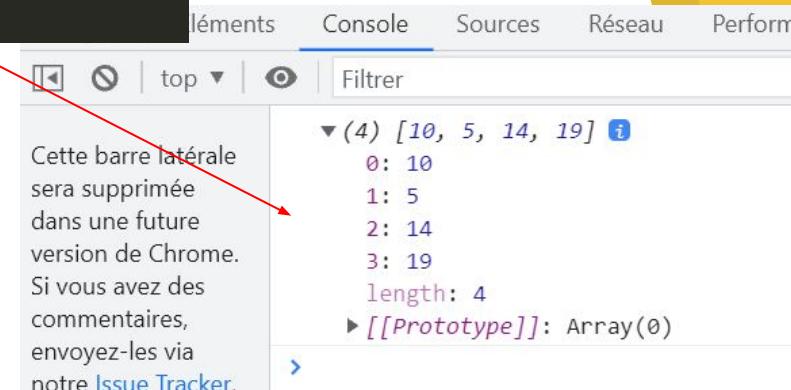
La variable "notes" est un tableau indexé.  
"10" est en première position, '5' en deuxième, etc...

Détail important, JS commence à compter à partir de 0. Donc le premier élément à l'index 0 et pas 1.

notes[0] // On demande la valeur 1 du tableau "notes" -> 10  
notes[3] // On demande la valeur 4 du tableau "notes" -> 19

# Algorithmie - Les Tableaux

```
const notes = [10, 5, 14, 19];
console.log(notes);
```



D'ailleurs, avec un “console.log” de notre tableau, on peut voir les index. On dit tableau indicé, car il n'y a pas de colonne, juste l'index des lignes.

# Algorithmie - Les Tableaux

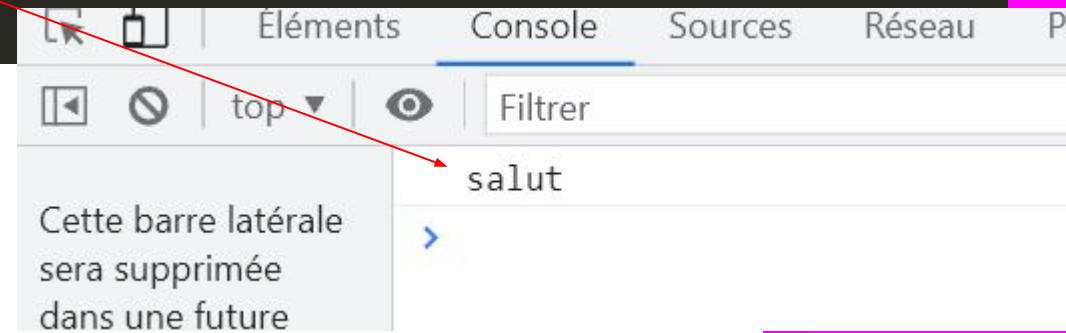
```
const notes = [10, 5, 14, 19, 'coucou',[12,13,'salut']];
console.log(notes);
```

```
[10, 5, 14, 19, "coucou", Array(3)]  
0: 10  
1: 5  
2: 14  
3: 19  
4: "coucou"  
5: Array(3)  
0: 12  
1: 13  
2: "salut"  
length: 3  
[[Prototype]]: Array(0)  
length: 6  
[[Prototype]]: Array(0)
```

Dans un tableau indicé (ou autres), il peut y avoir des chiffres, mots et même d'autres tableau...A votre avis, comment je fais pour afficher "salut" ?

## Algorithmie - Les Tableaux

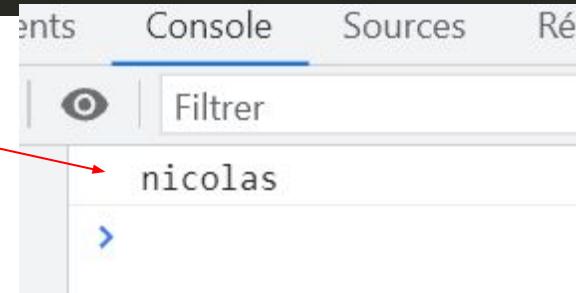
```
const notes = [10, 5, 14, 19, 'coucou',[12,13,'salut']];
console.log(notes[5][2]);
```



Pour sélectionner mon tableau qui est dans mon tableau “notes”, je dois sélectionner son index, qui est 5. Une fois sélectionné, je dois trouver l’index de “salut” dans ce tableau, qui est 3.

# Algorithmie - Les Tableaux

```
const eleve = [ {'nom': 'nicolas', 'prenom': 'dupont', 'moyenne': 15, 'notes': [15, 10, 15]}];  
console.log(eleve[0].nom);
```



Voici un tableau associatif (avec des colonnes). Ce tableau n'a qu'une ligne. En JS, une ligne dans un tableau est entre accolades.  
Pour sélectionner une valeur, il faut cette syntaxe :  
**tableau[numero\_de\_ligne].nom\_de\_la\_colonne**

# Algorithmie - Les Tableaux

```
const eleve = [ { 'nom': 'nicolas', 'prenom': 'dupont', 'moyenne': 15, 'notes': [15, 10, 15] } ];
```

Ayons le bon vocabulaire..... Clés (c'est les colonnes), Valeurs

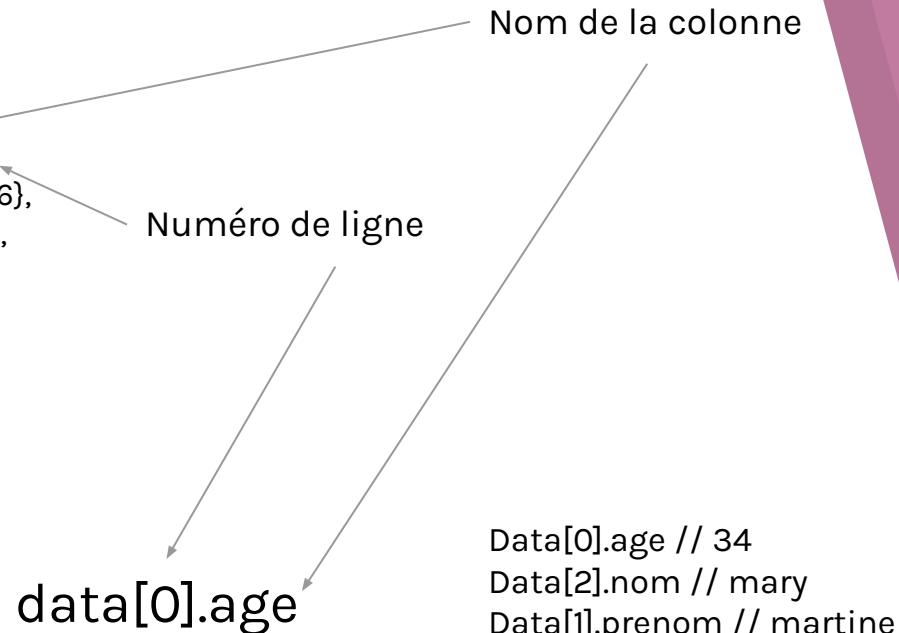
# Algorithmie - Les Tableaux

```
const data = [  
    {nom:'dupont', prenom: 'jean', age: 34}, ← Lignes du tableau  
    {nom:'pouley', prenom: 'martine', age: 56}, ←  
    {nom:'mary', prenom: 'thomas', age: 22}, ←  
    {nom:'dupuis', prenom: 'ines', age: 30} ←  
]; ← Nom des colonnes (clés)  
“Data” est un tableau associatif (avec des colonnes en fait)
```

Autre exemple pour expliquer la structure d'un tableau associatif

# Algorithmie - Les Tableaux

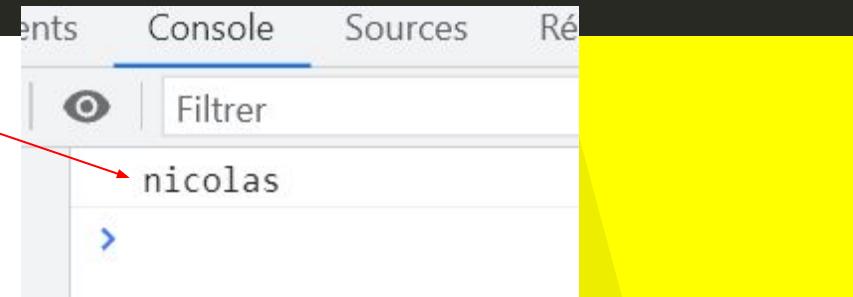
```
const data = [  
    {nom:'dupont', prenom: 'jean', age: 34},  
    {nom:'pouley', prenom: 'martine', age: 56},  
    {nom:'mary', prenom: 'thomas', age: 22},  
    {nom:'dupuis', prenom: 'ines', age: 30},  
];
```



Autre exemple pour expliquer la syntaxe de sélection d'une valeur dans un tableau associatif

# Algorithmie - Les Tableaux

```
const eleve = [{nom: 'nicolas', prenom:'dupont', moyenne:15, notes:[15, 10, 15]}];  
console.log(eleve[0]['nom']);
```



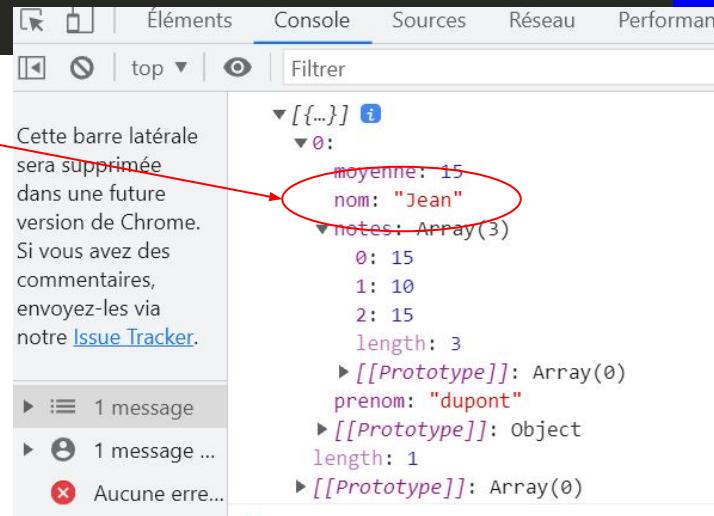
Cette syntaxe est aussi possible :

**Tableau[numero\_de\_ligne][nom\_de\_la\_colonne]**

J'ai un faible pour l'autre syntaxe (avec le point)

# Algorithmie - Les Tableaux

```
const eleve = [{nom: 'nicolas', prenom:'dupont','moyenne':15,'notes':[15, 10, 15]}];  
eleve[0].nom = 'Jean';  
console.log(eleve);
```



Il est possible de modifier la valeur d'un tableau en assignant une nouvelle valeur à la colonne souhaitée.  
Ci-dessus, j'assigne “Jean” à la clé (colonne) “nom”

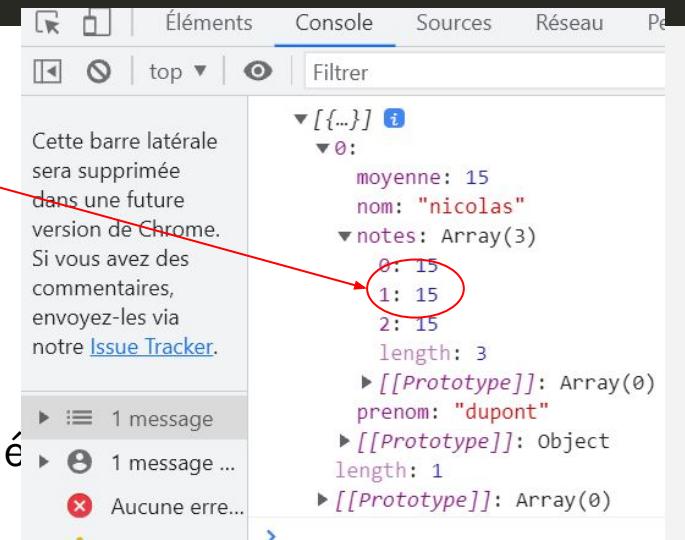
# Algorithmie - Les Tableaux

```
const eleve = [ {'nom': 'nicolas', 'prenom': 'dupont', 'moyenne':15, 'notes':[15, 10, 15]}];
```

Petit exercice, comment je fais pour modifier ma note de 10 en 15 ???

# Algorithmie - Les Tableaux

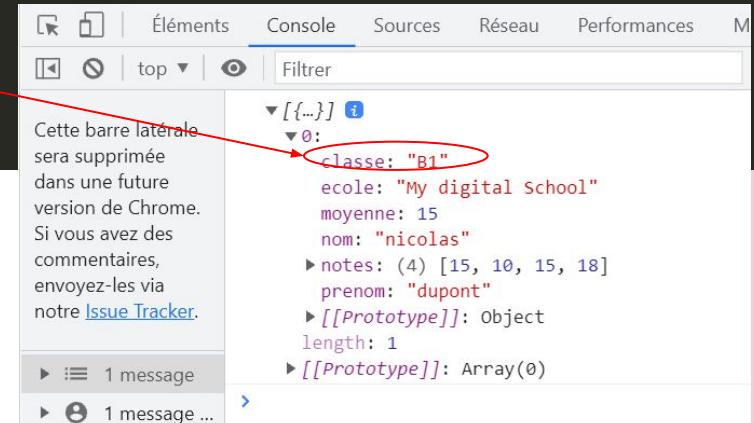
```
const eleve = [{nom: 'nicolas', prenom:'dupont','moyenne':15,'notes':[15, 10, 15]}];  
eleve[0].notes[1] = 15;  
console.log(eleve);
```



- 1/ Je sélectionne la note eleve[0].notes[1] (tableau[numé [index\_de\_la\_note]])
- 2/ Je lui assigne une nouvelle valeur : eleve[0].notes[1] = 15
- 3/ Voilà....

# Algorithmie - Les Tableaux

```
const eleve = [ { 'nom': 'nicolas', 'prenom': 'dupont', 'moyenne': 15, 'notes': [15, 10, 15] } ];  
eleve[0].classe = "B1";  
eleve[0].ecole = "My digital School";  
eleve[0].notes[3] = 18;  
console.log(eleve);
```



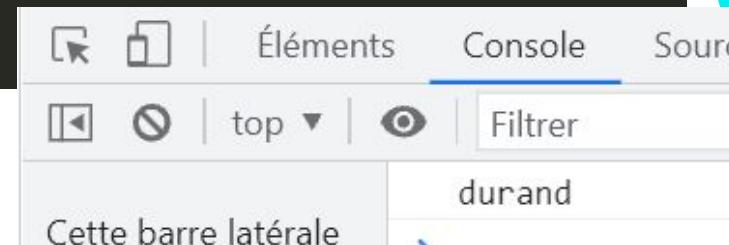
Si je peux modifier des valeurs à la volé, je peux aussi ajouter de nouvelles clés....

eleve[0].classe = "B1"; (tableau, ligne, nouvelle clé, valeur)

# Algorithmie - Les Tableaux

```
const eleve = [  
    {'nom': 'nicolas', 'prenom': 'dupont', 'moyenne':15, 'notes':[15, 10, 15]},  
    {'nom': 'sophie', 'prenom': 'durand', 'moyenne':18, 'notes':[20, 18, 14]}  
];
```

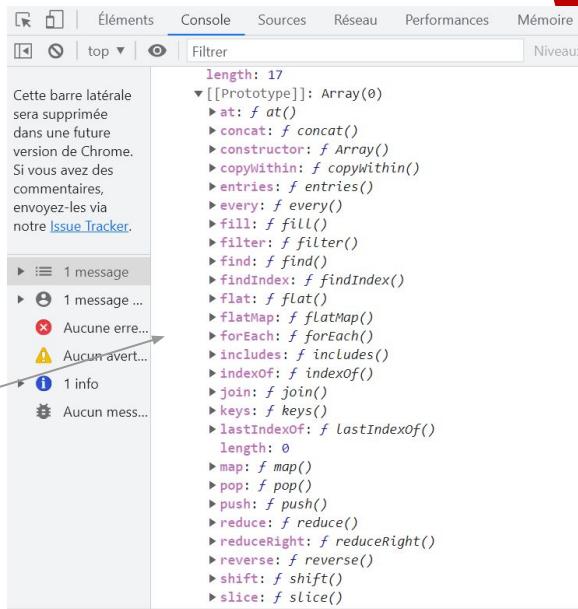
```
console.log(eleve[1].prenom);
```



Du coup, si j'ai 2 lignes (ou plus), j'ai juste à passer l'index de la ligne que je veux puis la clé pour afficher sa valeur.

# Algorithmie - Les Tableaux

```
(17) [{} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} ] i  
▶ 0: {prenom: 'nicolas', nom: 'dupont'}  
▶ 1: {prenom: 'sophie', nom: 'durand'}  
▶ 2: {prenom: 'clémence', nom: 'dupuis'}  
▶ 3: {prenom: 'emmanuel', nom: 'macron'}  
▶ 4: {prenom: 'elon', nom: 'musk'}  
▶ 5: {prenom: 'johnson', nom: 'boris'}  
▶ 6: {prenom: 'marlène', nom: 'shiappa'}  
▶ 7: {prenom: 'marcel', nom: 'jeanno'}  
▶ 8: {prenom: 'joe', nom: 'goldberg'}  
▶ 9: {prenom: 'han', nom: 'Mi-nyeo'}  
▶ 10: {prenom: 'el', nom: 'professor'}  
▶ 11: {prenom: 'cho', nom: 'Sang-Woo'}  
▶ 12: {prenom: 'oh', nom: 'Il-nam'}  
▶ 13: {prenom: 'john', nom: 'snow'}  
▶ 14: {prenom: 'harry', nom: 'potter'}  
▶ 15: {prenom: 'brahim', nom: 'bouhlel'}  
▶ 16: {prenom: 'stéphane', nom: 'chicheman'}  
length: 17  
▶ [[Prototype]]: Array(0)
```

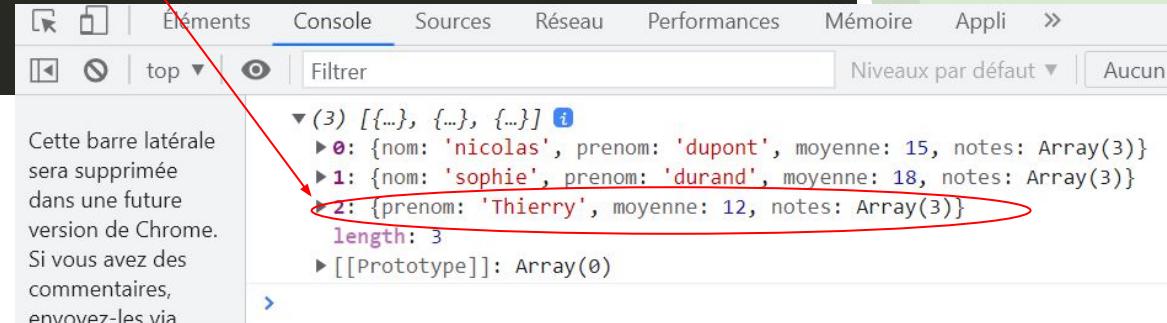


Lorsque je fais un `console.log` d'un tableau, en plus de m'afficher le détail de ce dernier, à la fin, on peut voir écrit dans la console "prototype"....JS essaierai-t-il de me faire passer un message ??

Et bien oui ! si on clique dessus, on voit que JS nous donne plusieurs méthodes (entre autres), pour manipuler les tableaux... Voyons en quelques unes

# Algorithmie - Les Tableaux

```
const eleve = [  
    {'nom': 'nicolas', 'prenom': 'dupont', 'moyenne': 15, 'notes': [15, 10, 15]},  
    {'nom': 'sophie', 'prenom': 'durand', 'moyenne': 18, 'notes': [20, 18, 14]}  
];  
  
eleve.push( {'nom': 'Henry', 'prenom': 'Thierry', 'moyenne': 12, 'notes': [12, 12, 12]} );  
  
console.log(eleve);
```



## .push()

La méthode `.push()` va permettre d'ajouter une ligne **à la fin** d'un tableau et nous retourner la taille du tableau après ajout. Voici la syntaxe : `tableau.push( {clé : valeur} )`. Attention, dans le cas d'un tableau associatif, il faut autant de clé (colonne) que dans les lignes déjà présentes. Sinon vous aurez des problèmes plus tard.

# Algorithmie - Les Tableaux

```
const eleve = [  
    {'nom': 'nicolas', 'prenom': 'dupont', 'moyenne': 15, 'notes': [15, 10, 15]},  
    {'nom': 'sophie', 'prenom': 'durand', 'moyenne': 18, 'notes': [20, 18, 14]}  
];  
  
let ligneSupprimee = eleve.pop();  
  
console.log(ligneSupprimee);  
console.log(`-----`);  
console.log(eleve);
```

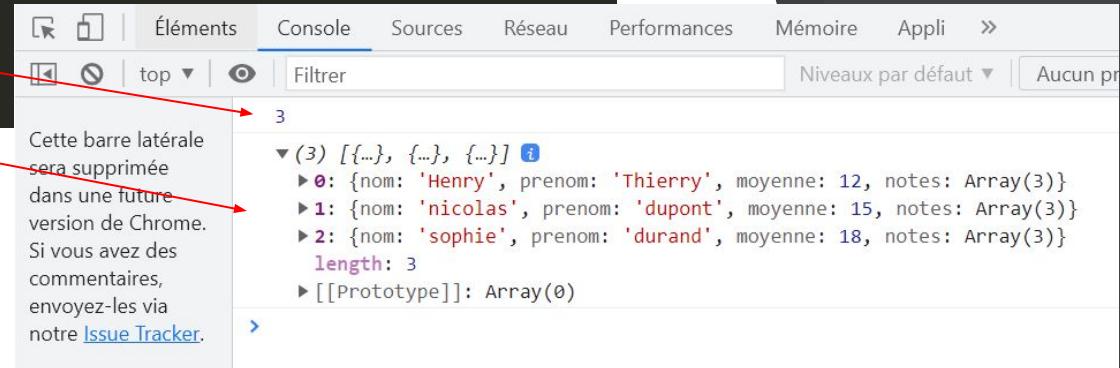
## .pop()

La méthode `.pop()` va supprimer le dernier élément du tableau et nous retourner l'élément qu'elle vient de supprimer (si souhaité). Dans l'exemple au-dessus, on constate que ma ligne supprimée est bien celle de Sophie et qu'il reste juste "Nicolas" dans mon tableau.  
Syntaxe : `tableau.pop()`



# Algorithmie - Les Tableaux

```
const eleve = [  
    {'nom': 'nicolas', 'prenom': 'dupont', 'moyenne': 15, 'notes': [15, 10, 15]},  
    {'nom': 'sophie', 'prenom': 'durand', 'moyenne': 18, 'notes': [20, 18, 14]}  
];  
  
let tailleDuTableau = eleve.unshift({'nom': 'Henry', 'prenom': 'Thierry', 'moyenne': 12, 'notes': [12, 12, 12]});  
  
console.log(tailleDuTableau);  
  
console.log(eleve);
```



## .unshift()

La méthode `.unshift()` à la même fonction que `.push()` sauf qu'elle va ajouter une ligne (ou élément) au début du tableau et nous retourner la nouvelle taille du tableau après l'ajout.  
Syntaxe : `tableau.unshift({clé : valeur})`

# Algorithmie - Les Tableaux

```
const eleve = [  
    {'nom': 'nicolas', 'prenom': 'dupont', 'moyenne': 15, 'notes': [15, 10, 15]},  
    {'nom': 'sophie', 'prenom': 'durand', 'moyenne': 18, 'notes': [20, 18, 14]}  
];  
  
let ligneSupprimee = eleve.shift();  
  
console.log(ligneSupprimee);  
  
console.log(eleve);
```



## .shift()

La méthode `.shift()` à la même fonction que `.pop()`, sauf qu'elle va supprimer la première ligne (ou élément) d'un tableau et nous retourner la ligne (ou l'élément) supprimé).

