Boucle While / For / ForEach / For ...of / For ...in / map

Comme dans tous les langages de programmation Javascript a un système de boucle, de base cela va nous permettre de répéter des instructions de code selon une condition. Les boucles vont également s'avérer utile par la suite, pour parcourir des itérables comme des tableaux ou des objets.

While

Correspond à répéter une ou plusieurs instructions TANT QUE une condition est vraie.

```
let unIndex = 0;
while (unIndex < 10) {
  console.log("Le Nombre : " + unIndex);
  unIndex++;
};
```

Ci-dessus on a un index initialisé à 0 et TANT QUE cet index est strictement inférieur à 10 ALORS, on va faire un console.log(), puis ne pas oublier d'incrémenter l'index pour pouvoir passer à une itération de boucle suivante.

```
Le Nombre : 0    app.js:20
Le Nombre : 1    app.js:20
Le Nombre : 2    app.js:20
Le Nombre : 3    app.js:20
Le Nombre : 4    app.js:20
Le Nombre : 5    app.js:20
Le Nombre : 6    app.js:20
Le Nombre : 7    app.js:20
Le Nombre : 8    app.js:20
Le Nombre : 9    app.js:20
```

FOR

Autre manière de créer des boucles, avec for(), dans les paramètres on va pouvoir directement initialiser un index, définir une condition et incrémenter l'index dans l'exemple ci-dessous nous allons faire une boucle visant à parcourir chaque case d'un tableau pour l'afficher en console.

```
let listeFilm = ['Ultime Décision','Mission Alcatraz','Attack Force'];
//? Boucle for, on définit un index (ici c'est i),
//? puis on définit une condition qui va définir le nombre de fois que le code dans la boucle sera éxecuter
//? puis on définit comment on passe à la prochaine itération de la boucle (ici i++, on augmente de 1 le numero de la case que l'on console.log)
for(i=0;i<listeFilm.length;i++){
    console.log('boucle FOR : ',listeFilm[i]);
};
```

```
boucle FOR :  Ultime Décision     app.js:14
boucle FOR :  Mission Alcatraz    app.js:14
boucle FOR :  Attack Force        app.js:14
```

ForEach

Une autre alternative, la fonction forEach de JS automatise le parcours d'un tableau ou objet (sans que l'on ait à gérer un système d'indexation (i++))
forEach va prendre en paramètre une fonction, cette même fonction pourra avoir un paramètre qui correspondra à chaque case parcourue. (Généralement dans la parenthèse de forEach on passe une fonction fléchée).

```
let listeFilm = ['Ultime Décision','Mission Alcatraz','Attack Force'];
//? La méthode forEach() permet d'exécuter une fonction donnée sur chaque élément du tableau.
// ? On va choisir une variable temporaire pour parcourir les elements du tableau
listeFilm.forEach(unFilm => console.log('boucle forEach Tableau : ',unFilm));
```

Ici chaque case du tableau sera stockée temporairement dans unFilm.

```
boucle forEach Tableau :  Ultime Décision          app.js:27
boucle forEach Tableau :  Mission Alcatraz         app.js:27
boucle forEach Tableau :  Attack Force             app.js:27
```

For … of

Encore une alternative pour parcourir des variables tableaux (et autre) c'est la boucle for … of, qui de la même manière que dans l'exemple précèdent dans lequel on va définir une variable temporaire pour parcourir chaque case du tableau :

```
for(let unElementTablo of listeFilm){
    console.log('boucle FOR...OF : ',unElementTablo);
};
```

```
boucle FOR...OF :  Ultime Décision                app.js:35
boucle FOR...OF :  Mission Alcatraz               app.js:35
boucle FOR...OF :  Attack Force                   app.js:35
```

For … in

Si l'on prend le cas des objets JS propose aussi un équivalent à for of (pour les variables de type Array), les boucles for in qui ont exactement la même utilisation que l'exemple précédent

```
const userData = {
    name: 'John Doe',
    email: 'john.doe@example.com',
    age: 25,
    dob: '08/02/1989',
    active: true
};
```

Il faut définir une variable temporaire qui stockera les clés (propriétés) de l'objet
Ici durant le parcours de l'objet chaque propriété ou clé seront stockées temporairement dans la

```
// on définit une variable temporaire pour parcourir le objet :)
for(let cleObjet in userData){
    console.log(`boucle FOR...IN (objet) : clé:${cleObjet} - valeur :  $
{userData[cleObjet]} `);
};
```

variable cleObjet.
Rappel : pour accéder aux propriétés d'un objet on la notation en tableau associatif unObjet[quelque chose]

```
boucle FOR...IN (objet) : clé:name - valeur :  John Doe
boucle FOR...IN (objet) : clé:email - valeur :  john.doe@example.com
boucle FOR...IN (objet) : clé:age - valeur :  25
boucle FOR...IN (objet) : clé:dob - valeur :  08/02/1989
boucle FOR...IN (objet) : clé:active - valeur :  true
```