Syntaxe : tableau.shift()

# Algorithmie - Les Tableaux

```
const eleves = [  
    {'prenom': 'nicolas', 'nom': 'dupont'},  
    {'prenom': 'sophie', 'nom': 'durand'},  
    {'prenom': 'clémence', 'nom': 'dupuis'},  
    {'prenom': 'emmanuel', 'nom': 'macron'},  
    {'prenom': 'elon', 'nom': 'musk'},  
    {'prenom': 'johnson', 'nom': 'boris'},  
    {'prenom': 'marlène', 'nom': 'shiappa'},  
    {'prenom': 'marcel', 'nom': 'jeanno'},  
    {'prenom': 'joe', 'nom': 'goldberg'},  
    {'prenom': 'han', 'nom': 'Mi-nyeo'},  
    {'prenom': 'el', 'nom': 'professor'},  
    {'prenom': 'cho', 'nom': 'Sang-Woo'},  
    {'prenom': 'oh', 'nom': 'Il-nam'},  
    {'prenom': 'john', 'nom': 'snow'},  
    {'prenom': 'harry', 'nom': 'potter'},  
    {'prenom': 'brahim', 'nom': 'bouhlel'},  
    {'prenom': 'stéphane', 'nom': 'chicheman'}  
];  
  
let elementsSupprimés = eleves.splice(3);  
console.log(elementsSupprimés);  
console.log(eleves);
```

Cette barre latérale sera supprimée dans une future version de Chrome. Si vous avez des commentaires, envoyez-les via notre Issue Tracker.

▶ 2 messages  
▶ 2 messages ...  
✖ Aucune erre...  
⚠ Aucun avert...  
▶ 2 infos  
✖ Aucun mess...

▼ (14) [{...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}] ⓘ  
▶ 0: {prenom: 'emmanuel', nom: 'macron'}  
▶ 1: {prenom: 'elon', nom: 'musk'}  
▶ 2: {prenom: 'johnson', nom: 'boris'}  
▶ 3: {prenom: 'marlène', nom: 'shiappa'}  
▶ 4: {prenom: 'marcel', nom: 'jeanno'}  
▶ 5: {prenom: 'joe', nom: 'goldberg'}  
▶ 6: {prenom: 'han', nom: 'Mi-nyeo'}  
▶ 7: {prenom: 'el', nom: 'professor'}  
▶ 8: {prenom: 'cho', nom: 'Sang-Woo'}  
▶ 9: {prenom: 'oh', nom: 'Il-nam'}  
▶ 10: {prenom: 'john', nom: 'snow'}  
▶ 11: {prenom: 'harry', nom: 'potter'}  
▶ 12: {prenom: 'brahim', nom: 'bouhlel'}  
▶ 13: {prenom: 'stéphane', nom: 'chicheman'}  
length: 14  
[[Prototype]]: Array(0)  
  
▼ (3) [{...}, {...}, {...}] ⓘ  
▶ 0: {prenom: 'nicolas', nom: 'dupont'}  
▶ 1: {prenom: 'sophie', nom: 'durand'}  
▶ 2: {prenom: 'clémence', nom: 'dupuis'}  
length: 3  
[[Prototype]]: Array(0)

## .splice()

La méthode `.splice()` peut modifier, remplacer, ajouter des éléments n'importe où dans le tableau.

Cette méthode va également retourner un tableau contenant les éléments supprimés.

Syntaxe : `tableau.splice(indexDeDepart, nbDeSuppression, elementAajouter)`

# Algorithmie - Les Tableaux

```
▼ (15) [{} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} ]  
  i  
► 0: {prenom: 'nicolas', nom: 'dupont'}  
► 1: {prenom: 'super', nom: 'woman'}  
► 2: {prenom: 'elon', nom: 'musk'}  
► 3: {prenom: 'johnson', nom: 'boris'}  
► 4: {prenom: 'marlène', nom: 'shiappa'}  
► 5: {prenom: 'marcel', nom: 'jeanno'}  
► 6: {prenom: 'joe', nom: 'goldberg'}  
► 7: {prenom: 'han', nom: 'Mi-nyeo'}  
► 8: {prenom: 'el', nom: 'professor'}  
► 9: {prenom: 'cho', nom: 'Sang-Woo'}  
► 10: {prenom: 'oh', nom: 'Il-nam'}  
► 11: {prenom: 'john', nom: 'snow'}  
► 12: {prenom: 'harry', nom: 'potter'}  
► 13: {prenom: 'brahim', nom: 'bouhlel'}  
► 14: {prenom: 'stéphane', nom: 'chicheman'}  
length: 15  
► [[Prototype]]: Array(0)
```

## .splice()

Exemple :

```
eleves.splice(1,3,{'prenom': 'super', 'nom': 'woman'});
```

Je lui demande “Dès la deuxième ligne (1), tu vas m'en supprimer 3 et tu vas m'ajouter une ligne ({'prenom': 'super', 'nom': 'woman'})

```
const eleves = [  
    {prenom: 'nicolas', nom: 'dupont'},  
    {prenom: 'sophie', nom: 'durand'},  
    {prenom: 'clémence', nom: 'dupuis'},  
    {prenom: 'emmanuel', nom: 'macron'},  
    {prenom: 'elon', nom: 'musk'},  
    {prenom: 'johnson', nom: 'boris'},  
    {prenom: 'marlène', nom: 'shiappa'},  
    {prenom: 'marcel', nom: 'jeanno'},  
    {prenom: 'joe', nom: 'goldberg'},  
    {prenom: 'han', nom: 'Mi-nyeo'},  
    {prenom: 'el', nom: 'professor'},  
    {prenom: 'cho', nom: 'Sang-Woo'},  
    {prenom: 'oh', nom: 'Il-nam'},  
    {prenom: 'john', nom: 'snow'},  
    {prenom: 'harry', nom: 'potter'},  
    {prenom: 'brahim', nom: 'bouhlel'},  
    {prenom: 'stéphane', nom: 'chicheman'},  
];  
  
eleves.splice(1,3,{ 'prenom': 'super', 'nom': 'woman' });  
console.log(eleves);
```

# Algorithmie - Les Tableaux

```
▼ (18) [{} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} , {} ] ⓘ
▶ 0: {prenom: 'nicolas', nom: 'dupont'}
▶ 1: {prenom: 'sophie', nom: 'durand'}
▶ 2: {prenom: 'super', nom: 'woman'} highlighted
▶ 3: {prenom: 'clemence', nom: 'dupuis'}
▶ 4: {prenom: 'emmanuel', nom: 'macron'}
▶ 5: {prenom: 'elon', nom: 'musk'}
▶ 6: {prenom: 'johnson', nom: 'boris'}
▶ 7: {prenom: 'marlene', nom: 'shiappa'}
▶ 8: {prenom: 'marcel', nom: 'jeanno'}
▶ 9: {prenom: 'joe', nom: 'goldberg'}
▶ 10: {prenom: 'han', nom: 'Mi-nyeo'}
▶ 11: {prenom: 'el', nom: 'professor'}
▶ 12: {prenom: 'cho', nom: 'Sang-Woo'}
▶ 13: {prenom: 'oh', nom: 'Il-nam'}
▶ 14: {prenom: 'john', nom: 'snow'}
▶ 15: {prenom: 'harry', nom: 'potter'}
▶ 16: {prenom: 'brahim', nom: 'bouhlel'}
▶ 17: {prenom: 'stéphane', nom: 'chicheman'}
  length: 18
▶ [[Prototype]]: Array(0)
```

## .splice()

Exemple :

```
eleves.splice(2,0,{prenom: 'super', 'nom':'woman'});
```

Je lui demande “Dès la troisième ligne (2), tu ne vas pas m'en supprimer (0) et tu vas m'ajouter une ligne ({'prenom': 'super', 'nom':'woman'})

```
const eleves = [
    {'prenom': 'nicolas', 'nom':'dupont'},
    {'prenom': 'sophie', 'nom':'durand'},
    {'prenom': 'clemence', 'nom':'dupuis'},
    {'prenom': 'emmanuel', 'nom':'macron'},
    {'prenom': 'elon', 'nom':'musk'},
    {'prenom': 'johnson', 'nom':'boris'},
    {'prenom': 'marlene', 'nom':'shiappa'},
    {'prenom': 'marcel', 'nom':'jeanno'},
    {'prenom': 'joe', 'nom':'goldberg'},
    {'prenom': 'han', 'nom':'Mi-nyeo'},
    {'prenom': 'el', 'nom':'professor'},
    {'prenom': 'cho', 'nom':'Sang-Woo'},
    {'prenom': 'oh', 'nom':'Il-nam'},
    {'prenom': 'john', 'nom':'snow'},
    {'prenom': 'harry', 'nom':'potter'},
    {'prenom': 'brahim', 'nom':'bouhlel'},
    {'prenom': 'stéphane', 'nom':'chicheman'},
];
eleves.splice(2,0,{prenom: 'super', 'nom':'woman'});
console.log(eleves);
```

# Algorithmie - Les Tableaux

Cette barre latérale sera supprimée dans une future version de Chrome. Si vous avez des commentaires, envoyez-les via notre [Issue Tracker](#).

▶ 1 message

▶ 1 message ...

```
▼ (10) [{} , {} , {} , {} , {} , {} , {} , {} , {} , {}]
  ▶ 0: {prenom: 'marcel', nom: 'jeanno'}
  ▶ 1: {prenom: 'joe', nom: 'goldberg'}
  ▶ 2: {prenom: 'han', nom: 'Mi-nyeo'}
  ▶ 3: {prenom: 'el', nom: 'professor'}
  ▶ 4: {prenom: 'cho', nom: 'Sang-Woo'}
  ▶ 5: {prenom: 'oh', nom: 'Il-nam'}
  ▶ 6: {prenom: 'john', nom: 'snow'}
  ▶ 7: {prenom: 'harry', nom: 'potter'}
  ▶ 8: {prenom: 'brahim', nom: 'bouhlel'}
  ▶ 9: {prenom: 'stéphane', nom: 'chicheman'}
    length: 10
  ▶ [[Prototype]]: Array(0)
```

## .splice()

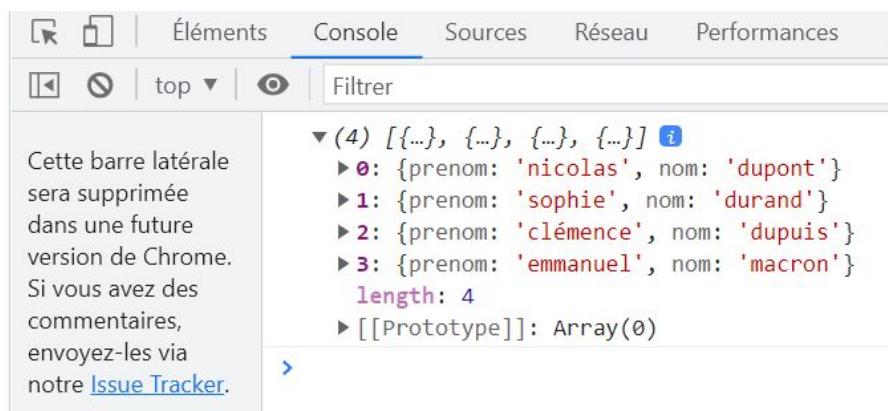
Exemple :

~~eleves.splice(0,7);~~

Je lui demande “Dès la première ligne (0), tu vas m'en supprimer 7 et tu vas rien m'ajouter.

```
const eleves = [
  {'prenom': 'nicolas', 'nom': 'dupont'},
  {'prenom': 'sophie', 'nom': 'durand'},
  {'prenom': 'clémence', 'nom': 'dupuis'},
  {'prenom': 'emmanuel', 'nom': 'macron'},
  {'prenom': 'elon', 'nom': 'musk'},
  {'prenom': 'johnson', 'nom': 'boris'},
  {'prenom': 'marlène', 'nom': 'shiappa'},
  {'prenom': 'marcel', 'nom': 'jeanno'},
  {'prenom': 'joe', 'nom': 'goldberg'},
  {'prenom': 'han', 'nom': 'Mi-nyeo'},
  {'prenom': 'el', 'nom': 'professor'},
  {'prenom': 'cho', 'nom': 'Sang-Woo'},
  {'prenom': 'oh', 'nom': 'Il-nam'},
  {'prenom': 'john', 'nom': 'snow'},
  {'prenom': 'harry', 'nom': 'potter'},
  {'prenom': 'brahim', 'nom': 'bouhlel'},
  {'prenom': 'stéphane', 'nom': 'chicheman'},
];
eleves.splice(0,7);
console.log(eleves);
```

# Algorithmie - Les Tableaux



## .splice()

Exemple :

```
eleves.splice(4);
```

Je lui demande “Dès la cinquième ligne (5), Tu me supprimes tout !

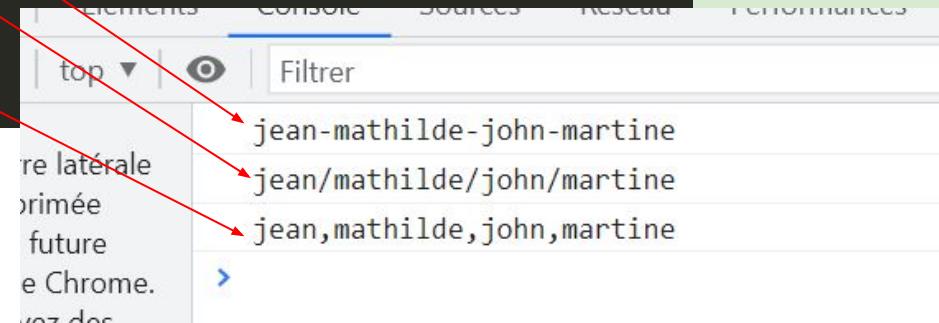
```
const eleves = [
    {'prenom': 'nicolas', 'nom': 'dupont'},
    {'prenom': 'sophie', 'nom': 'durand'},
    {'prenom': 'clémence', 'nom': 'dupuis'},
    {'prenom': 'emmanuel', 'nom': 'macron'},
    {'prenom': 'elon', 'nom': 'musk'},
    {'prenom': 'johnson', 'nom': 'boris'},
    {'prenom': 'marlène', 'nom': 'shiappa'},
    {'prenom': 'marcel', 'nom': 'jeanno'},
    {'prenom': 'joe', 'nom': 'goldberg'},
    {'prenom': 'han', 'nom': 'Mi-nyeo'},
    {'prenom': 'el', 'nom': 'professor'},
    {'prenom': 'cho', 'nom': 'Sang-Woo'},
    {'prenom': 'oh', 'nom': 'Il-nam'},
    {'prenom': 'john', 'nom': 'snow'},
    {'prenom': 'harry', 'nom': 'potter'},
    {'prenom': 'brahim', 'nom': 'bouhlel'},
    {'prenom': 'stéphane', 'nom': 'chicheman'},
];
```

```
eleves.splice(4);
```

```
console.log(eleves);
```

## Algorithmie - Les Tableaux

```
const noms = ['jean', 'mathilde', 'john', 'martine'];
console.log(noms.join('-'));
console.log(noms.join('/'));
console.log(noms.join());
```



### .join()

La méthode `join()` retourne une chaîne de caractères créée en concaténant les différentes valeurs d'un tableau. Le séparateur utilisé par défaut sera la virgule mais nous allons également pouvoir passer le séparateur de notre choix en **argument** de `join()`.  
Syntaxe : `tableau.join(séparateurSouhaité)`;

# Algorithmie - Les Tableaux

## .concat()

```
const eleves = [
    {'prenom': 'nicolas', 'nom': 'dupont'},
    {'prenom': 'sophie', 'nom': 'durand'},
    {'prenom': 'clémente', 'nom': 'dupuis'},
    {'prenom': 'emmanuel', 'nom': 'macron'},
    {'prenom': 'elon', 'nom': 'musk'},
    {'prenom': 'johnson', 'nom': 'boris'},
    {'prenom': 'marlène', 'nom': 'shiappa'},
    {'prenom': 'marcel', 'nom': 'jeanno'},
    {'prenom': 'joe', 'nom': 'goldberg'},
    {'prenom': 'han', 'nom': 'Mi-nyeo'},
    {'prenom': 'el', 'nom': 'professor'},
    {'prenom': 'cho', 'nom': 'Sang-Woo'},
    {'prenom': 'oh', 'nom': 'Il-nam'},
    {'prenom': 'john', 'nom': 'snow'},
    {'prenom': 'harry', 'nom': 'potter'},
    {'prenom': 'brahim', 'nom': 'bouhlel'},
    {'prenom': 'stéphane', 'nom': 'chicheman'},
];

const noms = ['jean', 'mathilde', 'john', 'martine'];

console.log(noms.concat(eleves));
```

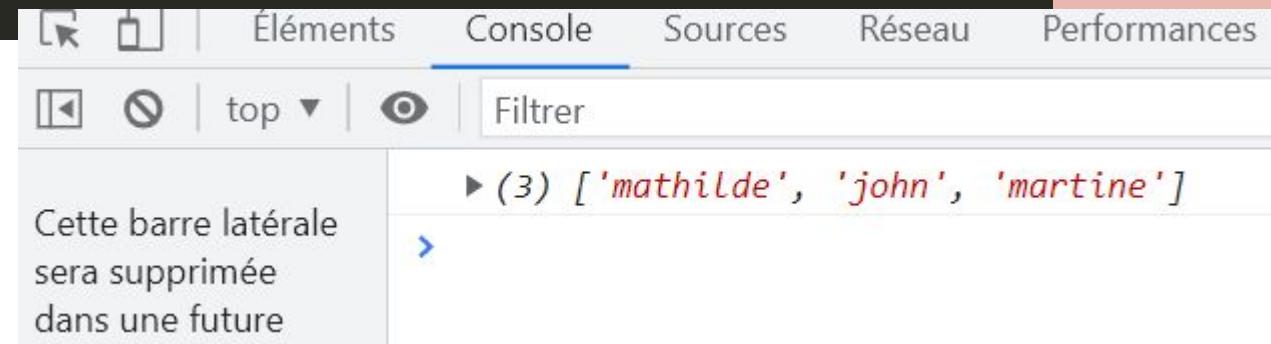
```
(21) [ 'jean', 'mathilde', 'john', 'martine', {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {} ] ⓘ
0: "jean"
1: "mathilde"
2: "john"
3: "martine"
▶ 4: {prenom: "nicolas", nom: "dupont"}
▶ 5: {prenom: "sophie", nom: "durand"}
▶ 6: {prenom: "clémente", nom: "dupuis"}
▶ 7: {prenom: "emmanuel", nom: "macron"}
▶ 8: {prenom: "elon", nom: "musk"}
▶ 9: {prenom: "johnson", nom: "boris"}
▶ 10: {prenom: "marlène", nom: "shiappa"}
▶ 11: {prenom: "marcel", nom: "jeanno"}
▶ 12: {prenom: "joe", nom: "goldberg"}
▶ 13: {prenom: "han", nom: "Mi-nyeo"}
▶ 14: {prenom: "el", nom: "professor"}
▶ 15: {prenom: "cho", nom: "Sang-Woo"}
▶ 16: {prenom: "oh", nom: "Il-nam"}
▶ 17: {prenom: "john", nom: "snow"}
▶ 18: {prenom: "harry", nom: "potter"}
▶ 19: {prenom: "brahim", nom: "bouhlel"}
▶ 20: {prenom: "stéphane", nom: "chicheman"}
length: 21
▶ [[Prototype]]: Array(0)
```

La méthode concat() va nous permettre de fusionner différents tableaux entre eux pour en créer un nouveau qu'elle va renvoyer. On peut fusionner plusieurs tableaux à la fois...  
Syntaxe : tableau.concat(tableauAfusionner)

# Algorithmie - Les Tableaux

```
const noms = ['jean', 'mathilde', 'john', 'martine','kevin','julie'];

console.log(noms.slice(1,4));
```



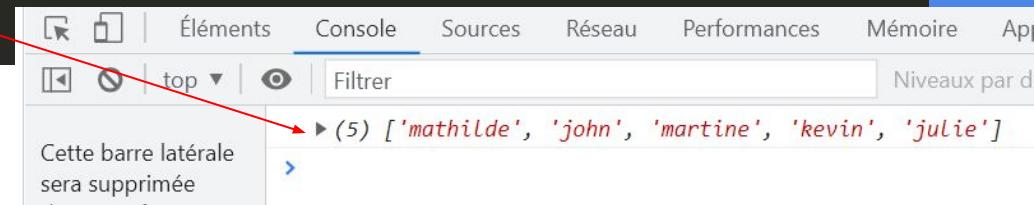
## .slice()

La méthode slice() va extraire des éléments d'un tableau sous forme...de tableau.

Syntaxe : tableau.slice(indexDeDepart, indexDeFin)

Dans l'exemple ci-dessus, on voit que j'extrais à partir du deuxième nom (inclus) jusqu'au cinquième (exclus).

```
const noms = ['jean', 'mathilde', 'john', 'martine', 'kevin', 'julie'];  
console.log(noms.slice(1));
```



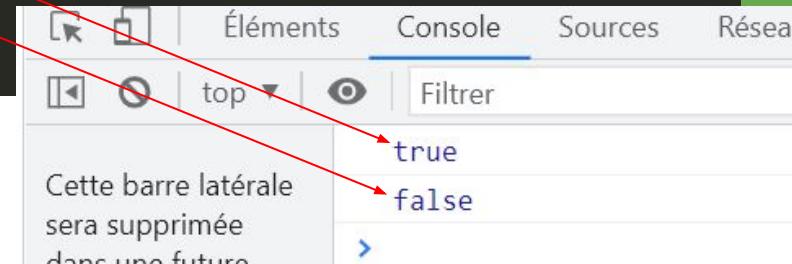
## .slice()

Si je ne passe qu'un argument (index de départ) et pas le deuxième (index de fin), la méthode va extraire tout à partir de l'index de départ.

# Algorithmie - Les Tableaux

```
const noms = ['jean', 'mathilde', 'john', 'martine', 'kevin', 'julie'];

console.log(noms.includes('john'));
console.log(noms.includes('julien'));
```



## .includes()

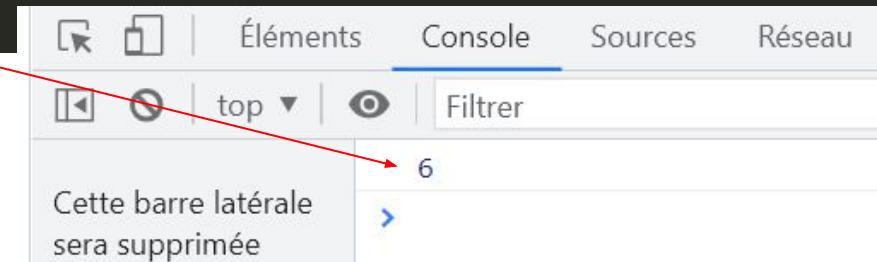
La méthode .includes() sert à vérifier si un élément en particulier est présent dans le tableau... Il retourne un booléen (true si il est dans le tableau, false si il n'y est pas)

Syntaxe : tableau.includes('ceQueVousCherchezDansLeTableau');

La méthode "indexOf()" est similaire....

## Algorithmie - Les Tableaux

```
const noms = ['jean', 'mathilde', 'john', 'martine', 'kevin', 'julie'];  
console.log(noms.length);
```



### .length

.length n'est pas une méthode mais une propriété. Elle retourne la taille d'un tableau (son nombre de ligne)

Syntaxe : tableau.length

# **Algorithmie, Les Boucles**

## Algorithmie - Les Boucles

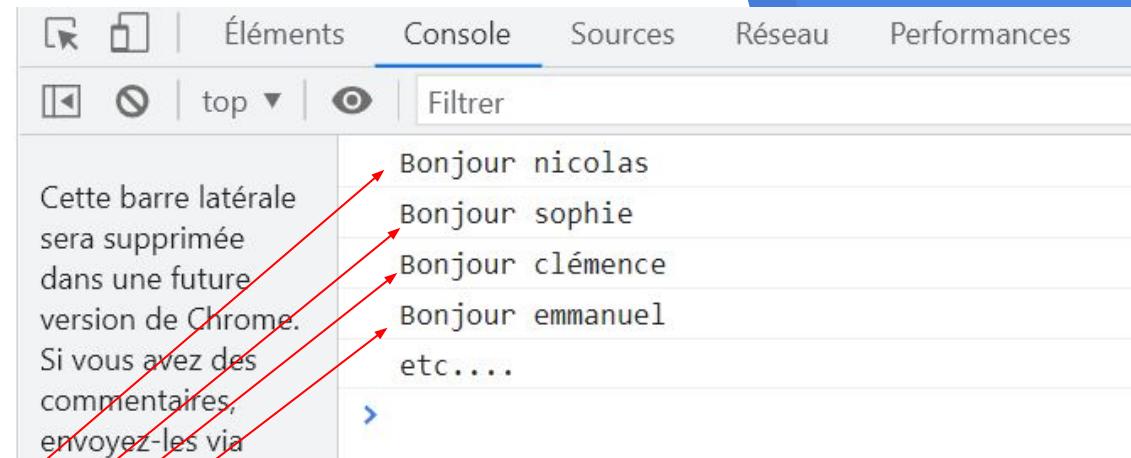
Les boucles vont nous permettre d'exécuter plusieurs fois un bloc de code ou parcourir des tableaux, c'est-à-dire d'exécuter un code « en boucle » tant qu'une condition donnée est vérifiée et donc ainsi nous faire gagner beaucoup de temps dans l'écriture de nos scripts.

Lorsqu'on code, on va en effet souvent devoir exécuter plusieurs fois un même code. Utiliser une boucle nous permet de n'écrire le code qu'on doit exécuter plusieurs fois qu'une seule fois.

# Algorithmie - Les Boucles

```
const eleves = [
    {'prenom': 'nicolas', 'nom':'dupont'},
    {'prenom': 'sophie', 'nom':'durand'},
    {'prenom': 'clémence', 'nom':'dupuis'},
    {'prenom': 'emmanuel', 'nom':'macron'},
    {'prenom': 'elon', 'nom':'musk'},
    {'prenom': 'johnson', 'nom':'boris'},
    {'prenom': 'marlène', 'nom':'shiappa'},
    {'prenom': 'marcel', 'nom':'jeanno'},
    {'prenom': 'joe', 'nom':'goldberg'},
    {'prenom': 'han', 'nom':'Mi-nyeo'},
    {'prenom': 'el', 'nom':'professor'},
    {'prenom': 'cho', 'nom':'Sang-Woo'},
    {'prenom': 'oh', 'nom':'Il-nam'},
    {'prenom': 'john', 'nom':'snow'},
    {'prenom': 'harry', 'nom':'potter'},
    {'prenom': 'brahim', 'nom':'bouhlel'},
    {'prenom': 'stéphane', 'nom':'chicheman'},
];

console.log(`Bonjour ${eleves[0].prenom}`);
console.log(`Bonjour ${eleves[1].prenom}`);
console.log(`Bonjour ${eleves[2].prenom}`);
console.log(`Bonjour ${eleves[3].prenom}`);
console.log(`etc....`);
```

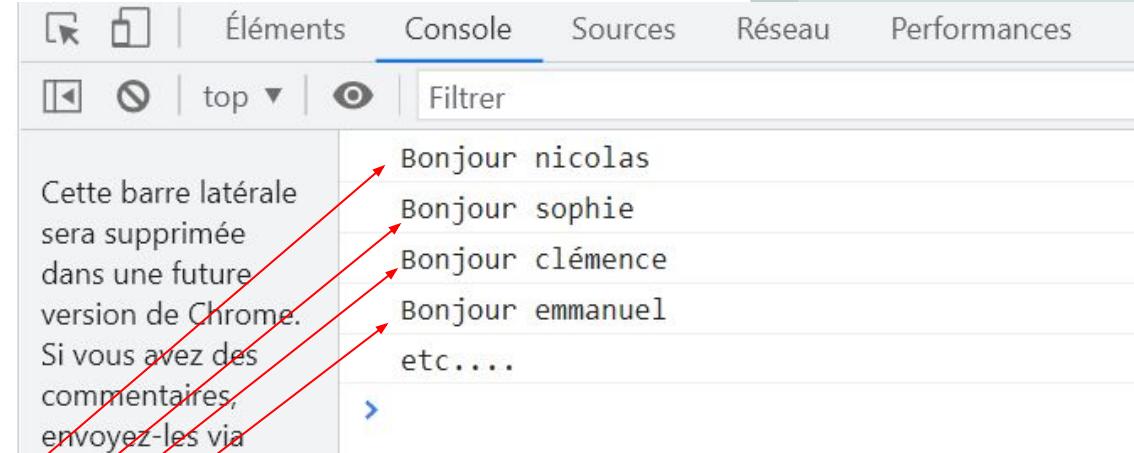


Imaginons que nous souhaitons saluer tous les élèves.... Faire un `console.log()` et les appeler un par un serait long....

# Algorithmie - Les Boucles

```
const eleves = [
    {'prenom': 'nicolas', 'nom':'dupont'},
    {'prenom': 'sophie', 'nom':'durand'},
    {'prenom': 'clémence', 'nom':'dupuis'},
    {'prenom': 'emmanuel', 'nom':'macron'},
    {'prenom': 'elon', 'nom':'musk'},
    {'prenom': 'johnson', 'nom':'boris'},
    {'prenom': 'marlène', 'nom':'shiappa'},
    {'prenom': 'marcel', 'nom':'jeanno'},
    {'prenom': 'joe', 'nom':'goldberg'},
    {'prenom': 'han', 'nom':'Mi-nyeo'},
    {'prenom': 'el', 'nom':'professor'},
    {'prenom': 'cho', 'nom':'Sang-Woo'},
    {'prenom': 'oh', 'nom':'Il-nam'},
    {'prenom': 'john', 'nom':'snow'},
    {'prenom': 'harry', 'nom':'potter'},
    {'prenom': 'brahim', 'nom':'bouhlel'},
    {'prenom': 'stéphane', 'nom':'chicheman'},
];

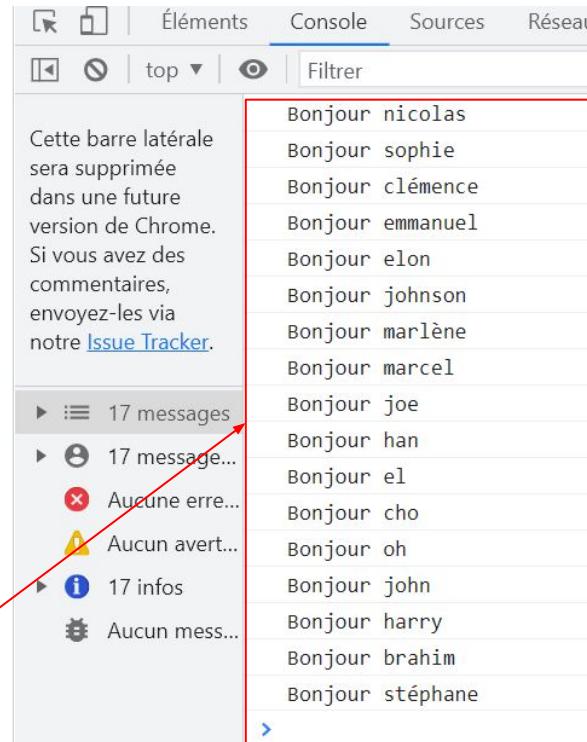
console.log(`Bonjour ${eleves[0].prenom}`);
console.log(`Bonjour ${eleves[1].prenom}`);
console.log(`Bonjour ${eleves[2].prenom}`);
console.log(`Bonjour ${eleves[3].prenom}`);
console.log(`etc....`);
```



Et en admettant qu'on les fasse tous, si une personne ajoute ou supprime des élèves, nous devrons rajouter ou supprimer manuellement (donc toujours vérifier le tableau).

# Algorithmie - Les Boucles

```
const eleves = [
    {'prenom': 'nicolas', 'nom': 'dupont'},
    {'prenom': 'sophie', 'nom': 'durand'},
    {'prenom': 'clémence', 'nom': 'dupuis'},
    {'prenom': 'emmanuel', 'nom': 'macron'},
    {'prenom': 'elon', 'nom': 'musk'},
    {'prenom': 'johnson', 'nom': 'boris'},
    {'prenom': 'marlène', 'nom': 'shiappa'},
    {'prenom': 'marcel', 'nom': 'jeanno'},
    {'prenom': 'joe', 'nom': 'goldberg'},
    {'prenom': 'han', 'nom': 'Mi-nyeo'},
    {'prenom': 'el', 'nom': 'professor'},
    {'prenom': 'cho', 'nom': 'Sang-Woo'},
    {'prenom': 'oh', 'nom': 'Il-nam'},
    {'prenom': 'john', 'nom': 'snow'},
    {'prenom': 'harry', 'nom': 'potter'},
    {'prenom': 'brahim', 'nom': 'bouhlel'},
    {'prenom': 'stéphane', 'nom': 'chicheman'},
];
for (var i = 0; i < eleves.length; i++) {
    console.log(`Bonjour ${eleves[i].prenom}`);
}
```



Avec la boucle, en 3 lignes, c'est réglé. Tous mes élèves seront salués et si une personnes modifie le tableau, la liste de salutations se mettra à jour automatiquement...

# Algorithmie - Les Boucles

```
for (var i = 0; i < Things.length; i++) {
    Things[i]
}

array.forEach( function(element, index) {
    // statements
});

while (condition) {
    // statement
}
```

Il existe plusieurs types de boucle... Nous allons en voir 3 for, forEach et while

## Algorithmie - Les Boucles

```
while (tant_que_je_suis_vrai) {  
    console.log('je serais exécuté....')  
}
```

La signification d'une boucle while est très simple. JS exécute l'instruction tant que l'expression de la boucle while est évaluée comme true. La valeur de l'expression est vérifiée à chaque début de boucle, et, si la valeur change durant l'exécution de l'instruction, l'exécution ne s'arrêtera qu'à la fin de l'itération (chaque fois que JS exécute l'instruction, on appelle cela une itération). Si l'expression du while est false avant la première itération, l'instruction ne sera jamais exécutée.

# Algorithmie - Les Boucles

```
const eleves = [
    {'prenom': 'nicolas', 'nom':'dupont'},
    {'prenom': 'sophie', 'nom':'durand'},
    {'prenom': 'clémence', 'nom':'dupuis'},
    {'prenom': 'emmanuel', 'nom':'macron'},
    {'prenom': 'elon', 'nom':'musk'},
    {'prenom': 'johnson', 'nom':'boris'},
    {'prenom': 'marlène', 'nom':'shiappa'},
    {'prenom': 'marcel', 'nom':'jeanno'},
    {'prenom': 'joe', 'nom':'goldberg'},
    {'prenom': 'han', 'nom':'Mi-nyeo'},
    {'prenom': 'el', 'nom':'professor'},
    {'prenom': 'cho', 'nom':'Sang-Woo'},
    {'prenom': 'oh', 'nom':'Il-nam'},
    {'prenom': 'john', 'nom':'snow'},
    {'prenom': 'harry', 'nom':'potter'},
    {'prenom': 'brahim', 'nom':'bouhlel'},
    {'prenom': 'stéphane', 'nom':'chicheman'},
];
```

```
let i = 0;
while(i < eleves.length) {
    console.log(`Bonjour ${eleves[i].prenom}`);
    i++;
}
```

Pouvez-vous m'expliquez svp ?

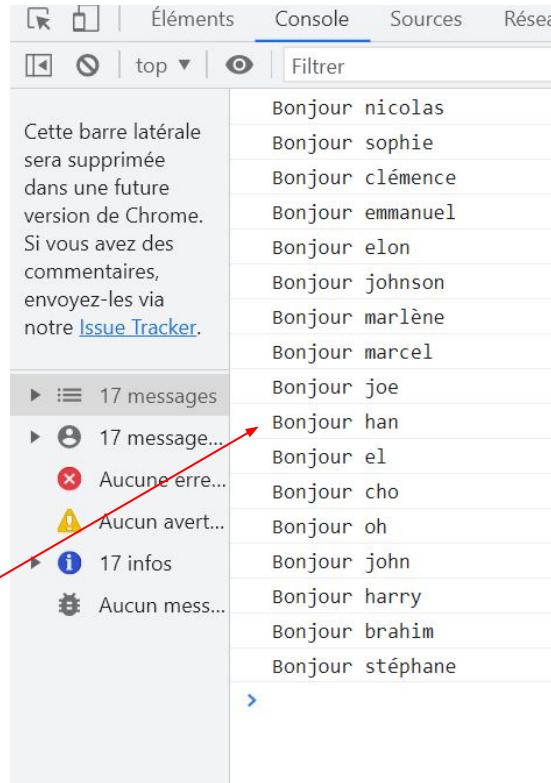
# Algorithmie - Les Boucles

```
//On crée une variable i (comme index) et
//on l'initialise à 0....
let i = 0;
//On dit "tant que i est strictement plus petit que la taille du tableau 'élèves'"
//La boucle s'arrêtera forcément à la fin du tableau car eleves.length = 17
//Et comme l'index de la dernière ligne est "16" (on commence à 0)
while(i < eleves.length) {
    //On affiche "Bonjour nom_de_l'eleve"
    //Donc au premier tour (i vaut 0) eleves[0].prenom soit 'nicolas'
    //Donc au deuxième tour (i vaut 1) eleves[1].prenom soit 'sophie'
    //...
    console.log(`Bonjour ${eleves[i].prenom}`);
    //On incrémente i de 1 (donc à la fin du premier tour, i sera égal à 1)
    //On incrémente i de 1 (donc à la fin du deuxième tour, i sera égal à 2)
    //...
    //i ne dépassera jamais 16 car dans ma condition il doit s'arrêter quand i
    //n'est plus strictement inférieur à 17 (donc 16)
    i++;
}
```

# Algorithmie - Les Boucles

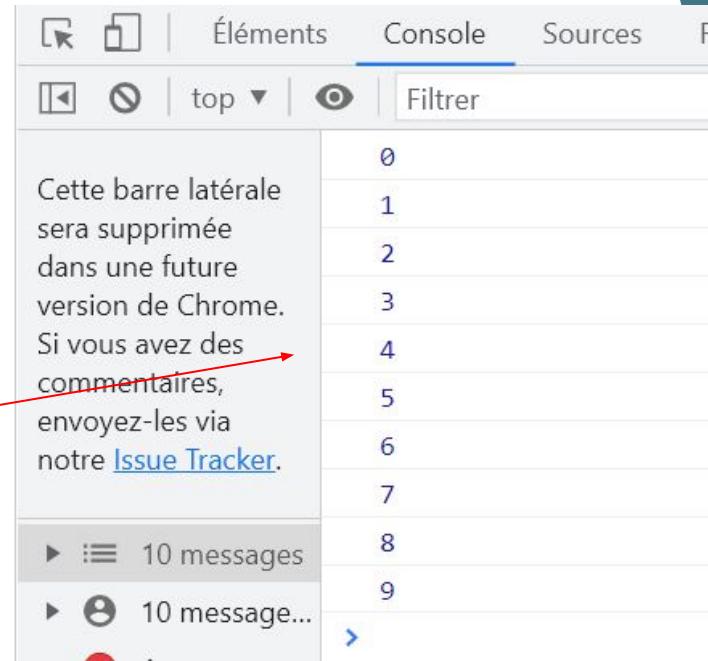
```
const eleves = [
    {'prenom': 'nicolas', 'nom':'dupont'},
    {'prenom': 'sophie', 'nom':'durand'},
    {'prenom': 'clémence', 'nom':'dupuis'},
    {'prenom': 'emmanuel', 'nom':'macron'},
    {'prenom': 'elon', 'nom':'musk'},
    {'prenom': 'johnson', 'nom':'boris'},
    {'prenom': 'marlène', 'nom':'shiappa'},
    {'prenom': 'marcel', 'nom':'jeanno'},
    {'prenom': 'joe', 'nom':'goldberg'},
    {'prenom': 'han', 'nom':'Mi-nyeo'},
    {'prenom': 'el', 'nom':'professor'},
    {'prenom': 'cho', 'nom':'Sang-Woo'},
    {'prenom': 'oh', 'nom':'Il-nam'},
    {'prenom': 'john', 'nom':'snow'},
    {'prenom': 'harry', 'nom':'potter'},
    {'prenom': 'brahim', 'nom':'bouhlel'},
    {'prenom': 'stéphane', 'nom':'chicheman'},
];

let i = 0;
while(i < eleves.length) {
    console.log(`Bonjour ${eleves[i].prenom}`);
    i++;
}
```



# Algorithmie - Les Boucles

```
let i = 0;  
while(i < 10) {  
    console.log(i);  
    i++;  
}
```



La boucle while n'a pas besoin de tableau pour fonctionner, juste une limite...

# Algorithmie - Les Boucles

```
for (var i = 0; i < limite_a_pas_depasser; i++) {  
    console.log('je m\'affiche...')  
}
```

La boucle for...

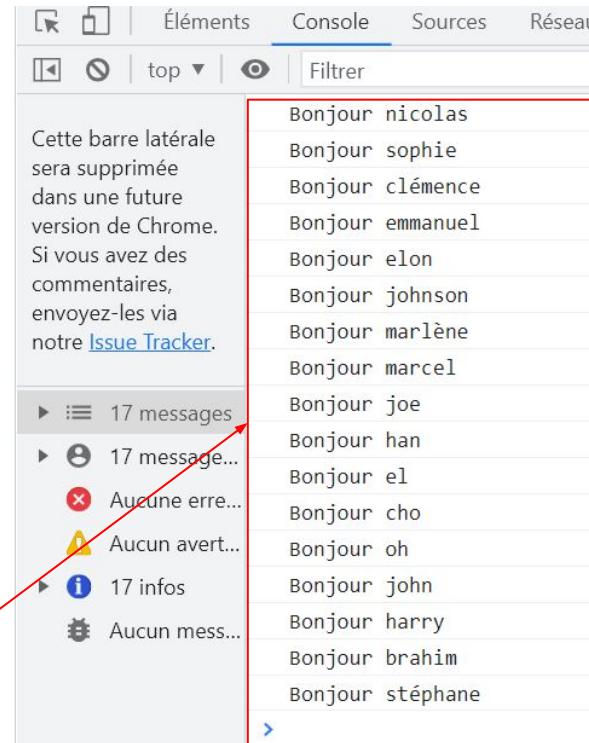
La première expression (expr1) est évaluée (exécutée), quoi qu'il arrive au début de la boucle.

Au début de chaque itération, l'expression expr2 est évaluée. Si l'évaluation vaut true, la boucle continue et les commandes sont exécutées. Si l'évaluation vaut false, l'exécution de la boucle s'arrête.

À la fin de chaque itération, l'expression expr3 est évaluée (exécutée).

# Algorithmie - Les Boucles

```
const eleves = [
    {'prenom': 'nicolas', 'nom': 'dupont'},
    {'prenom': 'sophie', 'nom': 'durand'},
    {'prenom': 'clémence', 'nom': 'dupuis'},
    {'prenom': 'emmanuel', 'nom': 'macron'},
    {'prenom': 'elon', 'nom': 'musk'},
    {'prenom': 'johnson', 'nom': 'boris'},
    {'prenom': 'marlène', 'nom': 'shiappa'},
    {'prenom': 'marcel', 'nom': 'jeanno'},
    {'prenom': 'joe', 'nom': 'goldberg'},
    {'prenom': 'han', 'nom': 'Mi-nyeo'},
    {'prenom': 'el', 'nom': 'professor'},
    {'prenom': 'cho', 'nom': 'Sang-Woo'},
    {'prenom': 'oh', 'nom': 'Il-nam'},
    {'prenom': 'john', 'nom': 'snow'},
    {'prenom': 'harry', 'nom': 'potter'},
    {'prenom': 'brahim', 'nom': 'bouhlel'},
    {'prenom': 'stéphane', 'nom': 'chicheman'},
];
for (var i = 0; i < eleves.length; i++) {
    console.log(`Bonjour ${eleves[i].prenom}`);
}
```



# Algorithmie - Les Boucles

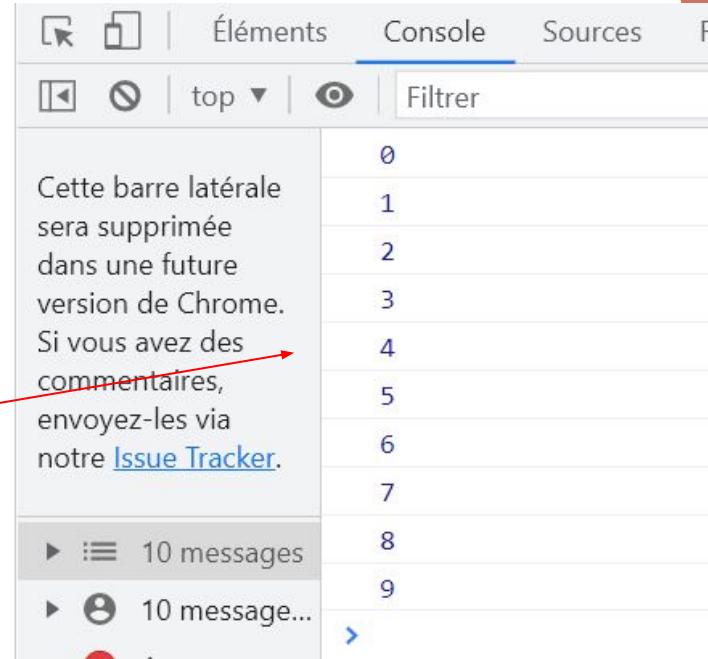
```
let i = 0;  
while(i < 10) {  
    console.log(i);  
    i++;  
}
```

```
for (var i = 0; i < 10; i++) {  
    console.log(i);  
}
```

La boucle “for” c'est juste une boucle while mieux organisée car l'index de départ, la condition et l'incrémentation (ou décrémentation) sont intégrés dans la boucles.

# Algorithmie - Les Boucles

```
for (var i = 0; i < 10; i++) {  
    console.log(i);  
}
```



Comme la boucle “while”, la boucle “for” n'a pas besoin de tableau pour fonctionner, juste une limite...