Convertir des objets en tableaux

Depuis sa version ES8 JS propose des fonctions utilisables sur les Objets qui vont pouvoir transformer leurs clés et ou leurs valeurs sous forme de tableau (pour utiliser les $f°$ forEach ou map par exemple).

```
//? Parcourir les Objet  (Depuis javaScript ES8)
//? La Method .keys() qui convertit les clés d'un objet en tableau
//? La Method .values() qui convertit les valeurs d'un objet en tableau
//? La Method .entries() qui renvoit un tableau dans un tableau pour combiner clé - valeur
const keyUser = Object.keys(userData);
console.log("les clé de l'objet converties en array : ",keyUser);

const valuesUser = Object.values(userData);
console.log("les valeur de l'objet converties en array : ",valuesUser);

const convertedDataUser = Object.entries(userData);
console.log("les entrées de l'objet converties en array : ",convertedDataUser);
```

```
// De fait, une fois les objets convertis en tableau  on peut ruser et utiliser forEach par exemple :
valuesUser.forEach((lesValeurs)=>{
    console.log(`FOREACH avec objet converti en tableau chaque valeurs : ${lesValeurs}`);
});
convertedDataUser.forEach(([key, value])=>{
    console.log(`FOREACH avec objet converti en tableau : ${key}: ${value}`);
});
```

Exercice : boucles

```
// TODO :JS map phase 1
// TODO : côté template html rajouter plein de <p></p>
// TODO :On va récupérer TOUS les <p> de notre page dans une
variable lesTxt via getElementsByTagName
// TODO :On va faire un console log de lesTxt
```

```
//TODO JS map Phase 2
//TODO Avec la methode Array.from(), dans une nouvelle variable
textesTab on va transformer notre htmlCollection en array
//TODO On console log la variables textesTab
//* On transforme le HTMLCollection(qui contient tous nos <p>) en
Array classique
```

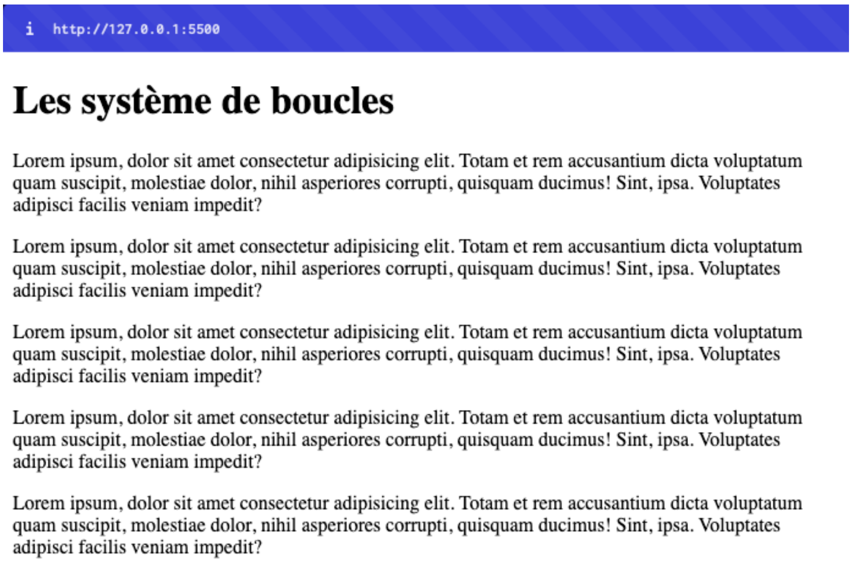
```
//TODO JS Map Phase 3 (on peut travailler sur un Array)
//TODO Sur textesTab on va utiliser la f° map(),
//TODO dans map(), on va lui passer en param une fonction fléchée
qui elle a en parametre une variable temporaire
(nom de la variable au choix)
//TODO cette fonction fléchée elle va modifier le innerHTML ou
```

Solution

```
const lesTxt = document.body.getElementsByTagName("p");
console.log(lesTxt);
```

```
//* On transforme le HTMLCollection(qui contient tous nos <p>) en
Array classique
const textesTab = Array.from(lesTxt);
console.log(textesTab);
```

```
textesTab.map(uneCase => uneCase.innerHTML = "LOL JE SUIS
HACKERMAN" );
```



https://github.com/jefff404/cours-js/tree/10-boucle

Point sur var let ou const

En JavaScript de base pour déclarer une variable on utilise le mot « var », mais avec la version ES6 on a eu deux autres manières de déclarer des variables.

(Var cela permet de voir si un tutoriel commence à dater)

En fait var peut poser problèmes avec la notion de scope que l'on a vu précédemment

Rappel scope : Jusqu'où notre variable va être disponible

Avec let et const on gère le scope en fonction des blocs dans lesquels on se situe.
(Scope de bloc)

C'est plus de contraintes mais ça permet de garder une logique et un code plus propre
Avec sa versionES6 JS introduit des nouvelles manières de déclarer des variables

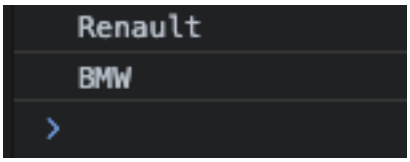
---

Exercice : Quiz Var

Je suis stagiaire dans votre entreprise (sous traitant Nasa) et je vous envoi ce code en « revue de code »
Que me répondez-vous ?

```
var voiture = "Renault";
console.log(voiture);
var voiture = "BMW";
console.log(voiture);
```

(Je vous jure cela fonctionne)



Cela fonctionne mais ce n'est pas correct dans la logique.
JS est un langage très permissif, mais cela peut engendrer des problèmes.