## Algorithmie - Les Boucles

```
array.forEach( function(element, index) {  
    // statements  
});
```

Ca, c'est une boucle “forEach” (en français “pour chaque”). Contrairement au 2 autres boucles, celle-ci à besoin d'un tableau pour être exécutée.  
La boucle “forEach” n'a pas besoin de limite, elle va parcourir toutes les lignes d'un tableau de la première à la dernière et s'arrêter d'elle-même.  
Pas besoin non plus de lui indiquer un index de départ, c'est 0 (la première ligne) et pas besoin non plus de lui dire d'incrémenter ou de décrémenter, elle incrémente.

# Algorithmie - Les Boucles

```
const eleves = [
    {'prenom': 'nicolas', 'nom':'dupont'},
    {'prenom': 'sophie', 'nom':'durand'},
    {'prenom': 'clémence', 'nom':'dupuis'},
    {'prenom': 'emmanuel', 'nom':'macron'},
    {'prenom': 'elon', 'nom':'musk'},
    {'prenom': 'johnson', 'nom':'boris'},
    {'prenom': 'marlène', 'nom':'shiappa'},
    {'prenom': 'marcel', 'nom':'jeanno'},
    {'prenom': 'joe', 'nom':'goldberg'},
    {'prenom': 'han', 'nom':'Mi-nyeo'},
    {'prenom': 'el', 'nom':'professor'},
    {'prenom': 'cho', 'nom':'Sang-Woo'},
    {'prenom': 'oh', 'nom':'Il-nam'},
    {'prenom': 'john', 'nom':'snow'},
    {'prenom': 'harry', 'nom':'potter'},
    {'prenom': 'brahim', 'nom':'bouhlel'},
    {'prenom': 'stéphane', 'nom':'chicheman'},
];

```

```
eleves.forEach( function(element, index) {
    console.log(element.prenom); // eleves[index].prenom
});
```

Ligne du tableau en cours  
(équivalent de `eleves[i]`)

Tableau à parcourir

Index de la ligne

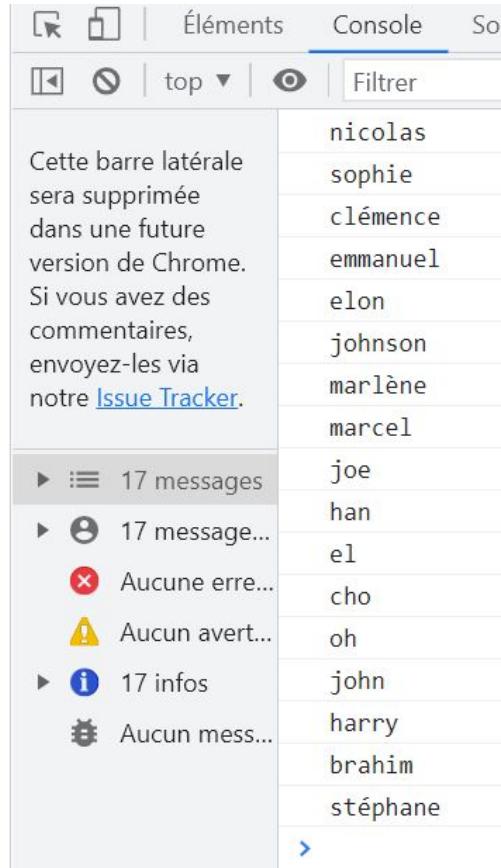
Éléments	Console	So
top	Filtrer	
Cette barre latérale sera supprimée dans une future version de Chrome. Si vous avez des commentaires, envoyez-les via notre <a href="#">Issue Tracker</a> .	nicolas sophie clémence emmanuel elon johnson marlène marcel joe han el cho oh john harry brahim stéphane	
▶ 17 messages		
▶ 17 message...		
✖ Aucune erre...		
⚠ Aucun avert...		
▶ 17 infos		
⚙ Aucun mess...		
>		

Du coup pour sélectionner une valeur, il suffit de mettre le nom de la clé à la suite d'"element" (qui est la ligne en cours)...

# Algorithmie - Les Boucles

```
const eleves = [
    {'prenom': 'nicolas', 'nom': 'dupont'},
    {'prenom': 'sophie', 'nom': 'durand'},
    {'prenom': 'clémence', 'nom': 'dupuis'},
    {'prenom': 'emmanuel', 'nom': 'macron'},
    {'prenom': 'elon', 'nom': 'musk'},
    {'prenom': 'johnson', 'nom': 'boris'},
    {'prenom': 'marlène', 'nom': 'shiappa'},
    {'prenom': 'marcel', 'nom': 'jeanno'},
    {'prenom': 'joe', 'nom': 'goldberg'},
    {'prenom': 'han', 'nom': 'Mi-nyeo'},
    {'prenom': 'el', 'nom': 'professor'},
    {'prenom': 'cho', 'nom': 'Sang-Woo'},
    {'prenom': 'oh', 'nom': 'Il-nam'},
    {'prenom': 'john', 'nom': 'snow'},
    {'prenom': 'harry', 'nom': 'potter'},
    {'prenom': 'brahim', 'nom': 'bouhlel'},
    {'prenom': 'stéphane', 'nom': 'chicheman'},
];
eleves.forEach( function(element, index) {
    console.log(eleves[index].prenom);
});
```

Du coup, cette façon fonctionne aussi pour récupérer les valeurs...



## Algorithmie - Les Boucles

```
eleves.forEach( function(element) {  
    console.log(element.prenom);  
});
```

Juste pour info, l'argument “index” est optionnel.

# **Algorithmie, Les Fonctions**

## Algorithmie - Les Fonctions

Une fonction correspond à un bloc de code nommé et réutilisable et dont le but est d'effectuer une tâche précise. Elles sont très utilisées en JS et dans les autres langages pouvant les supporter.

Elles servent à éviter la duplication de code et avoir une meilleure organisation. Elles peuvent retourner une valeur (chiffre, texte, booléen, tableau,...) ou ne rien retourner et juste exécuter un code.

Ils existent 2 type de fonctions, les natives et les prédéfinies. Dans les 2 cas, elles sont aussi appelées "méthodes"

## Algorithmie - Les Fonctions

```
eleves.unshift();
eleves.pop();

console.log('coucou');
eleves.push({ 'prenom': 'stéphane', 'nom': 'chicheman'});
eleves.splice(0,2);
```

Sans nous en rendre compte, nous avons utilisé plusieurs fonctions natives de JS. push(), pop(), shift(), log().... Une fonction se reconnaît à sa syntaxe particulière : **nomDeLaFonction()**

Avec ses parenthèses (vides ou non), la fonction est facilement reconnaissable. Dans les parenthèses, on peut passer un ou plusieurs paramètres (arguments).... Ils sont séparés par des virgules.

## Algorithmie - Les Fonctions

```
console.log('coucou');
eleves.push({'prenom': 'stéphane', 'nom':'chicheman'});
eleves.splice(0,2);
```

Les arguments peuvent être une chaîne de caractère, des nombres, un tableau, objet,... Ou même une autre fonction.

Un argument peut être optionnel ou obligatoire

# Algorithmie - Les Fonctions

```
eleves.forEach( function(element) {  
    console.log(element.prenom);  
});
```

Une fonction peut être nommée ( ex : push() ) ou non..... Dans le cas où elle ne l'est pas on dit qu'elle est anonyme.

La méthode (fonction) forEach à en arguments une fonction anonyme.

## Algorithmie - Les Fonctions

```
eleves.forEach( function(element) {  
    console.log(element.prenom);  
});
```

Quand nous utilisons une fonction sans que nous l'ayons prédefinie (créé nous-même), on dit qu'elle est native. C'est à dire que JS l'a créé et mis à notre disposition. Nous n'avons plus qu'à l'appeler pour l'utiliser.

# Algorithmie - Les Fonctions

```
function mon_nom_de_fonction(){
    //Mon code ici
}
```

Si nous voulons créer notre propre fonction, il faut l'initialiser (un peu comme une variable) avec la commande (mot) ‘function’, ensuite il faut lui donner un nom (ce sont les même règles de nommage que les variable, mettre 2 parenthèses (même si il n'y a pas d'arguments (paramètres) et 2 accolades avec entre deux le code à exécuter quand la fonction est appelée.

## Algorithmie - Les Fonctions

The screenshot shows a browser's developer tools open to the 'Console' tab. At the top, there are icons for back, forward, and search, followed by tabs for 'Éléments', 'Console' (which is underlined), 'Sources', 'Réseau', and 'Perfo'. Below the tabs is a toolbar with icons for back, forward, and search, followed by a dropdown set to 'top' and a 'Filtrer' (Filter) button. A sidebar on the left labeled 'Cette barre latérale' contains a blue arrow icon pointing right. The main area of the console shows the following code:

```
function afficherDansConsole(){
    console.log('ma première fonction ;');
}

afficherDansConsole();
```

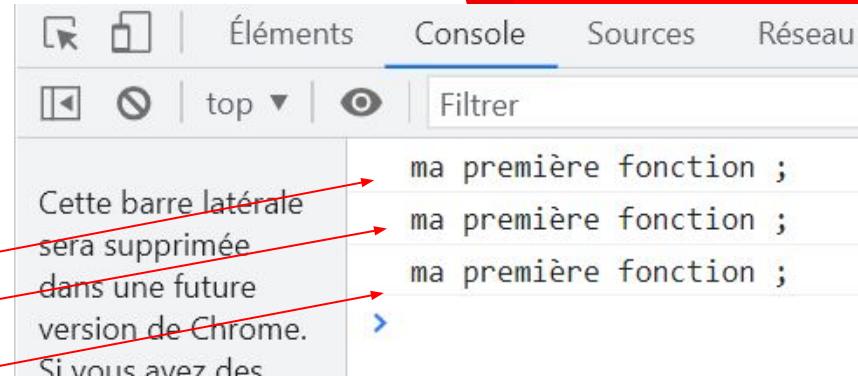
A red oval highlights the line 'afficherDansConsole();'. Two red arrows point from this oval to the text 'ma première fonction ;' in the console output above it.

Nous venons de créer notre première fonction (sommaire certe). Cette dernière va juste afficher 'ma première fonction ;' dans la console (pas foufou mais faut un début un tout).

Une fois créée, j'appelle ma fonction juste en copiant le nom de la fonction...

# Algorithmie - Les Fonctions

```
function afficherDansConsole(){  
    console.log('ma première fonction ;');  
}  
  
afficherDansConsole();  
afficherDansConsole();  
afficherDansConsole();
```

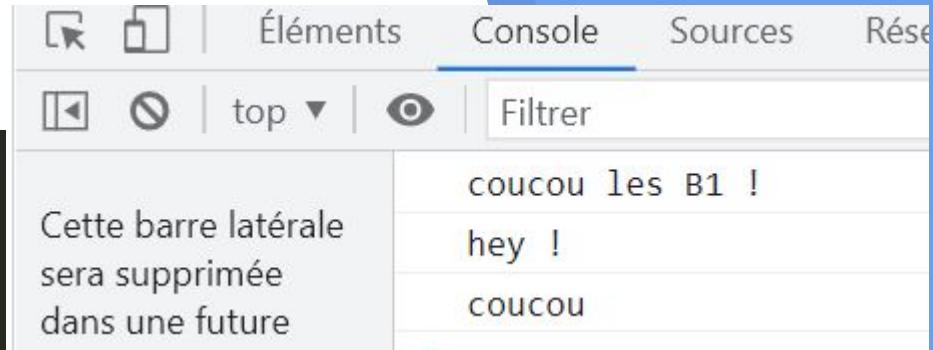


Bon, le truc c'est que je peux appeler ma fonction autant de fois que je veux,  
J'ai toujours le même texte....y a pas trop d'intérêt.

## Algorithmie - Les Fonctions

```
function afficherDansConsole($texte){  
    console.log($texte);  
}
```

```
afficherDansConsole('coucou les B1 !');  
afficherDansConsole('hey !');  
afficherDansConsole('coucou');
```

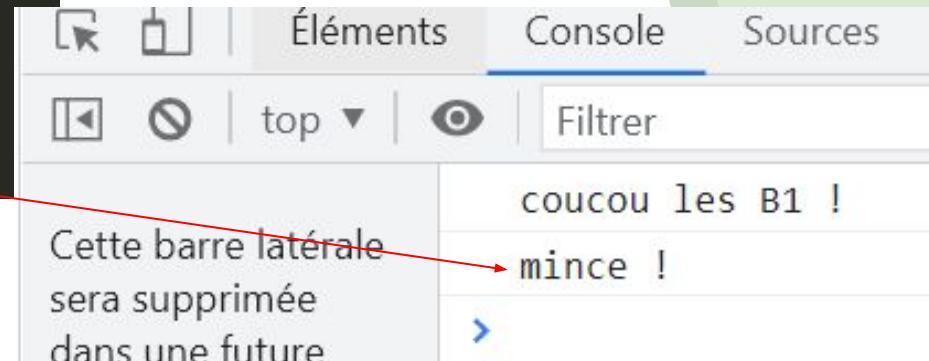


Pour changer un peu, je décide d'ajouter un argument que j'ai appelé ‘\$texte’. ‘\$texte’ est une variable qui va s'ajouter dans mon console.log() et donc varier en fonction du contenu....

# Algorithmie - Les Fonctions

```
function afficherDansConsole($texte='mince !'){
    console.log($texte);
}

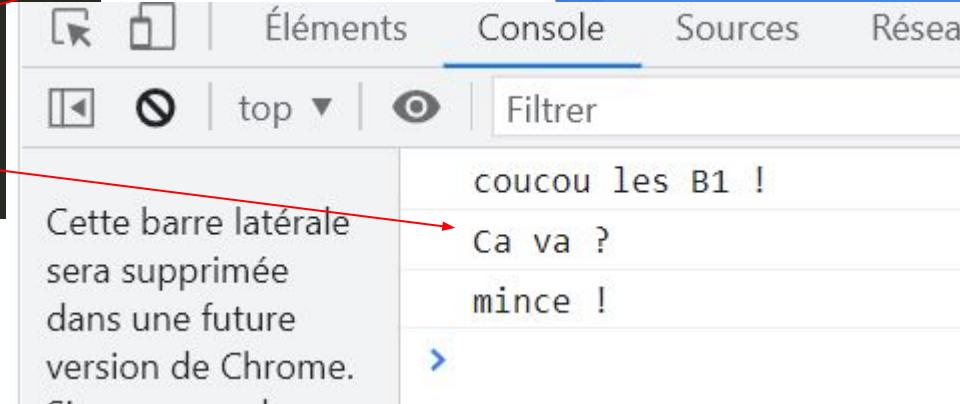
afficherDansConsole('coucou les B1 !');
afficherDansConsole();
```



Je peux assigner une valeur par défaut à mon argument qui sera affiché si j'appelle ma fonction sans lui préciser de texte. En gros, on lui dit “si vide, affiche moi la valeur par défaut (qui est “mince !”)”

# Algorithmie - Les Fonctions

```
function afficherDansConsole($texte='mince !', $option=false){  
    console.log($texte);  
    if($option) console.log($option);  
}  
  
afficherDansConsole('coucou les B1 !', 'Ca va ?');  
afficherDansConsole();
```



Nous venons de créer un second paramètre optionnel...

Avec une condition dans la fonction qui dit “si \$option existe, affiche le, sinon rien”.

# Algorithmie - Les Fonctions

```
//fonction déclarative
function somme(a, b){
    return a + b;
}

//fonction d'expression
let somme = function(a,b){
    return a + b;
}

// fonction fléchée
let somme = (a,b)=> a+b;

// fonction fléchée
let double = a => a*2;
```

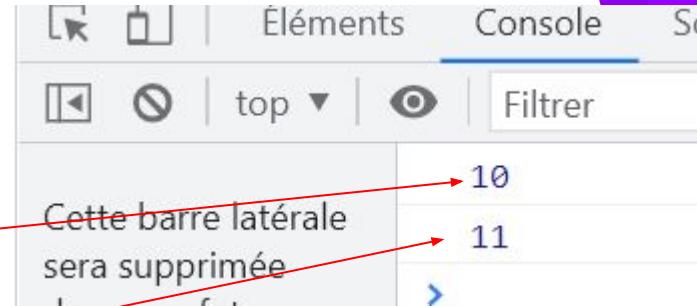
Autre syntaxe que vous pouvez croiser....

# Algorithmie - Les Fonctions

```
//fonction déclarative
function somme(a, b){
    return a + b;
    console.log('je ne serais jamais exécuté');
}

console.log(somme(2,8));

let sum = somme(2,8);
sum++;
console.log(sum);
```



L'instruction “return” met fin à l'exécution d'une fonction et définit une valeur à renvoyer à la fonction appelante.

Une fonction sans “return” ne retourne rien, elle exécute un code.

L'avantage du “return” c'est qu'il est possible de stocker le résultat d'une fonction dans une variable soit pour l'afficher plus tard soit la manipuler puis l'afficher.

# **Algorithmie, AJAX**

## Algorithmie - AJAX

“L’AJAX” ou plutôt “l’Ajax” (Asynchronous JavaScript and XML) 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.

En gros, il est possible de récupérer (ou d’envoyer) des données sur un serveur et de les importer directement dans notre page pour les traiter.

# Algorithmie - AJAX

```
fetch('https://random-data-api.com/api/users/random_user?size=3')
.then((data)=>{
    return data.json();
})
.then((data)=>{
    console.log(data);
})
```

```
api.js:34
▼ (3) [ {...}, {...}, {...} ] ⓘ
  ▼ 0:
    ► address: {city: 'Bogisichland', street_name: 'DuBuque Union', street_address: '9852 Taren Free...', avatar: "https://robohash.org/remipsameaque.png?size=300x300&set=set1"
    ► credit_card: {cc_number: '6771-8996-0338-3505'}
    ► date_of_birth: "1958-05-23"
    ► email: "norberto.dooley@email.com"
    ► employment: {title: 'Dynamic Retail Supervisor', key_skill: 'Leadership'}
    ► first_name: "Norberto"
    ► gender: "Male"
    ► id: 9121
    ► last_name: "Dooley"
    ► password: "wuopYdOTkl"
    ► phone_number: "+354 (847) 215-2323"
    ► social_insurance_number: "655371334"
    ► subscription: {plan: 'Diamond', status: 'Pending', payment_method: 'Cheque', term: 'Monthly'}
    ► uid: "64f7426d-ad34-4af0-861b-75d5d24c41d3"
    ► username: "norberto.dooley"
    ► [[Prototype]]: Object
  ▷ 1: {id: 5707, uid: '04a7c704-8c05-4ff0-9961-1f74d2bd950c', password: 'vzDdb1Hh0V', first_name: ...}
  ▷ 2: {id: 5829, uid: '7df9aae3-82f1-4e4a-a179-7e6aede2f05d', password: 'l8PmRuZ200', first_name: ...}
  length: 3
  ► [[Prototype]]: Array(0)
```

Dans cette exemple, j'utilise la fonction “fetch” (qui elle utilise des promesses, mais on y va doucement) qui va me récupérer des données via l'api “<https://random-data-api.com/>” (c'est une api de test).

## Algorithmie - AJAX

```
fetch('https://random-data-api.com/api/users/random_user?size=3')
.then((data)=>{
    return data.json();
})
.then((data)=>{
    console.log(data);
})
```

Dans un premier temps, fetch envoie une requête à un serveur :

[https://random-data-api.com/api/users/random\\_user](https://random-data-api.com/api/users/random_user)

Dans un second temps, dès qu'il reçoit les données de ce serveur, il les converti on JSON (data.json())

Et enfin, dès que c'est fini, il les affiche....

# Algorithmie - AJAX

```
api.js:34
▼ (3) [ {...}, {...}, {...} ] ⓘ
  ▼ 0: ⚡
    ► address: {city: 'Bogisichland', street_name: 'DuBuque Union', street_address: '9852 Taren Free...', avatar: "https://robohash.org/remipsameaque.png?size=300x300&set=set1"}
    ► credit_card: {cc_number: '6771-8996-0338-3505'}
    ► date_of_birth: "1958-05-23"
    ► email: "horberto.dooley@email.com"
    ► employment: {title: 'Dynamic Retail Supervisor', key_skill: 'Leadership'}
    ► first_name: "Norberto"
    ► gender: "Male"
    ► id: 9121
    ► last_name: "Dooley"
    ► password: "wuopVd0Tkl"
    ► phone_number: "+354 (847) 215-2323"
    ► social_insurance_number: "655371334"
    ► subscription: {plan: 'Diamond', status: 'Pending', payment_method: 'Cheque', term: 'Monthly'}
    ► uid: "64f7426d-ad34-4af0-861b-75d5d24c41d3"
    ► username: "norberto.dooley"
    ► [[Prototype]]: Object
  ▶ 1: {id: 5707, uid: '04a7c704-8c05-4ff0-9961-1f74d2bd950c', password: 'vzDdblHh0V', first_name: ...}
  ▶ 2: {id: 5829, uid: '7df9aae3-82f1-4e4a-a179-7e6aede2f05d', password: 'l8PmRuZ200', first_name: ...}
  ► length: 3
  ► [[Prototype]]: Array(0)
>
```

La réponse de cette api est un tableau à 3 lignes. Chaque ligne correspond à une personne avec des données personnelles (nom, genre, poste, email, adresse,...). Pour info, tout est faux, c'est une api pour s'entraîner.

# Algorithmie - AJAX

```
fetch('https://www.googleapis.com/youtube/v3/playlistItems?part=snippet&playlistId=UUWyVrwrcCJ-tVVj0qU5Je4Q&maxResults=1&key=AIzaSy...').then((data)=>{ return data.json();}).then((data)=>{ console.log(data);})
```



```
▼ object 1
  etag: "GYYQ_1Hlge933_3FBuxDFylWhf8"
  ▼ items: Array(1)
    ▼ 0:
      etag: "37MuBsGpHsRGwj2ndPeCAADyuJ0"
      id: "VVVxeVZyd3j0otdFZWajBxTVKZTRRLmM2Mmt2M2VSzk1j"
      kind: "youtube#playlistItem"
      ▼ snippet:
        channelId: "UCWyVrwrcCJ-tVVj0qU5Je4Q"
        channelTitle: "sdcomm channel"
        description: ""
        playlistId: "UUWyVrwrcCJ-tVVj0qU5Je4Q"
        position: 0
        publishedAt: "2021-12-01T14:21:22Z"
      ▼ resourceId:
        kind: "youtube#video"
        videoId: "c62kv3erfIc"
      ► [[Prototype]]: object
      ▶ thumbnails: {default: {...}, medium: {...}, high: {...}, standard: {...}, maxres: {...}}
      title: "Stores"
      videoOwnerChannelId: "UCWyVrwrcCJ-tVVj0qU5Je4Q"
      videoOwnerChannelTitle: "sdcomm channel"
      ► [[Prototype]]: Object
      ► [[Prototype]]: Object
      length: 1
      ► [[Prototype]]: Array(0)
      kind: "youtube#playlistItemListResponse"
      nextPageToken: "EAAaBIBUOkNBRQ"
```

Autre exemple, cette api permet d'aller récupérer des données directement sur youtube pour pouvoir intégrer de façon automatique des vidéo sur le site. Vous mettez une vidéo sur youtube, elle s'affiche directement sur le site....

# Algorithmie - AJAX

The screenshot shows a web browser window with the URL [https://vitemadose.covidtracker.fr/centres-vaccination-covid-dpt14-calvados/commune14047-14400-bayeux/...](https://vitemadose.covidtracker.fr/centres-vaccination-covid-dpt14-calvados/commune14047-14400-bayeux/). The main content area displays a search result for a vaccination center:

- 1 Lieu de vaccination avec des disponibilités
- 1 créneau trouvé - 11 km
- Mme Marion HOTTIN  
11 Rue de Southampton, 14960 Asnelles
- 07 69 70 64 66
- Centre de vaccination
- Pfizer-BioNTech
- [Prendre rendez-vous](#)
- 1 créneau |
- Aucun rendez-vous - 12 km

A red arrow points from the "Mme Marion HOTTIN" link on the left to the "Network" tab of the developer tools on the right. The Network tab shows a single request to <https://www.doctolib.fr/vaccination-covid-19/asnelles/marion-hottin?pid=practice-222271>. The response body contains the following JSON data:

```
{
  "departement": "14",
  "nom": "Mme Marion HOTTIN",
  "url": "https://www.doctolib.fr/vaccination-covid-19/asnelles/marion-hottin?pid=practice-222271",
  "location": {
    "longitude": -0.5857566,
    "latitude": 49.3381674,
    "city": "Asnelles",
    "cp": "14960"
  },
  "metadata": {
    "address": "11 Rue de Southampton, 14960 Asnelles",
    "business_hours": {
      "lundi": "07:00-12:00, 17:00-19:00",
      "mardi": "07:00-12:00, 17:00-19:00",
      "mercredi": "07:00-12:00, 17:00-19:00",
      "jeudi": "07:00-12:00, 17:00-19:00",
      "vendredi": "07:00-12:00, 17:00-19:00",
      "samedi": "07:00-12:00, 17:00-19:00",
      "dimanche": "07:00-12:00, 17:00-19:00"
    },
    "phone_number": "+33769706466"
  },
  "prochain_rdv": "2021-12-15T17:30:00+01:00",
  "plateforme": "Doctolib",
  "type": "vaccination-center",
  "appointment_count": 686,
  "internal_id": "doctolib320656pid222271",
  "vaccine_type": [
    "Pfizer-BioNTech"
  ]
}
```

Autre exemple, le site “vitemadose”, utilise les api des plateformes de prise de rendez-vous et les affiche directement dans son site

# Algorithmie - AJAX

## Centres de vaccination du 14

### CHU CAEN - Vaccination Covid-19

Adresse : AVENUE COTE DE NACRE, 14033 CAEN

Téléphone : +33279461156

Url : <https://www.doctolib.fr/vaccination-covid-19/caen/chu-caen-vaccination-covid?pid=practice-163773>

### Centre de vaccination COVID-19 - Opérations éphémères Calvados

Adresse : 6 Rue du Régiment Mont Royal, 14320 Laize-Clinchamps

Téléphone : +33800009110

Url : <https://www.doctolib.fr/vaccination-covid-19/mondeville/centre-de-vaccination-covid-19-operactions-ephemeres-calvados?pid=practice-224321>

### Cabinet SOS de vaccination COVID

Adresse : 3 Place Jean Nouzille, 14000 Caen

Téléphone : +33221002200

Url : <https://www.doctolib.fr/vaccination-covid-19/caen/cabinet-sos-de-vaccination-covid?pid=practice-181799>

D'ailleurs, si je veux afficher tous les centres du calvados (et même ailleurs) sur mon site, j'ai juste à récupérer l'API et injecter les donner dans mon site (bon, là ya pas de css....)

The screenshot shows a browser's developer tools with the DOM tree and the JavaScript code side-by-side.

**DOM Tree:**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

    <h1>Centres de vaccination du 14</h1>
    <div class="display"></div>

    <script src="api.js"></script>
</body>
</html>
```

**JavaScript Code:**

```
element.style = {}

fetch('https://vitemadose.gitlab.io/vitemadose/14.json')
.then((data)=>{
    return data.json();
})
.then((data)=>{
    data.centres_disponibles.forEach((el,i)=>{
        let template=<h2>{nom}</h2> <p>Adresse : {address}</p> <p>Téléphone : {phone}</p> <p>Url : <a href='{url}'>{url}</a></p>;
        const div = document.createElement('div');
        div.classList.add('centre');
        template = template.replace('{nom}',el.nom).replace('{address}', el.metadata.address).replace('{phone}', el.metadata.phone_number).replace(//{url}/g,el.url);
        div.innerHTML = template;
        document.querySelector('.display').appendChild(div);
    })
})
```

# Algorithmie - AJAX

**API Découpage Administratif - (API Geo)** 

Interrogez les référentiels géographiques plus facilement

Produit par : Direction Interministérielle du Numérique

 Accès libre     100.00% actif / dernier mois

**API Impôt particulier** 

Raccordez-vous directement à la DGFiP pour récupérer les éléments fiscaux nécessaires à vos téléservices, éliminez le traitement et le stockage des pièces justificatives

 Peut s'utiliser avec FranceConnect

Produit par : Direction Générale des Finances Publiques

 Sous habilitation

**API Entreprise** 

Entités administratives, simplifiez les démarches des entreprises et des associations en récupérant pour elles leurs informations administratives.

Produit par : Direction Interministérielle du Numérique

 Sous habilitation     100.00% actif / dernier mois

**API Adresse (Base Adresse Nationale - BAN)** 

Interrogez la Base Adresse Nationale, base de données de l'intégralité des adresses du territoire français

**API Recherche des personnes physiques (R2P)** 

Récupérez les données connues par l'administration fiscale (DGFiP) sur une personne physique (état civil, adresse

**API Fichier des Comptes Bancaires et Assimilés (FICOBAN)** 

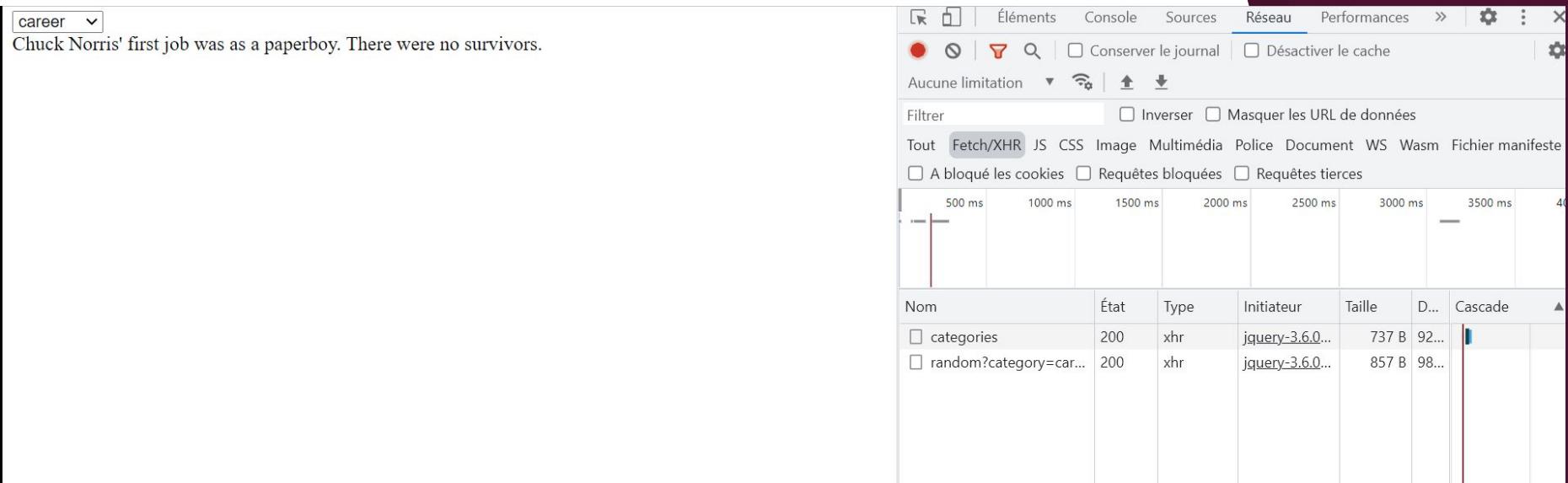
Accédez aux coordonnées des usagers connues de l'administration fiscale (DGFiP) et transmises par les établissements

 Une question ?

Le gouvernement met à disposition plusieurs API (certaines nécessite une autorisation...)

<https://api.gouv.fr/rechercher-api>

# Algorithmie - AJAX



The screenshot shows a browser's developer tools Network tab. The URL bar at the top has 'career' selected. The main content area displays the text: "Chuck Norris' first job was as a paperboy. There were no survivors." Below this, the Network tab is active, showing two requests:

Nom	État	Type	Initiateur	Taille	D...	Cascade
categories	200	xhr	jquery-3.6.0...	737 B	92...	
random?category=car...	200	xhr	jquery-3.6.0...	857 B	98...	

The Network tab includes various filters like 'Fetch/XHR', 'JS', 'CSS', etc., and a timeline at the top.

On constate aussi qu l'ajax permet de recharger les données d'une page sans la recharger.... (oui, j'utilise l'api de chuck norris...)

Toutes ces requêtes sont visibles dans la console > réseaux > onget "fetch/XHR"

# **Algorithmie, Les “callback” et les “promesses”**

## Algorithmie - Promesses et callback

Dans la vie de tous les jours, on dit que deux actions sont synchrones lorsqu'elles se déroulent en même temps ou de manière synchronisée. Au contraire, deux opérations sont asynchrones si elles ne se déroulent pas en même temps ou ne sont pas synchronisées.

En informatique, on dit que deux opérations sont synchrones lorsque la seconde attend que la première ait fini son travail pour démarrer. En bref : le début de l'opération suivante dépend de la complétude de l'opération précédente.

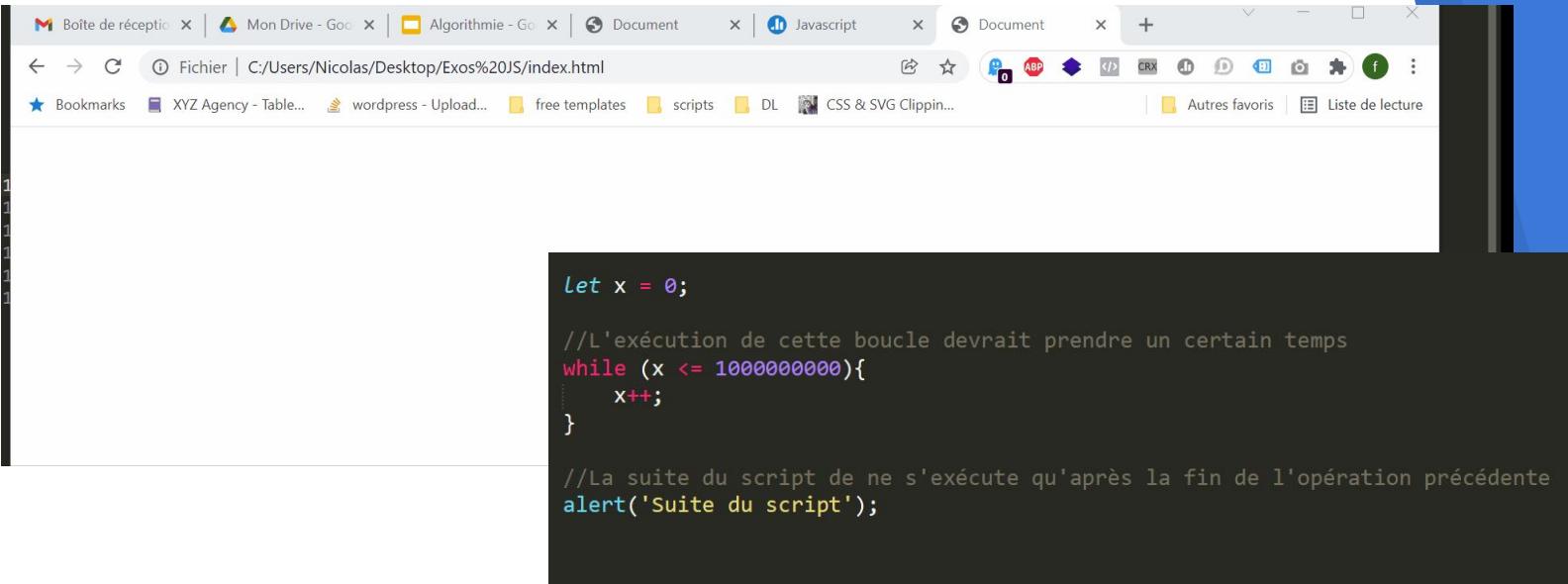
Au contraire, deux opérations sont qualifiées d'asynchrones en informatique lorsqu'elles sont indépendantes c'est-à-dire lorsque la deuxième opération n'a pas besoin d'attendre que la première se termine pour démarrer.

## Algorithmie - Promesses et callback

Par défaut, le JavaScript est un langage synchrone, bloquant et qui ne s'exécute que sur un seul thread. Cela signifie que :

- Les différentes opérations vont s'exécuter les unes à la suite des autres (elles sont synchrones) ;
- Chaque nouvelle opération doit attendre que la précédente ait terminé pour démarrer (l'opération précédente est « bloquante ») ;
- Le JavaScript ne peut exécuter qu'une instruction à la fois (il s'exécute sur un thread, c'est-à-dire un « fil » ou une « tache » ou un « processus » unique).

# Algorithmie - Promesses et callback



The screenshot shows a web browser window with multiple tabs open. The active tab displays a piece of JavaScript code. The code initializes a variable `x` to 0, enters a loop that increments `x` by 1 until it reaches 1 billion, and then displays an alert message "Suite du script". The browser interface includes a toolbar with various icons and a bookmarks bar.

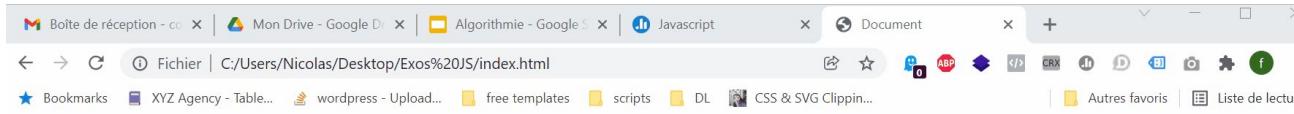
```
Let x = 0;

//L'exécution de cette boucle devrait prendre un certain temps
while (x <= 1000000000){
    ...
    x++;
}

//La suite du script ne s'exécute qu'après la fin de l'opération précédente
alert('Suite du script');
```

Cela peut rapidement poser problème dans un contexte Web : imaginons qu'une de nos fonctions ou qu'une boucle prenne beaucoup de temps à s'exécuter. Tant que cette fonction n'a pas terminé son travail, la suite du script ne peut pas s'exécuter (elle est bloquée) et le programme dans son ensemble paraît complètement arrêté du point de vue de l'utilisateur.

# Algorithmie - Promesses et callback



```
/*setTimeout() est asynchrone : le reste du script va pouvoir s'exécuter
 *sans avoir à attendre la fin de l'exécution de setTimeout()*/
setTimeout(
    ()=>{
        alert('Message après 3 secondes');
    }
, 3000);

//Cette alerte sera affichée avant celle définie dans setTimeout()
alert('Suite du script');
```

En JavaScript, les opérations asynchrones sont placées dans des files d'attentes qui vont s'exécuter après que le fil d'exécution principal ou la tâche principale (le « main thread » en anglais) ait terminé ses opérations. Elles ne bloquent donc pas l'exécution du reste du code JavaScript. `setTimeout()` permet d'exécuter une fonction de rappel après un certain délai ou encore avec la création de gestionnaires d'événements qui vont exécuter une fonction seulement lorsqu'un événement particulier se déclenche.

# Algorithmie - Promesses et callback

```
console.log('je suis le premier')

fetch('https://vitemadose.gitlab.io/vitemadose/14.json')
.then((data)=>{
    return data.json();
})
.then((data)=>{
    console.log('je suis le deuxième')
})

console.log('je suis le troisième')
console.log('je suis le dernier')
```

je suis le premier	<a href="#">api.js:27</a>
je suis le troisième	<a href="#">api.js:40</a>
je suis le dernier	<a href="#">api.js:41</a>
je suis le deuxième	<a href="#">api.js:36</a>

On voit que le deuxième console.log est exécuté en dernier car il est dans une “promise” (promesse) et donc n'est pas dans le fil principal (main thread) mais dans un thread secondaire.

**Mais comment on différencie les promises  
(promesses) des callback (fonction de rappel) ?**

# Algorithmie - Promesses et callback

```
document.querySelector('element').addEventListener('click', function(e){  
    console.log('sera effectué au click sur un élément')  
})  
  
setTimeout(  
    function() {  
        console.log('sera effectué dans 3 secondes')  
    }  
, 3000);
```

Les Callback sont reconnaissables car elles sont effectuées après une conditions (ex 1 après un click, ex 2 après 3 secondes)

## Algorithmie - Promesses et callback

```
fetch('url')
  .then(function(reponse){
    return reponse.json();
})
  .then(function(reponse){
    console.log(reponse)
})
```

Les promesses sont reconnaissables au “then”. Les promesses sont des versions plus évoluées des callbacks. C'est une façon plus propre d'écrire le code.

Les promesses (comme les callback) sont asynchrone, donc ne bloque pas le code principal.

# Algorithmie - Promesses et callback

```
let get = function($url, $success,$error){
    fetch($url)
    .then((data)=>{
        if(data.ok){
            return data.json()
            .then((data)=>{
                return $success(data);
            })
        }
    })
    .catch((error)=>{
        return $error(error.message);
    });
}

get('https://jsonplaceholder.typicode.com/users', (data)=>{
    get('https://jsonplaceholder.typicode.com/comments?userId=' +data[0], (data)=>{
        console.log(data)
    },(error)=>{
        console.error('error ajax',error);
    })
},(error)=>{
    console.error('error ajax',error);
})
```

```
let get = function($url){
    return new Promise((resolve, reject)=>{
        fetch($url)
        .then((data)=>{
            if(data.ok){
                return data.json()
                .then((data)=>{
                    resolve(data);
                })
            }
        })
        .catch((error)=>{
            reject(error.message);
        });
    })
}

get('https://jsonplaceholder.typicode.com/users').then((data)=>{
    return get('https://jsonplaceholder.typicode.com/comments?userId=' +data[0])
}).then((data)=>{
    console.log(data)
}).catch((error)=>{
    console.error('error',error);
})
```



Callback Hell (l'enfer des callback)

# Algorithmie - Promesses et callback

```
const getPosts = async function(){
    let persons = await fetch('https://jsonplaceholder.typicode.com/users');
    persons = await persons.json();
    let comments = await fetch('https://jsonplaceholder.typicode.com/comments?userId=' + persons[0]);
    comments = await comments.json();
    return comments;
}

getPosts().then(function(comments){
    console.log(comments);
})
```

Maintenant, il y a une façon encore plus propre d'écrire des “promises”.  
Grace aux await et async

# **La manipulation du DOM**

# Ajouter un élément dans une page via JavaScript

# Javascript - insérer un élément

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

<script>
    let div = document.createElement('div');

</script>
</body>
</html>
```

Tout d'abord il faut savoir que beaucoup de nos commandes commenceront par “document” (car ça affecte le document en cours), ensuite, pour la création d'un élément, on utilise la méthode ‘createElement(“typeDelementAcréer”)’ et on passe en paramètre le type d' élément que l'on veut créer (dans notre exemple, une “div”). Je stocke l' élément dans une variable “div”.

# Javascript - insérer un élément

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

    <script>
        let div = document.createElement('div');
        document.body.append(div);

    </script>
</body>
</html>
```

Attention, dans l'étape précédente, nous juste créé l'~~élément~~ mais nous ne l'avions pas inséré dans la page. Pour ce faire nous devons écrire "document" (j'avais prévenu qu'on allait souvent l'utiliser), ensuite 'body' ~~car~~ nous voulons l'insérer dans le body du document, puis 'append(élémentAinsérer)' qui prend en paramètre l'élément à insérer.

# Javascript - insérer un élément

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

<script>
    let div = document.createElement('div');
    document.body.append(div);
    ...
</script>
</body>
</html>
```



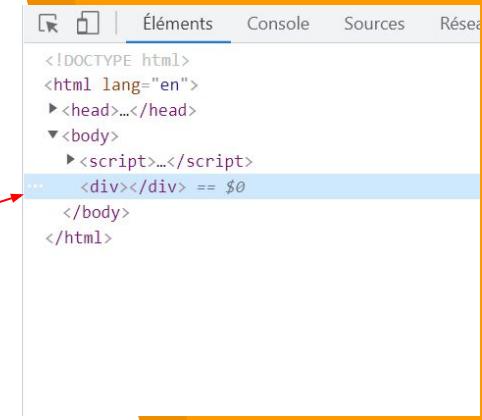
“Append” signifie “à la suite” c'est pour cela que notre “div” se met sous notre balise javascript (elle se met à la suite des éléments déjà présents dans “body”). Si j'insère une autre “div” à l'élément “body”, elle ira sous la balise “div” déjà créée, et ainsi de suite.... Pour l'instant rien ne s'affiche dans la page car notre “div” est vide mais on peut la voir dans notre console.

# Javascript - insérer un élément

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

<script>
    let div = document.createElement('div');
    document.body.appendChild(div);

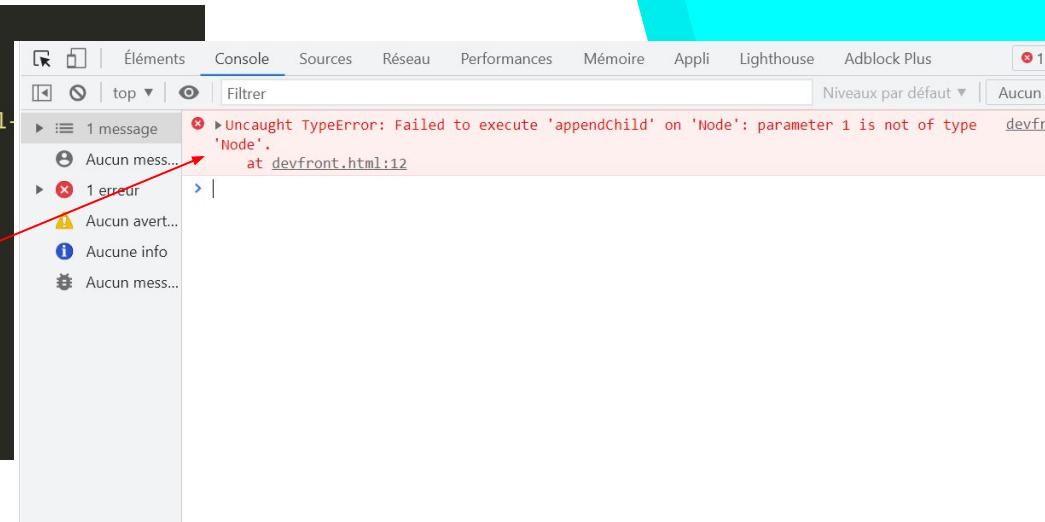
</script>
</body>
</html>
```



Il existe une autre méthode qui ressemble beaucoup à “append”, c'est “appendChild(“élémentAintégrer”)”.

# Javascript - insérer un élément

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-
6     <title>Document</title>
7 </head>
8 <body>
9
10 <script>
11     Let div = document.createElement('div');
12     document.body.appendChild('coucou');
13
14 </script>
15 </body>
16 </html>
```



Les 2 différences entre ces méthodes sont :

- 1/ “appendChild” ne peut pas insérer de texte....