Exercice : Quizz

```
//!Quizz : ca bug
console.log(bolide);
let bolide = 'Jaguar'
```

```
Uncaught ReferenceError: Cannot access 'bolide' before initialization                app.js:12
    at app.js:12:13
  (anonymous) @ app.js:12
```

Exercice : Quizz

```
function choixVoiture(){
    var uneVoiture = "Harley Davidson"
}

choixVoiture();
console.log(uneVoiture);
```

```
Uncaught ReferenceError: uneVoiture is not defined                app.js:22
    at app.js:22:13
```

Solution

Pour le quizz précédent
Erreur : La var voiture est déclaré au sein de ma fonction et ne peut pas être utilisée en dehors.

```
var uneVoiture = "Harley Davidson"
function choixVoiture(){
}

choixVoiture();
console.log(uneVoiture);
```

Ici ça fonctionne car var est déclaré au-dessus.

Exercice : Quizz

Ca fonctionne, pour vous est-ce normal ?

```
var car = "Nissan";

if(car=="Nissan"){
    var vitesse = 800;
}
console.log(vitesse);
```

```
800                                                    app.js:38
```

Var est basé sur un scope de fonction uniquement
If il ne le considère pas comme une fonction du coup ça fonctionne.

Si l'on y prête pas attention ce genre de soucis peut engendrer de plus gros problèmes quand le code de vos applications deviendra plus complexe

```
var theCar = "Nissan";

if(theCar=="Nissan"){
    let speed = 800;
}
console.log(speed);
```

```
▶Uncaught ReferenceError: speed is not defined                app.js:46
    at app.js:46:13
```

Le let fonctionne comme le var sauf que :
il. Ne rend les variables disponible que à un bloc { }
Boucle, fonction, condition peu importe.

Reprenons les Bug de tout à l'heure mais en utilisant le mot clé **let**

```
console.log(moto);
let moto = 'Yamaha'
```

```
▶Uncaught ReferenceError: Cannot access 'moto' before          app.js:48
initialization
    at app.js:48:13
```

On obtient dès lors des erreurs certes mais beaucoup plus précises.

```
let voiture2 = 'Mitsubishi';
const modele = 'Sport';

let voiture2 = 'Citroen';
```

```
Uncaught SyntaxError: Identifier 'voiture2' has already been declared (at app.js:54:5)        app.js:54
```

Ici avec const : erreur voiture2 a déjà été déclarée

Exercice Quizz :

```
let superCar = 'BMW';
const superModel = 'Sport';

if(superCar =='BMW'){
    const superVitesse = 900;
    let superCar = "Citroen";
    console.log(superCar);

}
console.log(superCar);
```

Solution

Ici c'est Ok Il faut bien comprendre que JS va créer 2 variable Différentes
(le voiture du haut n'est pas le même qu'en bas)
Importance du scope

Const

```
const superConstante = 'YES';
superConstante = 12;
```

Une constante c'est une variable dont on sait qu'on ne va pas lui ré assigner une valeur, cela permet d'appliquer plus de restrictions dans notre code afin de ne pas créer plusieurs variable du même nom (des doublons) et qui servent à la même chose dans le programme.

Le code est mieux structuré donc plus lisible / compréhensible donc plus facile à maintenir.

```
Uncaught TypeError: Assignment to constant variable.
    at app.js:70:16
```

Une légère exception ?
On peut quand même « appliquer une modification » à une constante si c'est un objet, on peut modifier une **propriété** de cet objet (ci dessous on ne peut pas modifier la structure (l'objet en lui même), mais seulement une de ses propriétés.

```
const MyTracklist = {
    track1:'lofteurs up and down',
    track2:'David Hallyday',
    track3:'Crazy Frog'
}
console.log(MyTracklist);

MyTracklist.track1 = 'félicien'
```

```
app.js:92
▶{track1: 'lofteurs up and down', track2: 'David Hallyday', track3: 'Crazy Frog'}
app.js:95
▶{track1: 'félicien', track2: 'David Hallyday', track3: 'Crazy Frog'}
```

Au final en JS moderne (ES6 et +) on va privilégier l'utilisation de **let** et **const** pour créer des variables.

D.O.M

Présentation

Le DOM est donc une arborescence que va fabriquer le navigateur quand il interprète un fichier HTML, comme cité précédemment en JS tout est objet, et dans le DOM nous allons pouvoir retrouver tous les éléments HTML de notre page sous forme d'objets (ayants des propriétés) manipulables par JS.