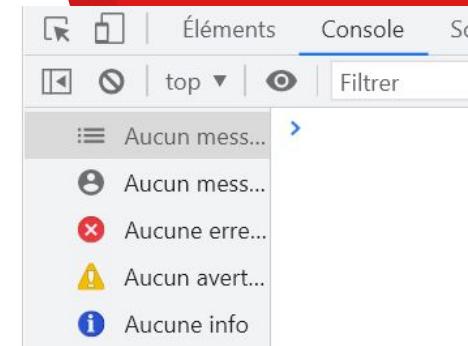
# Javascript - insérer un élément

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

    <script>
        let div = document.createElement('div');
        document.body.append('coucou');

    </script>
</body>
</html>
```

A red arrow points from the word "coucou" in the code to its corresponding appearance in the browser's DOM preview.



Les 2 différences entre ces méthodes sont :

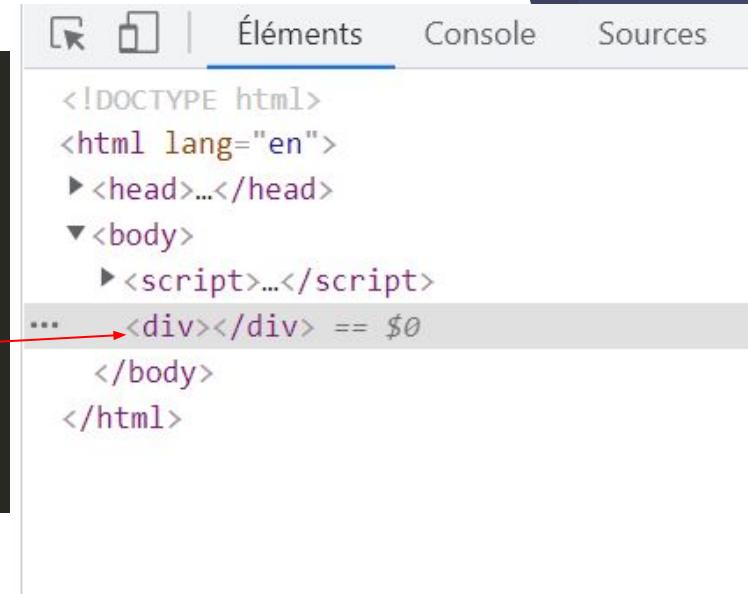
1/ “appendChild” ne peut pas insérer de texte.... Alors que “append” oui !

# Javascript - insérer un élément

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

    <script>
        let div = document.createElement('div');
        let div2 = document.createElement('div');
        document.body.appendChild(div, div2);

    </script>
</body>
</html>
```



Les 2 différences entre ces méthodes sont :

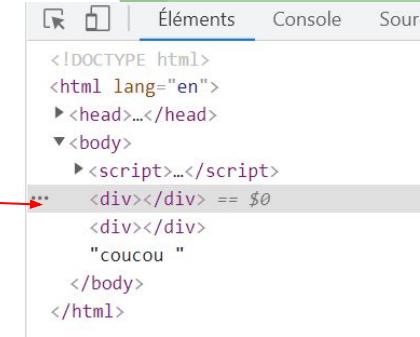
2/ “appendChild” ne peut pas insérer plusieurs éléments à la fois....

# Javascript - insérer un élément

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>

<script>
  let div = document.createElement('div');
  let div2 = document.createElement('div');
  document.body.append(div, div2, 'coucou');
</script>
</body>
</html>
```

coucou



Les 2 différences entre ces méthodes sont :

2/ “appendChild” ne peut pas insérer plusieurs éléments à la fois.... Alors que “append” oui !

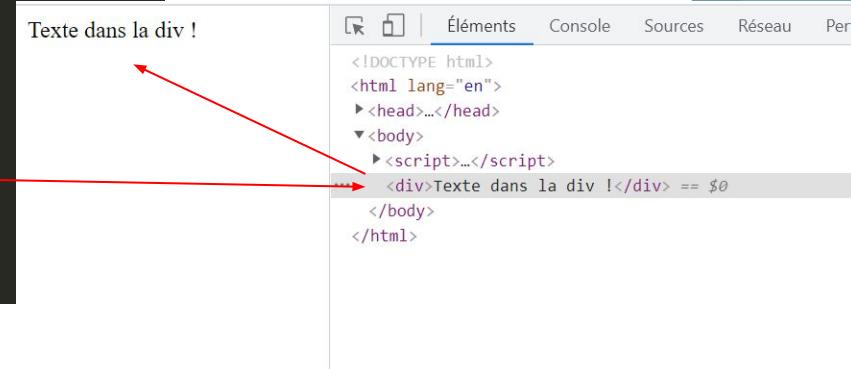
# Ajouter un texte dans une page via JavaScript

# Javascript - insérer un texte

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>

    <script>
      let div = document.createElement('div');
      div.innerText = "Texte dans la div !";
      document.body.append(div);

    </script>
  </body>
</html>
```



Pour mettre du texte dans notre “div”, avant son insertion, j’utilise la propriété “.innerText” à laquelle j’attribue une valeur (ma phrase)

# Javascript - insérer un texte

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

    <script>
        let div = document.createElement('div');
        div.textContent = "Texte dans la div !";
        document.body.append(div);

    </script>
</body>
</html>
```

Texte dans la div !

The screenshot shows the browser's developer tools with the 'Éléments' (Elements) tab selected. The DOM tree is displayed, starting with the <!DOCTYPE html> declaration, followed by the <html> element, then the <head> and <body> elements. Inside the <body> element, there is a <script> block and a <div> element. The <div> element has the text content 'Texte dans la div !'. A red arrow points from the highlighted line of code in the left panel to this <div> element in the DOM tree.

```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  <body>
    <script>...</script>
    ...   <div>Texte dans la div !</div> == $0
    </body>
  </html>
```

Il y a une seconde propriété qui permet l'ajout de texte, c'est “.textContent”. Cependant, il y a une légère différence entre ces 2 propriétés....

# Javascript - insérer un texte

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div>
        <span>Coucou</span>
        <span>Salut</span>
    </div>

    <script>
        // let div = document.createElement('div');
        // div.textContent = "Texte dans la div !";
        // document.body.append(div);

    </script>
</body>
</html>
```

Coucou Salut

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div>
        <span>Coucou</span>
        <span style="display:none">Salut</span>
    </div>

    <script>
        // let div = document.createElement('div');
        // div.textContent = "Texte dans la div !";
        // document.body.append(div);

    </script>
</body>
</html>
```

Coucou

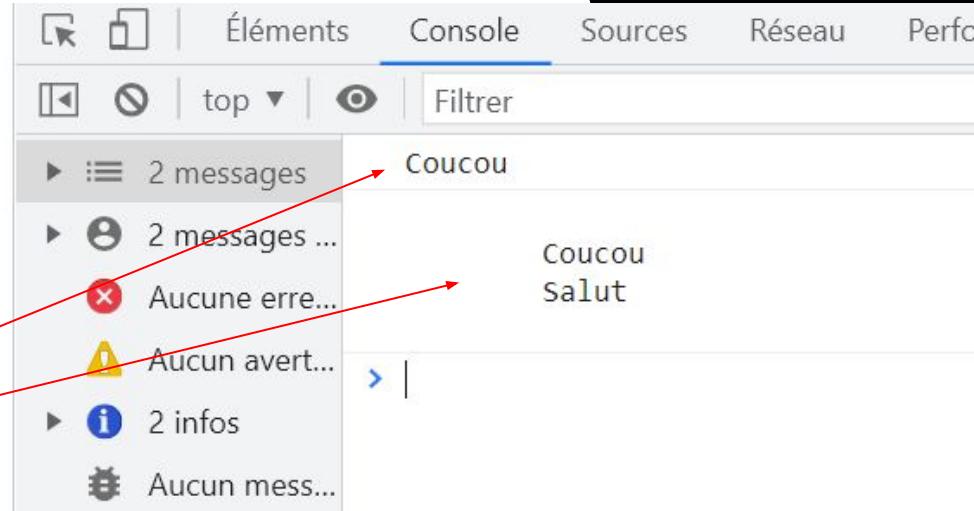
Pour mieux comprendre, je créé une “div” avec à l’intérieur 2 “span”. Dans chaque “span” j’écris un mot. La première “Coucou” et la deuxième “Salut”. Puis je mets un “display:none;” à la seconde “span”. Seul “Coucou” reste visible

# Javascript - insérer un texte

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div>
        <span>Coucou</span>
        <span style="display:none;">Salut</span>
    </div>

    <script>
        // let div = document.createElement('div');
        // div.textContent = "Texte dans la div !";
        // document.body.append(div);

        console.log(document.querySelector('div').innerText);
        console.log(document.querySelector('div').textContent);
    </script>
</body>
</html>
```



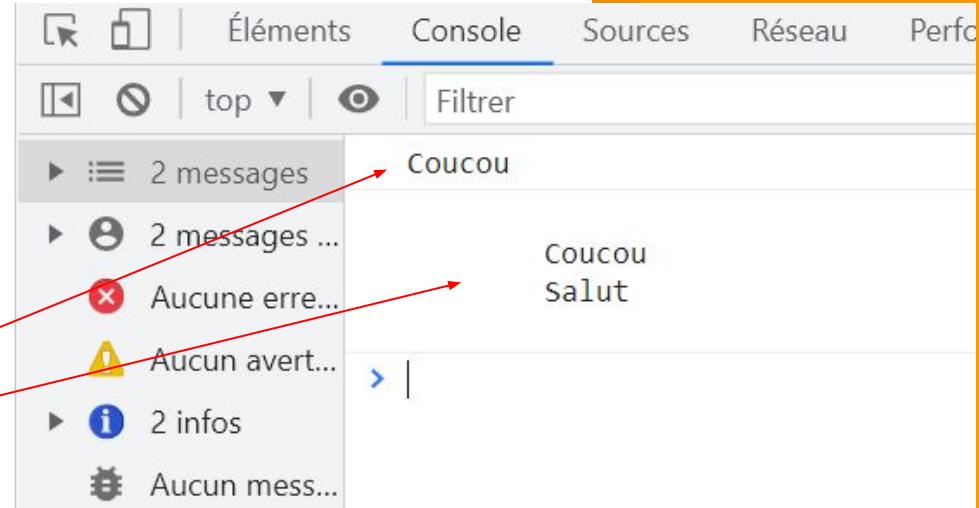
Deux choses à voir avant l'explication, la méthode “querySelector(“élémentAsélectionner”)” sert à sélectionner un élément de la page. A part le “body” et le “head”, il faudra forcément utiliser cette méthode pour sélectionner un élément. Cela fonctionne comme le CSS (querySelector('#monId .maClass')). Attention, cette méthode fonctionne si l’élément est unique que la page.

# Javascript - insérer un texte

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div>
        <span>Coucou</span>
        <span style="display:none;">Salut</span>
    </div>

    <script>
        // let div = document.createElement('div');
        // div.textContent = "Texte dans la div !";
        // document.body.append(div);

        console.log(document.querySelector('div').innerText);
        console.log(document.querySelector('div').textContent);
    </script>
</body>
</html>
```



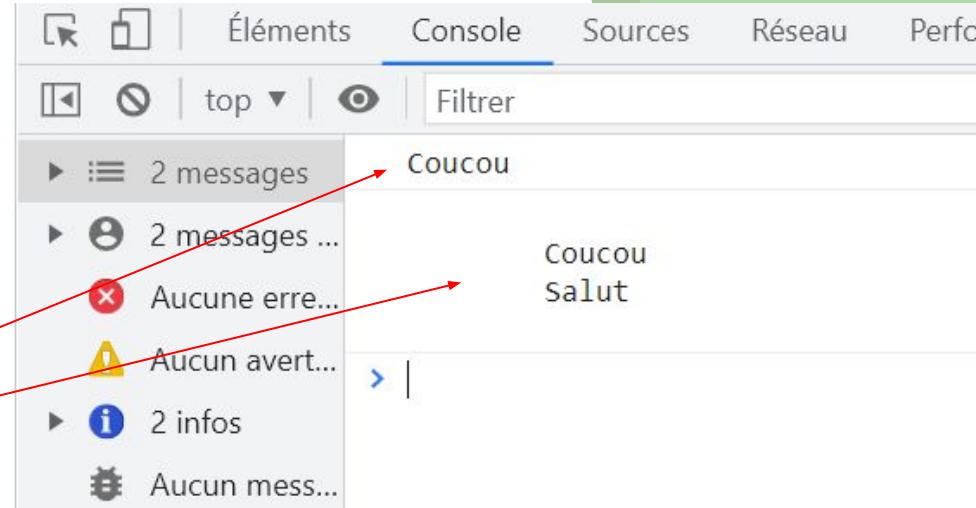
La deuxième chose c'est que, comme beaucoup d'autres propriétés, si ".textContent" ou ".innerText" n'ont pas d'assignation (Ex: `innerText = "un truc"`), au lieu d'ajouter un texte, ces propriétés vont récupérer un texte.

# Javascript - insérer un texte

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div>
        <span>Coucou</span>
        <span style="display:none;">Salut</span>
    </div>

    <script>
        // let div = document.createElement('div');
        // div.textContent = "Texte dans la div !";
        // document.body.append(div);

        console.log(document.querySelector('div').innerText);
        console.log(document.querySelector('div').textContent);
    </script>
</body>
</html>
```



Donc on peut voir que “.innerText” va récupérer le texte visible par l'utilisateur alors que “.textContent” va récupérer tout, même le texte caché par le style. De plus, il respecte l'indentation et la structure du HTML.

# Javascript - insérer un texte

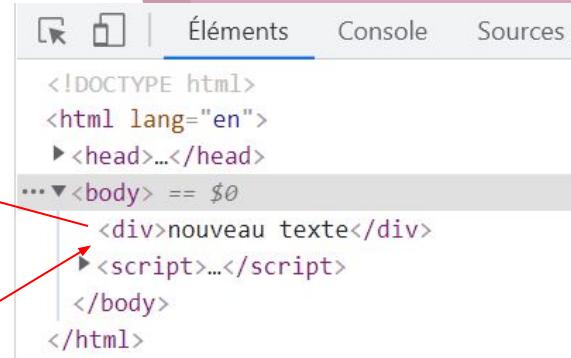
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=
      <title>Document</title>
    </head>
  <body>
    <div>
      <span>Coucou</span>
      <span style="display:none;">Salut</span>
    </div>

    <script>
      // let div = document.createElement('div');
      // div.textContent = "Texte dans la div !";
      // document.body.append(div);

      document.querySelector('div').innerText = "nouveau texte";

    </script>
  </body>
</html>
```

nouveau texte



The screenshot shows the browser's developer tools with the "Elements" tab selected. The DOM tree is displayed on the right, showing the following structure:

```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  ... <body> == $0
    <div>nouveau texte</div>
    <script>...</script>
  </body>
</html>
```

A red arrow points from the text "nouveau texte" in the code editor to the corresponding `nouveau texte` node in the DOM tree. Another red arrow points from the `document.querySelector('div').innerText = "nouveau texte";` line in the script editor to the same node in the DOM tree.

Bien entendu, si j'attribue une nouvelle valeur à la propriété “`innerText`” (ou “`textContent`”), il remplace le texte (et même le code) précédent.

# Ajouter du HTML dans une page via JavaScript

# Javascript - insérer du HTML

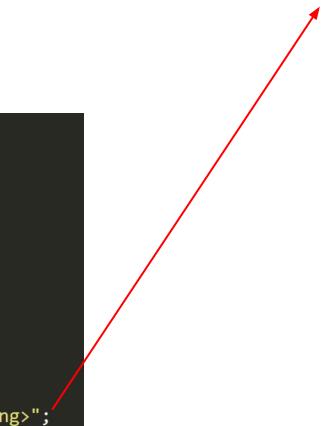
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div>
        <span>Coucou</span>
        <span style="display:none;">Salut</span>
    </div>

    <script>
        // let div = document.createElement('div');
        // div.textContent = "Texte dans la div !";
        // document.body.append(div);

        document.querySelector('div').innerText = "<strong>nouveau texte</strong>";

    </script>
</body>
</html>
```

<strong>nouveau texte</strong>



Maintenant, si je souhaite insérer ce même texte en gras, je serais tenté de mettre des balises “`<strong>`”....mais “`innerText`” (et “`textContent`”) ne gère pas les balises HTML.

# Javascript - insérer du HTML

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>
    <div>
      <span>Coucou</span>
      <span style="display:none;">Salut</span>
    </div>
    <script>
      document.querySelector('div').innerHTML = "<strong>nouveau texte</strong>";
    </script>
  </body>
</html>
```

nouveau texte



Pour l'insertion de balises HTML, on peut utiliser la propriété “innerHTML”  
ATTENTION ! Mal utilisée, cette propriété peut rendre vulnérable votre site et permettre à un hacker d'insérer un code malveillant.

# Javascript - insérer du HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <div></div>

  <script>

    let strong = document.createElement('strong');
    strong.innerText = "nouveau texte"
    document.querySelector('div').append(strong);

  </script>
</body>
</html>
```

nouveau texte

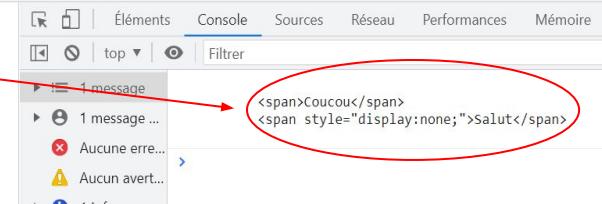
Une façon plus “safe” d’insérer du HTML dans une page est de créer un élément HTML en JS, puis d’insérer du contenu directement à l’intérieur. Enfin, insérer directement la balise HTML créée avec le contenu dans la “div”

# Javascript - insérer du HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div>
        <span>Coucou</span>
        <span style="display:none;">Salut</span>
    </div>

    <script>
        console.log(document.querySelector('div').innerHTML)
    </script>
</body>
</html>
```

Coucou



De la même façon que “innerText” ou “textContent”, si j’utilise “innerHTML” seul, cette méthode ira me récupérer le code situé dans un élément souhaité.

# **Supprimer du HTML dans une page via JavaScript**

# Javascript - supprimer du HTML

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>
    <div class="coucou">coucou</div>
    <div class="belette">belette</div>
    <div class="salut">salut</div>

    <script>
      const divBelette = document.querySelector('.belette');
      divBelette.remove();

    </script>
  </body>
</html>
```

coucou  
salut



```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  <body>
    <div class="coucou">coucou</div>
    ...
    <div class="salut">salut</div> == $0
    <script>...</script>
  </body>
</html>
```

La propriété “remove()” permet de détruire un élément HTML.  
Dans l'exemple ci-dessus, j'aurais pu écrire pour aller plus vite :  
document.querySelector('.belette').remove();

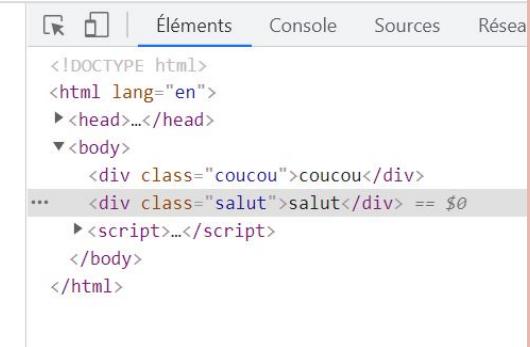
# Javascript - supprimer du HTML

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>
    <div id="container">
      <div class="coucou">coucou</div>
      <div class="belette">belette</div>
      <div class="salut">salut</div>
    </div>

    <script>
      const container = document.querySelector('#container');
      const divBelette = document.querySelector('.belette');
      container.removeChild(divBelette);

    </script>
  </body>
</html>
```

coucou  
salut



```
!DOCTYPE html
<html lang="en">
  <head>...</head>
  <body>
    <div class="coucou">coucou</div>
    ... <div class="salut">salut</div> == $0
    <script>...</script>
  </body>
</html>
```

Une autre méthode peut également détruire un élément HTML, c'est la méthode “removeChild(“élémentEnfantAdétruire””)”. Il faut l'utiliser directement sur l'élément parent et passer en paramètre, l'enfant à détruire. Autre syntaxe :  
`document.querySelector('#container').removeChild(document.querySelector('.belette'));`

# **Manipuler les attributs via JavaScript**

# Javascript - manipuler les attributs

The screenshot shows a browser developer tools window. On the left is the HTML code:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>
    <div id="container">
      <div id="test" class="coucou">coucou</div>
    </div>

    <script>
      const divCoucou = document.querySelector('.coucou');
      console.log(divCoucou.getAttribute('id'))
    </script>
  </body>
</html>
```

In the center is a preview of the page with the text "coucou". In the bottom right corner of the preview area, there is a small red arrow pointing towards the "Console" tab in the developer tools.

The "Console" tab is selected, showing the output of the script:

- 1 message
- 1 message ...
- Aucune erre...
- Aucun avert...
- 1 info
- Aucun mess...

The "info" message "test" is highlighted with a red arrow pointing from the preview area.

Pour lire un attribut, on utilise la méthode “`getAttribute('attributArécupérer')`” avec en paramètre l’attribut à lire. Dans l’exemple ci-dessus, l’attribut “`id`” de la “`divCoucou`”, est bien égal à “`test`”. Autre syntaxe possible : `document.querySelector('.coucou').getAttribute('id')`

# Javascript - manipuler les attributs

The screenshot shows a browser developer tools window. On the left, the DOM tree displays the HTML structure:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>
    <div id="container">
      <div id="test" class="coucou">coucou</div>
    </div>

    <script>
      const divCoucou = document.querySelector('.coucou');
      console.log(divCoucou.getAttribute('class'))
    </script>
  </body>
</html>
```

In the center, a preview window shows the text "coucou". To the right, the JavaScript console shows the output of the script:

```
coucou
coucou
1 message
1 message ...
Aucune erre...
Aucun avert...
1 info
```

Red arrows point from the code line `console.log(divCoucou.getAttribute('class'))` to both the preview window and the console output.

Si je change d'attribut, nous récupérons bien sa valeur.

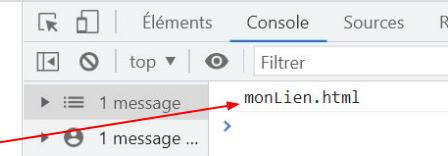
Autre syntaxe possible :

```
document.querySelector('.coucou').getAttribute('class')
```

# Javascript - manipuler les attributs

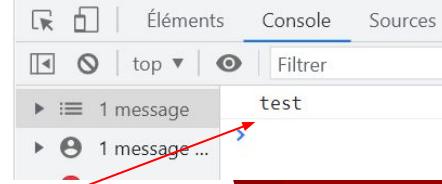
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>
    <div id="container">
      <div id="test" class="coucou">coucou</div>
      <a href="monLien.html">Lien</a>
    </div>
    <script>
      console.log(document.querySelector('#container a').getAttribute('href'));
    </script>
  </body>
</html>
```

coucou  
[Lien](#)



Dernier exemple sur un lien en sélectionnant son attribut “href”

# Javascript - manipuler les attributs



```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>
    <div id="container">
      <div id="test" data-test="hjhjh" class="coucou">coucou</div>
      <a class="lien" href="monLien.html">Lien</a>
    </div>

    <script>
      const divCoucou = document.querySelector('.coucou');
      console.log(divCoucou.id);
    </script>
  </body>
</html>
```

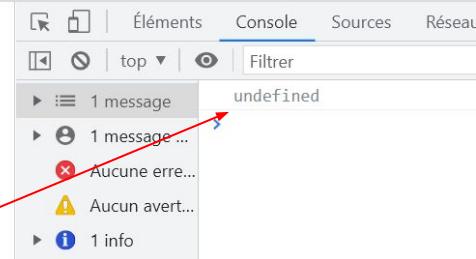
Il existe une syntaxe plus simple, c'est de mettre directement un point, puis l'attribut que vous cherchez à sélectionner ( ".id" pour l'id). Autre syntaxe : `document.querySelector('.coucou').id`

# Javascript - manipuler les attributs

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>
    <div id="container">
      <div id="test" class="coucou">coucou</div>
      <a class="lien" href="monLien.html">Lien</a>
    </div>

    <script>
      const divCoucou = document.querySelector('.coucou');
      console.log(divCoucou.class);
    </script>
  </body>
</html>
```

coucou  
[Lien](#)



Malheureusement, cette syntaxe ne fonctionne pas sur tous les attributs...

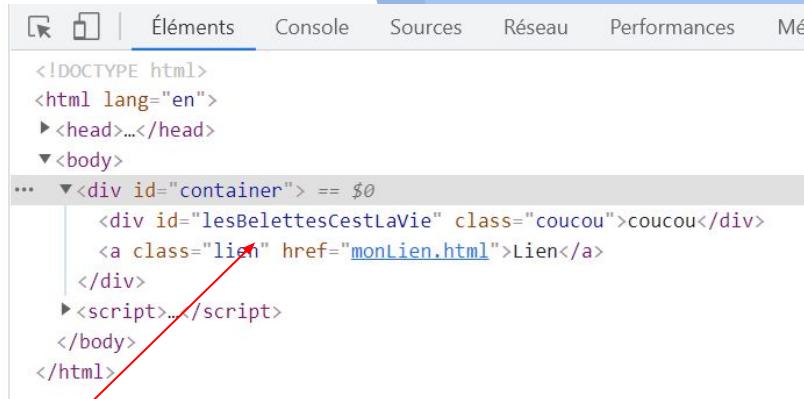
# Javascript - manipuler les attributs

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
  <title>Document</title>
</head>
<body>
  <div id="container">
    <div id="test" class="coucou">coucou</div>
    <a class="lien" href="monLien.html">Lien</a>
  </div>

  <script>
    document.querySelector('.coucou').setAttribute('id', 'lesBelettesCestLaVie');

  </script>
</body>
</html>
```

coucou  
Lien



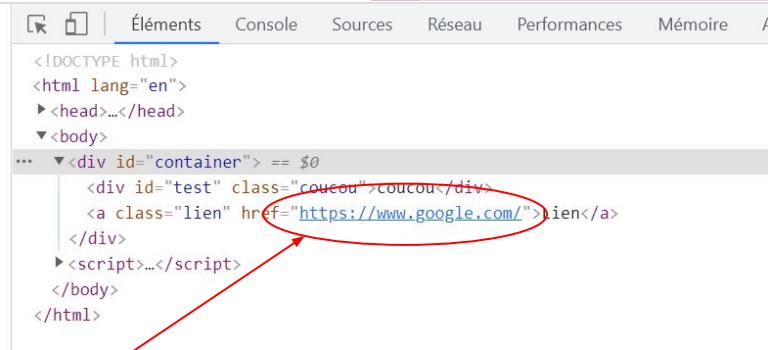
Pour modifier la valeur d'un attribut, il faut utiliser une autre méthode “`setAttribute('attributAchanger', 'nouvelleValeur')`” . Si l'attribut n'existe pas, il l'aurait créé.

# Javascript - manipuler les attributs

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, i
        <title>Document</title>
    </head>
    <body>
        <div id="container">
            <div id="test" class="coucou">coucou</div>
            <a class="lien" href="monLien.html">Lien</a>
        </div>

        <script>
            document.querySelector('a').setAttribute('href', 'https://www.google.com/');
        </script>
    </body>
</html>
```

coucou  
Lien



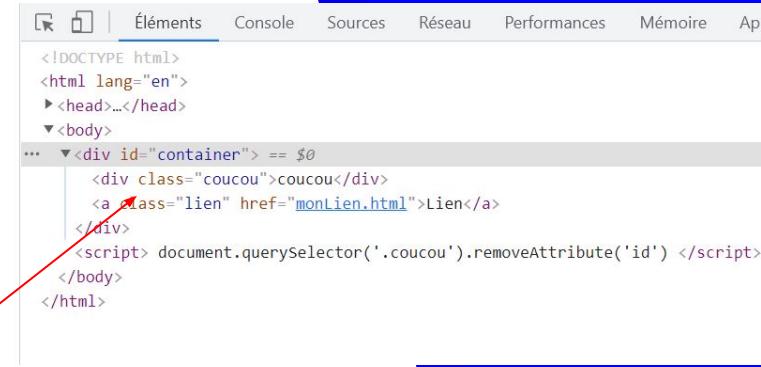
Dans cet exemple, je modifie la destination du lien.

# Javascript - manipuler les attributs

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>
    <div id="container">
      <div id="test" class="coucou">coucou</div>
      <a class="lien" href="monLien.html">Lien</a>
    </div>

    <script>
      document.querySelector('.coucou').removeAttribute('id')
    </script>
  </body>
</html>
```

coucou  
[Lien](#)



Bien sûr, si il est possible de récupérer, modifier un attribut, il est aussi possible de le supprimer avec la méthode "removeAttribute('élémentAdétruire')"

# Javascript - manipuler les attributs

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div id="container">
        <div id="test" class="coucou">coucou</div>
        <a data-test="test" data-quantite="2" class="lien" href="monLien.html">Lien</a>
    </div>
    | 
</body>
</html>
```

Il y a aussi la possibilité d'ajouter plus d'attributs à n'importe quels éléments (notamment quand le site commence à être complexe). Pour ce faire, il existe les “data-attributes”. Ce sont des attributs comme les autres, à part qu'ils commencent tous par “data-”.  
Syntaxe “data-nomQueVousVoulez = “votre valeur” ”

# Javascript - manipuler les attributs

```
<!DOCTYPE html>
<html lang="fr" dir="ltr" prefix="content: http://purl.org/rss/1.0/modules/content/ dc: http://purl.org/dc/terms/ foaf: http://xmlns.com/foaf/0.1/ og: http://ogp.me/ns#>
<head>
<meta charset="utf-8" />
<link rel="shortlink" href="https://www.mydigitalschool.com/" />
<link rel="canonical" href="https://www.mydigitalschool.com/" />
<meta name="description" content="MyDigitalSchool, l'école multimédia qui développe votre logique et votre raisonnement et stimule votre créativité et votre analyse." />
<meta name="Generator" content="Drupal 8 (https://www.drupal.org)" />
<meta name="MobileOptimized" content="width" />
<meta name="HandheldFriendly" content="true" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<script>
if (typeof Drupal !== 'undefined'){
if (!Drupal.eu_cookie_compliance.hasAgreed()){
(function(f,b,e,v,n,t,s){
if(f.fbq) return; n=f.fbq=n||function(){n.callMethod?
n.callMethod.apply(n,arguments):n.queue.push(arguments)};
if(t._fbq) f._fbq=t._fbq; n.push=n.loaded=0;n.version='2.0';
n.queue=[];t=b.createElement(e);t.async=0;
t.src=s;b.getElementsByTagName(e)[0];
s.parentNode.insertBefore(t,s)}(window,document,'script',
'https://connect.facebook.net/en_US/fbevents.js');
fbq('init', '432314497439006');
fbq('track', 'PageView');
}
}
</script>
<noscript>

</noscript>
<script>(function(w,d,s,l,i){w[l]=w[l]||[];w[l].push({'gtm.start':
new Date().getTime(),event:'gtm.js'});var f=d.getElementsByTagName(s)[0],
j=d.createElement(s),dl=l['dataLayer']?'&l=':'';j.async=true;j.src=
'https://www.googletagmanager.com/gtm.js?id='+i+dl;f.parentNode.insertBefore(j,f);
})(window,document,'script','dataLayer','GTM-M3GFBB4');
```

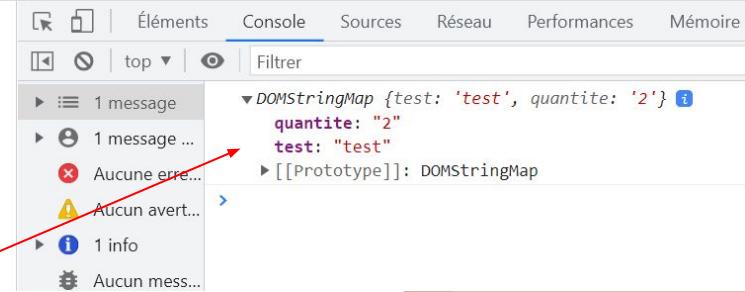
Dans cet exemple, les attributs "data" sont très utiles car ils permettent de mettre des informations récupérables en JavaScript

# Javascript - manipuler les attributs

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.
  <title>Document</title>
</head>
  <body>
    <div id="container">
      <div id="test" class="coucou">coucou</div>
      <a data-test="test" data-quantite="2" class="lien" href="monLien.html">Lien</a>
    </div>

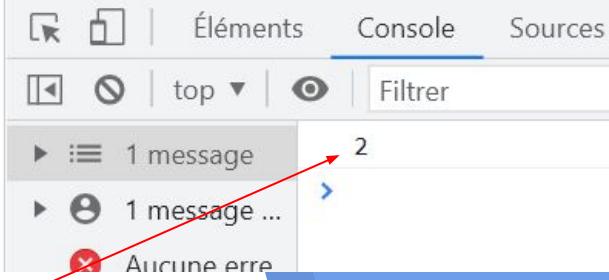
    <script>
      console.log(document.querySelector('a').dataset)
    </script>
  </body>
</html>
```

coucou  
[Lien](#)



Pour pouvoir accéder aux attributs “data”, il existe la propriété “dataset” qui n'est ni plus, ni moins un objet recensant tous les attributs “data” d'un élément choisi.

# Javascript - manipuler les attributs

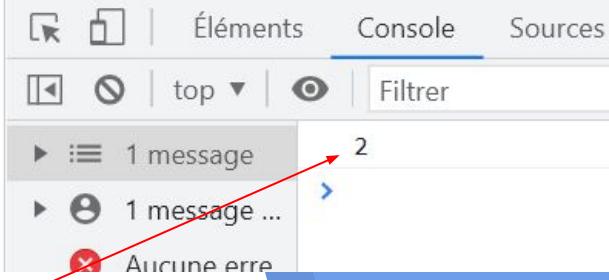


```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
  <title>Document</title>
</head>
<body>
  <div id="container">
    <div id="test" class="coucou">coucou</div>
    <a data-test="test" data-quantite="2" class="lien" href="monLien.html">Lien</a>
  </div>

  <script>
    console.log(document.querySelector('a').dataset.quantite)
  </script>
</body>
</html>
```

The code shows an HTML document with a container div containing a test div with the text "coucou" and a link with the text "Lien". The link has a data-quantite attribute set to "2". A script logs the value of this attribute to the console.

coucou  
Lien



Après "dataset", il me suffit de taper le nom de l'attribut "data" pour le récupérer.

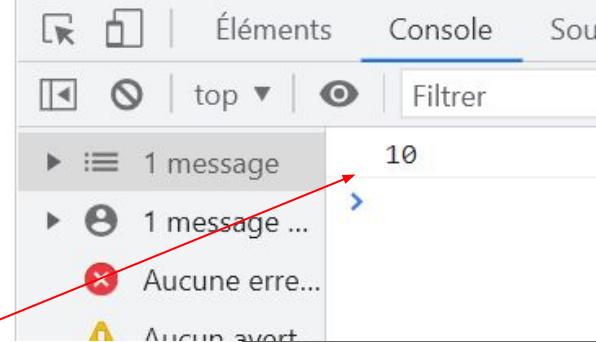
# Javascript - manipuler les attributs

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
      <title>Document</title>
    </head>
  <body>
    <div id="container">
      <div id="test" class="coucou">coucou</div>
      <a data-test="test" data-quantite="2" class="lien" href="monLien.html">Lien</a>
    </div>

    <script>
      document.querySelector('a').dataset.quantite = 10
      console.log(document.querySelector('a').dataset.quantite)

    </script>
  </body>
</html>
```

coucou  
[Lien](#)



Comme toutes propriétés, je peux lui assigner une nouvelle valeur....

# Javascript - manipuler les attributs

The screenshot shows a browser's developer tools with the "Éléments" (Elements) tab selected. The DOM tree on the left displays an HTML document structure. A red oval highlights the line of code in the script block below that adds a dataset attribute to the anchor tag.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>
    <div id="container">
      <div id="test" class="coucou">coucou</div>
      <a data-test="test" data-quantite="2" class="lien" href="monLien.html" data-belette="animal trop souvent oublé">Lien</a>
    </div>

    <script>
      document.querySelector('a').dataset.belette = "animal trop souvent oublé"
    </script>
  </body>
</html>
```

Je peux lui également créer un nouvel attribut (comme on crée une nouvelle clé dans un tableau et une nouvelle valeur)

# Javascript - manipuler les attributs

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, ini
      <title>Document</title>
    </head>
  <body>
    <div id="container">
      <div id="test" class="coucou">coucou</div>
      <a data-test="test" data-quantite="2" class="lien" href="monLien.html">Lien</a>
    </div>

    <script>
      document.querySelector('a').removeAttribute('data-quantite')

    </script>
  </body>
</html>
```

coucou  
[Lien](#)



Et pour supprimer un “data-attribut”, il suffit d’utiliser la méthode  
“removeAttribute(nomDuData-attributAdétruire)”

# Javascript - manipuler les attributs

The screenshot shows a browser's developer tools. On the left, the DOM tree is displayed with several nodes highlighted in red. A red arrow points from the highlighted 'class' attribute in the DOM tree to the corresponding line of code in the code editor on the right. The code editor contains the following HTML and JavaScript:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>
    <div id="container">
      <div id="test" class="coucou">coucou</div>
      <a class="lien class2" href="monLien.html">Lien</a>
    </div>

    <script>
      document.querySelector('a').setAttribute('class', 'maClass');
    </script>
  </body>
</html>
```

The DOM tree highlights the 'class' attribute of the 'a' element under the 'lien class2' class, and the 'class' attribute of the 'div' under the 'coucou' class.

Maintenant, voyons comment manipuler les “class” (car oui, c'est un peu spécial...). Dans l'exemple ci-dessus, on voit que mon élément “a” possède 2 “class”. Si je veux en ajouter une, je serais tenté d'utiliser la méthode “`setAttribute()`”. Malheureusement, cette méthode va juste supprimer les “class” existantes et les remplacer par l'autre...

# Javascript - manipuler les attributs

The screenshot shows a browser's developer tools. On the left, the DOM tree is displayed with several nodes highlighted in red. A red arrow points from the highlighted node in the DOM tree to the corresponding line of code in the code editor on the right. The code editor contains the following HTML and JavaScript:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>
    <div id="container">
      <div id="test" class="coucou">coucou</div>
      <a class="lien class2" href="monLien.html">Lien</a>
    </div>

    <script>
      document.querySelector('a').setAttribute('class', 'maClass');
    </script>
  </body>
</html>
```

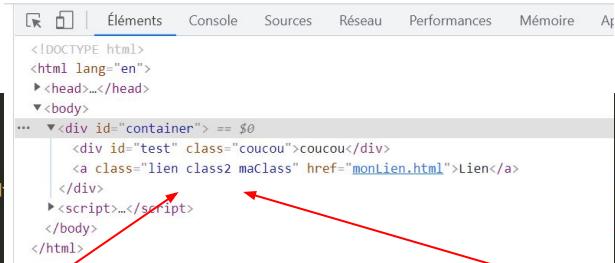
The DOM tree highlights the `a` element with the classes `lien` and `class2`. The code editor shows the addition of the `maClass` attribute to this element via a script.

Alors, vous allez me dire “oui mais quand il n'y a qu'une ‘class’, c'est possible...” Et c'est pas faux mais souvent il y a plusieurs “class” dans cet attribut....

# Javascript - manipuler les attributs

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <div id="container">
    <div id="test" class="coucou">coucou</div>
    <a class="lien class2" href="monLien.html">Lien</a>
  </div>

  <script>
    const classes = document.querySelector('a').getAttribute('class');
    document.querySelector('a').setAttribute('class', classes+' maClass');
  </script>
</body>
</html>
```



```
<body>
  <div id="container">
    <div id="test" class="coucou">coucou</div>
    <a class="lien class2" href="monLien.html">Lien</a>
  </div>

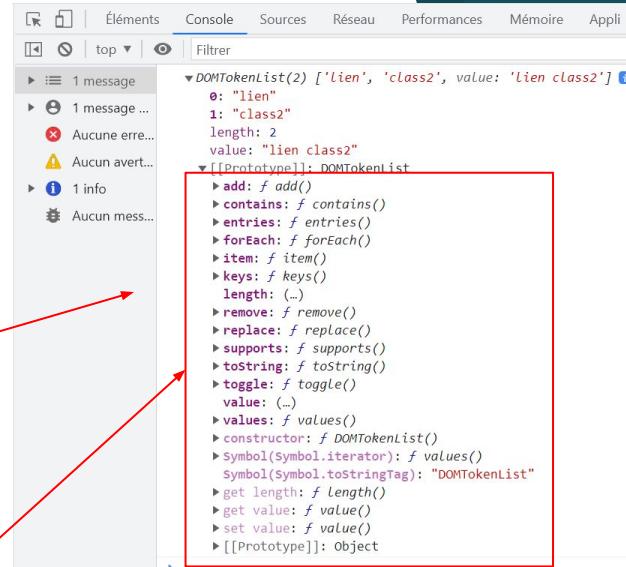
  <script>
    document.querySelector('a').setAttribute('class', 'lien class2 maClass');
  </script>
</body>
</html>
```

D'autres pourraient me dire “et si j'ajoute les “class” existantes plus ma nouvelle “class” dans la méthode “setAttribute()”.... Oui, ça pourrait passer....au début.....

# Javascript - manipuler les attributs

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div id="container">
        <div id="test" class="coucou">coucou</div>
        <a class="lien class2" href="monLien.html">Lien</a>
    </div>

    <script>
        console.log(document.querySelector('a').classList);
    </script>
</body>
</html>
```



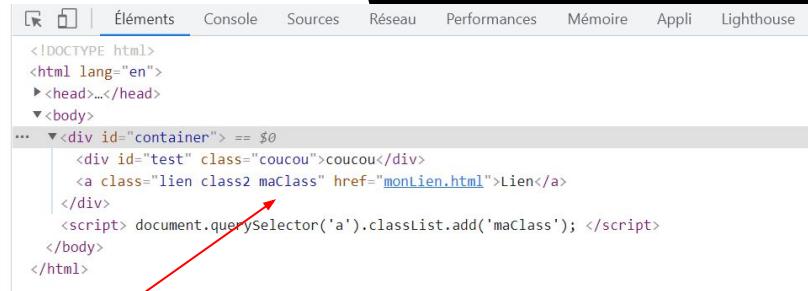
La propriété “classList” est là pour ça !  
Elle possède de nombreuses méthodes plus simples et très utiles !

# Javascript - manipuler les attributs

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-sca
    <title>Document</title>
</head>
<body>
    <div id="container">
        <div id="test" class="coucou">coucou</div>
        <a class="lien class2" href="monLien.html">Lien</a>
    </div>

    <script>
        document.querySelector('a').classList.add('maClass');
    </script>
</body>
</html>
```

coucou  
Lien



La méthode “add(“classAajouter”) ajoute simplement une “class” en plus (et ne détruit pas les “class” existantes)

# Javascript - manipuler les attributs

The screenshot shows a browser's developer tools with the "Éléments" (Elements) tab selected. The left panel displays the HTML code:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>
    <div id="container">
      <div id="test" class="coucou">coucou</div>
      <a class="lien class2" href="monLien.html">Lien</a>
    </div>

    <script>
      document.querySelector('a').classList.remove('class2');
    </script>
  </body>
</html>
```

The right panel shows the DOM tree under the body node:

- <html lang="en">
- <head>...</head>
- <body>
  - <div id="container"> == \$0
    - <div id="test" class="coucou">coucou</div>
    - <a class="lien class2" href="monLien.html">Lien</a>
- </body>
- </html>

A red arrow points from the line `document.querySelector('a').classList.remove('class2');` in the script to the class attribute of the anchor element in the DOM tree. Another red arrow points from the same line to the `remove` method call.

La méthode “remove(“classAdétaire”) supprime simplement une “class” (sans détruire les autres “class” existantes)

# Javascript - manipuler les attributs

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
      a.classToggle{
        color: red;
      }
    </style>
  </head>
  <body>
    <div id="container">
      <div id="test" class="coucou">coucou</div>
      <a class="lien class2" href="nonLien.html">Lien</a>
    </div>

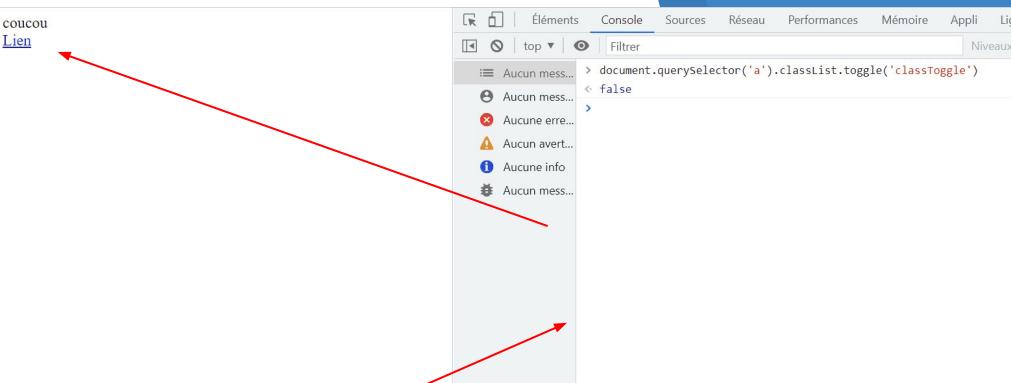
    <script>
      document.querySelector('a').classList.toggle('classToggle');
    </script>
  </body>
</html>
```

La méthode “`toggle(“classAjouterOuSupprimer”)`” est très utile ! Elle permet d’ajouter une “`class`” si elle n’existe pas OU de la supprimer si elle existe. Pour pouvoir voir son effet, j’ai ajouté du CSS qui dit ”Si la balise ‘`a`’ a la ‘`class`’ ‘`toggleClass`’ , alors le lien s’affichera en rouge, sinon, couleur de base....”

# Javascript - manipuler les attributs

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        a.classToggle{
            color: red;
        }
    </style>
</head>
<body>
    <div id="container">
        <div id="test" class="coucou">coucou</div>
        <a class="lien class2" href="monLien.html">Lien</a>
    </div>

    <script>
        document.querySelector('a').classList.toggle('classToggle');
    </script>
</body>
</html>
```



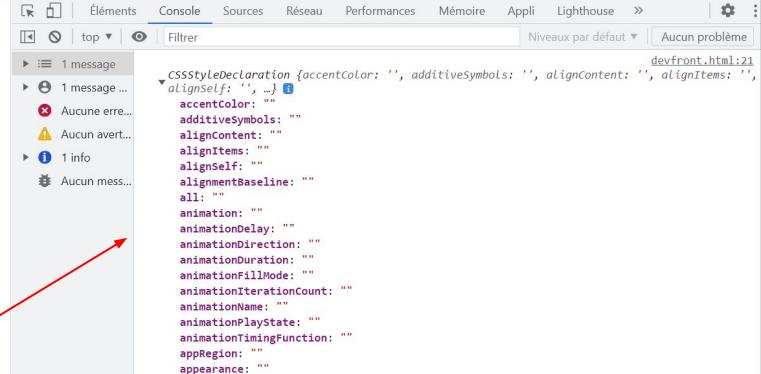
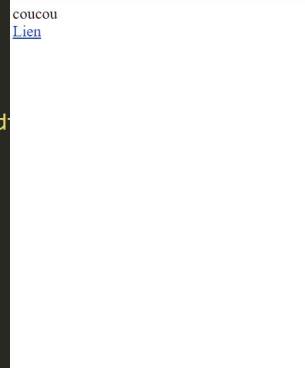
On voit bien que lorsque que j'exécute la méthode “toggle()”, le JS ajoute puis supprime la “class” “toggleClass” car le lien change de couleur.

# **Manipuler le style via JavaScript**

# Javascript - manipuler les attributs

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        a.classToggle{
            color: red;
        }
    </style>
</head>
<body>
    <div id="container">
        <div id="test" class="coucou">coucou</div>
        <a class="lien class2" href="monLien.html">Lien</a>
    </div>

    <script>
        console.log(document.querySelector('a').style);
    </script>
</body>
</html>
```

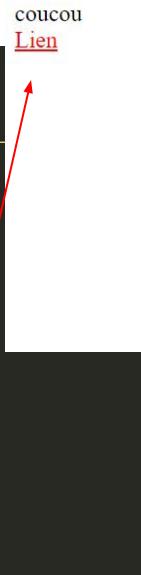


De même que JS nous met à disposition une propriété pour manipuler les “class”, il nous met également une propriété pour le style.....  
La propriété “style” vous donne accès à toutes les propriétés CSS que vous connaissez.

# Javascript - manipuler les attributs

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
    <title>Document</title>
    <style>
        .classToggle{
            color: red;
        }
    </style>
</head>
<body>
    <div id="container">
        <div id="test" class="coucou">coucou</div>
        <a class="lien class2" href="monLien.html">Lien</a>
    </div>

    <script>
        document.querySelector('a').style.color = "red";
    </script>
</body>
</html>
```



A screenshot of the browser's developer tools, specifically the "Elements" tab. It shows the DOM tree with a red circle highlighting the inline style "style="color: red;" on the anchor tag. The script section of the developer tools also shows the line of JavaScript code that adds this style.

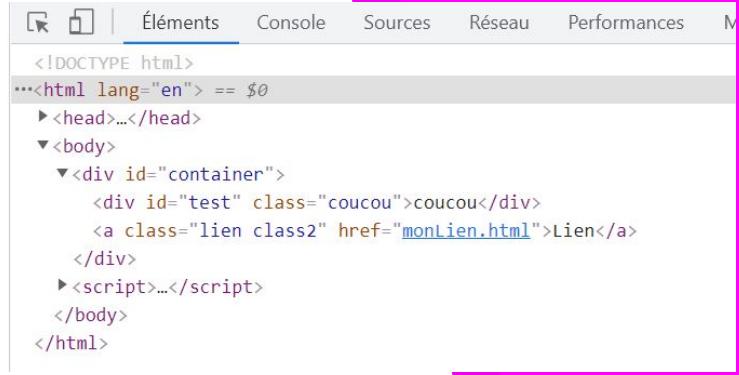
Il suffit simplement d'ajouter la propriété CSS que vous souhaitez modifier et lui assigner une nouvelle valeur. Exemple avec “color”.  
On peut voir dans la console qui lui a ajouté une balise “style”.

# Javascript - manipuler les attributs

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, i
<title>Document</title>
<style>
    .classToggle{
        color: red;
    }
</style>
</head>
<body>
    <div id="container">
        <div id="test" class="coucou">coucou</div>
        <a class="lien class2" href="monLien.html">Lien</a>
    </div>

    <script>
        document.querySelector('a').style.color = "red";
        document.querySelector('a').style.text-decoration = "none";
    </script>
</body>
</html>
```

coucou  
[Lien](#)

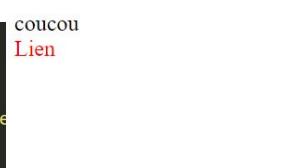


The screenshot shows the Chrome DevTools interface with the 'Elements' tab selected. The DOM tree is displayed, starting with the root <html> element. Inside the <body> element, there is a <div> with the id 'container'. Within 'container', there is another <div> with the id 'test' and class 'coucou', containing the text 'coucou'. Next to it is an anchor tag (<a>) with the class 'lien class2' and href 'monLien.html', labeled 'Lien'. A script block is also present in the body.

```
<!DOCTYPE html>
<html lang="en"> == $0
    <head>...</head>
    <body>
        <div id="container">
            <div id="test" class="coucou">coucou</div>
            <a class="lien class2" href="monLien.html">Lien</a>
        </div>
        <script>...</script>
    </body>
</html>
```

Si je veux lui modifier une valeur CSS qui s'écrive avec le symbole “-”, cela ne fonctionnera pas. Exemple “text-decoration”

# Javascript - manipuler les attributs



The screenshot shows a browser's developer tools element inspector. On the left is the HTML code, and on the right is the DOM tree. The 'lien' element is highlighted with a red oval. In the DOM tree, its 'style' attribute is set to 'color: red; text-decoration: none;'. A red arrow points from the 'text-decoration' part of the CSS in the code to this attribute in the DOM tree.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
      a.classToggle{
        color: red;
      }
    </style>
  </head>
  <body>
    <div id="container">
      <div id="test" class="coucou">coucou</div>
      <a class="lien class2" href="monLien.html">Lien</a>
    </div>

    <script>
      document.querySelector('a').style.color = "red";
      document.querySelector('a').style.textDecoration = "none";
    </script>
  </body>
</html>
```

coucou  
Lien



The screenshot shows a browser's developer tools element inspector. The 'lien' element is highlighted with a red oval. In the DOM tree, its 'style' attribute is set to 'color: red; text-decoration: none;'. A red arrow points from the 'text-decoration' part of the CSS in the code to this attribute in the DOM tree.

```
<!DOCTYPE html>
<html lang="en"> == $0
  <head>...</head>
  <body>
    <div id="container">
      <div id="test" class="coucou">coucou</div>
      <a class="lien class2" href="monLien.html" style="color: red; text-decoration: none;">Lien</a>
    </div>
    <script>...</script>
  </body>
</html>
```

Pour faire passer des propriétés CSS qui contiennent des “-”, vous devrez supprimer l'écrire en “chameau” (camel case). “text-decoration” deviendra “textDecoration”. “margin-top” deviendra “marginTop”. “padding-left” deviendra “paddingLeft”....

# Javascript - manipuler les attributs

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Document</title>
7   <style>
8     .classToggle{
9       color: red;
10    }
11   </style>
12 </head>
13 <body>
14   <div id="container">
15     <div id="test" class="coucou">coucou</div>
16     <a class="lien class2" href="monLien.html" style="color: white; text-decoration: none; background: black; padding: 10px 15px; display: block; width: 30px;">Lien</a>
17   </div>
18
19   <script>
20     document.querySelector('a').style.cssText = "color: white; text-decoration: none; background: black; padding: 10px 15px; display: block; width: 30px;";
21   </script>
22 </body>
23 </html>
```

coucou  
Lien



Si vous avez beaucoup de propriétés à modifier, écrire pour chaque propriété “document.querySelector('a').style.proprieteCSS = valeur” peut être loooonnnnnng ! Vous pouvez alors utiliser la propriété “cssText” et lui assigner comme valeur toutes les propriétés / valeurs que vous voulez (comme dans un attribut “style” classique)

# Javascript - manipuler les attributs

The screenshot shows a browser window with developer tools open. On the left, the page content displays "coucou" and a black button labeled "Lien". On the right, the element inspector shows the DOM structure. A red box highlights the "lien class2" link element, and a red arrow points from this element to the line of code in the script block that adds the "style" attribute.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        a.classToggle{
            color: red;
        }
    </style>
</head>
<body>
    <div id="container">
        <div id="test" class="coucou">coucou</div>
        <a class="lien class2" href="monLien.html">Lien</a>
    </div>

    <script>
        document.querySelector('a').setAttribute('style', 'color: white; text-decoration: none; background: black; padding: 10px 15px; display: block; width: 30px');
    </script>
</body>
</html>
```

Ou alors, on peut lui ajouter un attribut “style” avec la méthode “`setAttribute(“style”, “styleAajouter...”)`”

# **querySelector** **&** **querySelectorAll**

## Javascript - querySelector & querySelectorAll

```
document.querySelector('monElementASélectionner')
document.querySelector('monID')
document.querySelector('maClass')
document.querySelector('#id .maClass p')
```

Nous avons vu plus tôt que la méthode “querySelector(“element”)” permet de sélectionner un élément (par sa “class”, son “ID”, son type d’élément HTML,...) présent dans la page.

## Javascript - querySelector & querySelectorAll

```
document.getElementById('JeDoisForcementMettreUnID')
document.getElementsByClassName('jeDoisForcementMettreUneClass')
document.getElementsByTagName('jeDoisForcementMettreUnElementHTML')
```

Avant la méthode “querySelector(“element”)\”, nous avions une méthode pour sélectionner soit une “class”, soit un “ID”, soit un élément HTML. La méthode “querySelector()” est nouvelle par rapport aux autres.

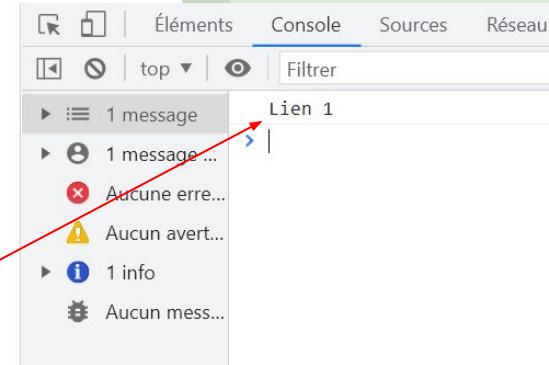
# Javascript - querySelector & querySelectorAll

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

    <a href="#" class="unLien">Lien 1</a>
    <a href="#" class="unLien">Lien 2</a>
    <a href="#" class="unLien">Lien 3</a>
    <a href="#" class="unLien">Lien 4</a>
    <a href="#" class="unLien">Lien 5</a>

    <script>
        console.log(document.querySelector('a.unLien').innerText);
    </script>
</body>
</html>
```

[Lien 1](#) [Lien 2](#) [Lien 3](#) [Lien 4](#) [Lien 5](#)



Le problème avec la méthode “querySelector()” est que si je souhaite sélectionner plusieurs éléments.....bah je peux pas. Cette méthode s’arrête au premier lien. Alors comment faire pour afficher tous les textes présent dans chaque lien ?

# Javascript - querySelector & querySelectorAll

The screenshot shows a browser's developer tools open to the 'Console' tab. A red arrow points from the explanatory text below to the 'script' block in the code editor on the left, and another red arrow points from the 'script' block to the NodeList object in the console output on the right.

Lien 1 Lien 2 Lien 3 Lien 4 Lien 5

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <a href="#" class="unLien">Lien 1</a>
    <a href="#" class="unLien">Lien 2</a>
    <a href="#" class="unLien">Lien 3</a>
    <a href="#" class="unLien">Lien 4</a>
    <a href="#" class="unLien">Lien 5</a>

<script>
    console.log(document.querySelectorAll('a.unLien'));
</script>
</body>
</html>
```

Console output:

```
1 message
1 message ...
Aucune erreur...
Aucun avertissement...
1 info
Aucun message.

▼ NodeList(5) [a.unLien, a.unLien, a.unLien, a.unLien, a.unLien]
▶ 0: a.unLien
▶ 1: a.unLien
▶ 2: a.unLien
▶ 3: a.unLien
▶ 4: a.unLien
length: 5
[[Prototype]]: NodeList
```

La solution est la méthode “querySelectorAll(‘groupeDelements’)” car cette méthode va générer un tableau de tous ces éléments

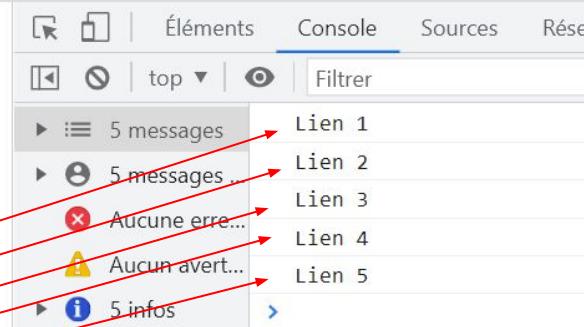
# Javascript - querySelector & querySelectorAll

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

    <a href="#" class="unLien">Lien 1</a>
    <a href="#" class="unLien">Lien 2</a>
    <a href="#" class="unLien">Lien 3</a>
    <a href="#" class="unLien">Lien 4</a>
    <a href="#" class="unLien">Lien 5</a>

<script>
    console.log(document.querySelectorAll('a.unLien')[0].innerText);
    console.log(document.querySelectorAll('a.unLien')[1].innerText);
    console.log(document.querySelectorAll('a.unLien')[2].innerText);
    console.log(document.querySelectorAll('a.unLien')[3].innerText);
    console.log(document.querySelectorAll('a.unLien')[4].innerText);
</script>
</body>
</html>
```

[Lien 1](#) [Lien 2](#) [Lien 3](#) [Lien 4](#) [Lien 5](#)



Comme n'importe quel tableau, je peux mettre un index pour sélectionner l'index de l'élément que je souhaite sélectionner pour ensuite récupérer le texte. Bon, là c'est marrant, j'ai 5 éléments, mais si j'en avais 300, je vais pas les faire un par un. Par hasard, on connaît pas moyen lus rapide de parcourir un tableau ??????????

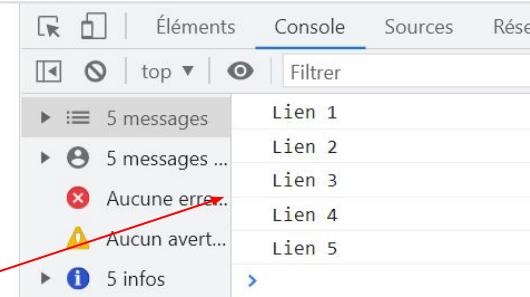
# Javascript - querySelector & querySelectorAll

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>

    <a href="#" class="unLien">Lien 1</a>
    <a href="#" class="unLien">Lien 2</a>
    <a href="#" class="unLien">Lien 3</a>
    <a href="#" class="unLien">Lien 4</a>
    <a href="#" class="unLien">Lien 5</a>

  <script>
    const tableauDeLiens = document.querySelectorAll('a.unLien');
    tableauDeLiens.forEach(function(element, index) {
      console.log(element.innerText);
    });
  </script>
</body>
</html>
```

Lien 1 Lien 2 Lien 3 Lien 4 Lien 5



La boucle forEach !!! (ou autre du moment que c'est une boucle)

# Javascript - querySelector & querySelectorAll

```
//Je stocke tous mes éléments dans un tableau "tableauDeLiens"  
const tableauDeLiens = document.querySelectorAll('a.unLien');  
//Je parcours ce tableau  
tableauDeLiens.forEach( function(element, index) {  
    // "element" est égal à la ligne du tableau en cours (  
    document.querySelectorAll('a.unLien')[indexDeLaLigne])  
    //Avec la propriété "innerText", on va récupérer le texte présent dans le lien ligne par ligne  
    console.log(element.innerText);  
});
```

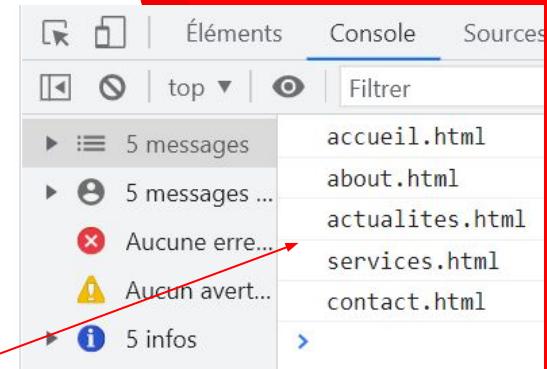
## Explications

# Javascript - querySelector & querySelectorAll

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, ini
<title>Document</title>
</head>
<body>
    <a href="accueil.html" class="unLien">Lien 1</a>
    <a href="about.html" class="unLien">Lien 2</a>
    <a href="actualites.html" class="unLien">Lien 3</a>
    <a href="services.html" class="unLien">Lien 4</a>
    <a href="contact.html" class="unLien">Lien 5</a>

    <script>
        const tableauDeLiens = document.querySelectorAll('a.unLien');
        tableauDeLiens.forEach( element, index) {
            console.log(element.getAttribute('href'));
        };
    </script>
</body>
</html>
```

Lien 1 Lien 2 Lien 3 Lien 4 Lien 5



Autre exemple où on utilise la méthode “getAttribute()” pour sélectionner le texte de l'attribut “href”

# Javascript - querySelector & querySelectorAll

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

    <a href="accueil.html" class="unLien">Lien 1</a>
    <a href="about.html" class="unLien">Lien 2</a>
    <a href="actualites.html" class="unLien">Lien 3</a>
    <a href="services.html" class="unLien">Lien 4</a>
    <a href="contact.html" class="unLien">Lien 5</a>

    <script>
        const tableauDeLiens = document.querySelectorAll('a.unLien');
        tableauDeLiens.forEach( element, index) {
            element.setAttribute('href', element.innerText)
        });
    </script>

</body>
</html>
```

[Lien 1](#) [Lien 2](#) [Lien 3](#) [Lien 4](#) [Lien 5](#)

The screenshot shows the browser's developer tools with the "Éléments" (Elements) tab selected. The DOM tree is displayed, starting with the <html> tag. Inside the <body> tag, there are five anchor elements (a) each with a href attribute set to "Lien 1" through "Lien 5" respectively, and a class attribute set to "unLien". A script tag is also present in the body.

```
<!DOCTYPE html>
<html lang="en">
    <head>...</head>
    <body>
        <a href="Lien_1" class="unLien">Lien 1</a>
        <a href="Lien_2" class="unLien">Lien 2</a>
        <a href="Lien_3" class="unLien">Lien 3</a>
        <a href="Lien_4" class="unLien">Lien 4</a>
        <a href="Lien_5" class="unLien">Lien 5</a>
        <script>...</script>
    </body>
</html>
```

Dans l'exemple ci-dessus, je récupère le texte cliquable de chaque lien et je l'insère dans chaque attribut “href”

# **Les écouteurs d'événements**

## Javascript - EventListener

```
Element_a_ecouter.addEventListener('action', function(e){  
    //Code à exécuter quand action à lieu  
})
```

# Javascript - EventListener

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>
    <div class="lien">Coucou</div>

    <script>
      const div = document.querySelector('div.lien');
      div.addEventListener('click', function(e){
        alert(div.innerText);
      })
    </script>
  </body>
</html>
```

Coucou

Dans cet exemple, on sélectionne l'élément HTML “div” que l'on stocke dans une variable “div”. Ensuite, on utilise la méthode d'écoute d'événements “addEventListener('action', 'function callback')” avec comme action “click” (donc quand je clique sur la “div”) et une fonction anonyme qui sera exécutée au moment où la “div” sera cliquée (donc lancer une “alert” avec comme message le texte présent dans la “div”)

# Javascript - EventListener

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>
    <div class="lien">Coucou</div>

    <script>
      const div = document.querySelector('div.lien');
      div.addEventListener('mouseover', function(e){
        alert(div.innerText);
      })
    </script>
  
```

Console

L'événement “mouseover” est l'équivalent de “hover” en CSS

# Javascript - EventListener

```
<option value="Wallis and Futuna">Wallis and Futuna</option>
<option value="Western Sahara">Western Sahara</option>
<option value="Yemen">Yemen</option>
<option value="Zambia">Zambia</option>
<option value="Zimbabwe">Zimbabwe</option>

</select>

<script>
    document.querySelector('select').addEventListener('change', function(e){
        alert(document.querySelector('select').value);
    })
</script>

</body>
</html>
```

Afghanistan ▾

L'événement “change” détecte le moindre changement sur un élément...

Vous avez la liste des événements :

<https://developer.mozilla.org/fr/docs/Web/Events>

# Javascript - EventListener

```
<script>
    //Attend que la partie HTML soit chargé pour être exécuté
    document.addEventListener('DOMContentLoaded', function(){
        //Tout le code ici sera exécuté quand le reste de la page sera chargée

        document.querySelector('select').addEventListener('change', function(e){
            alert(document.querySelector('select').value);
        })

    })
</script>
```

D'ailleurs, très important, tout le code JS doit être à l'intérieur d'un écouteur déclenché par l'événement “DOMContentLoaded”. Cela évite les erreurs de chargement. Imaginons que le JS charge un écouteur d'événement sur un élément HTML qui n'est pas encore chargé.... Vous aurez une erreur !

# Javascript - EventListener

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>

    <a href="#" class="lien">Lien 1</a>
    <a href="#" class="lien">Lien 2</a>
    <a href="#" class="lien">Lien 3</a>
    <a href="#" class="lien">Lien 4</a>
    <a href="#" class="lien">Lien 5</a>

  </body>
</html>

<script>
  document.addEventListener('DOMContentLoaded', function(){

    document.querySelector('a.lien').addEventListener('change', function(e){
      e.preventDefault();
      alert(document.querySelector('a.lien').innerText);
    })
  })
</script>
```

On a vu comment écouter un élément HTML, mais comment écouter plusieurs éléments HTML ????  
Dans l'exemple ci-dessus, rien ne fonctionnera.... :(

Pour info, la méthode “preventDefault()” annule le comportement par défaut d'un élément. (ex: un lien cliqué nous envoie vers une autre page, avec cette méthode, il se passera rien)

# Javascript - EventListener

```
document.addEventListener('DOMContentLoaded', fu  
  
    //Je mets mon groupe d'élément dans un tableau  
    const a = document.querySelectorAll('.lien');  
    //Je parcours mon tableau grâce à ma boucle forEach  
    a.forEach( function(element, index) {  
        //Sur chaque élément, je mets un écouteur d'événement au "click"  
        element.addEventListener('click',function(e){  
            //Je bloque le comportement par défaut du lien  
            e.preventDefault();  
            //Je lance une "alert" avec comme texte le texte cliquable.  
            alert(this.innerText);  
        })  
    });  
})  
</script>
```

Tadaaaa ! Et là, vous vous dîtes..... "PU\*\*\*\*, c'est quoi ce "this" encore..."

## Javascript - EventListener

```
<a href="#" class="lien">Lien 1</a>
<a href="#" class="lien">Lien 2</a>
<a href="#" class="lien">Lien 3</a>
<a href="#" class="lien">Lien 4</a>
<a href="#" class="lien">Lien 5</a>
```

Je suis obligé d'utiliser ‘this’, car il détermine quel élément parmis tous ceux qui sont sur “écoute” a été cliqué. Il permet de sélectionner précisément l’élément qui a reçu le “click” et donc d’en extraire le texte cliquable (l’ancre)