Javascript

partie 3

Les boucles

1) Définition :

Les boucles permettent de répéter des actions simplement et rapidement.
Une boucle peut être vue comme une version informatique de « faire X fois quelque chose ».

Il y a différents types de boucles mais elles se ressemblent toutes au sens où elles répètent une action un certain nombre de fois (ce nombre peut éventuellement être zéro). Les différents types de boucles permettent d'utiliser différentes façon de commencer et de terminer une boucle. Chaque type de boucle pourra être utilisé en fonction de la situation et du problème que l'on cherche à résoudre.

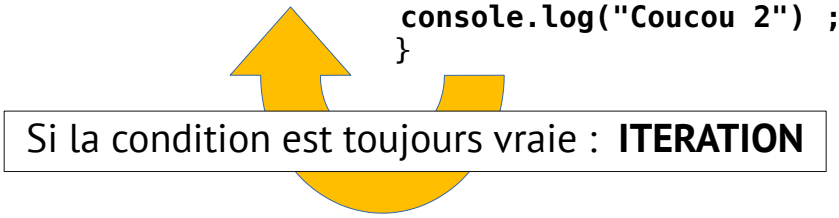
Nous allons travailler sur 5 types de boucles :

- while ;
- do while.
- for ;
- for in et for of ;

2) Quelques infos avant commencer :

Un peu de lexique :
Itération - En français « répétition ».
En informatique, une action qui se répète jusqu'à ce qu'une condition particulière soit donnée comme étant vraie. Tant que cette condition n'est pas vraie, l'itération continuera à être exécutée. Exemple :

```
répéter les instructions ci dessous (jusqu'à que cette
condition soit vraie) {
            console.log("Coucou 1") ;//une instruction
            console.log("Coucou 2") ;//une autre instruction
            }
```



Quelques opérateurs importants dans une boucle :

Incrémentation et décrémentation

L'opérateur d'incrément ajoute une unité à son opérande (une variable, par exemple) et renvoie une valeur.

- Si l'opérateur est utilisé en suffixe (par exemple : x++), il renvoie la valeur avant l'incrémentation.
- Si l'opérateur est utilisé en préfixe (par exemple : ++x), il renvoie la valeur après l'incrémentation.

```
// Suffixe
let x = 2;
y = x++; // y = 2, x = 3 car la variable y va recevoir la valeur de x avec l'incrémentation

// Préfixe
let x = 2;
y = ++x; // y = 3, x = 3 car la variable y va recevoir la valeur de x après l'incrémentation
```

La variable nommée i (ou compteur):

Par convention commune (jargon) **une variable nommée i** est utilisée pour compter le nombre d'itérations dans une boucle. Cette variable peut aussi être appelée « compteur »

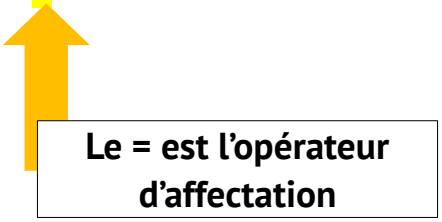
```
let i = 0;
while(i < 10) {
    console.log("log :" + i);
    i++;
}
```

tant que i (démarré à zéro) est plus petit que 10, la boucle itérera

Opérateur d'affectation

Pour attribuer une valeur à une variable précédemment déclarée, on utilise l'opérateur d'affectation =

```
a = 6;
b = "Foo";
```



3) La Boucle *while*

La boucle while (« tant que » en français) va nous permettre de répéter une série d'instructions tant qu'une condition donnée est vraie.

Le lexique :

```
while ( condition ) {
instruction;
}
```

*PS : le lexique de la boucle while est très similaire à if. La différence c'est que, dans **while**, les instructions à l'intérieur de l'accolade vont itérer (répéter) jusqu'à que la condition soit donnée comme étant vraie*

Exemple 1 :

```
let x = 0;
let y = 0;

while (x < 3) {
  x++; //instruction 1
  y += x;  //instruction 2
}
```

Dans notre exemple, à chaque itération :

1. la boucle incrémente x (ajoute +1 à x, dans ce cas)
2. et additionne la nouvelle valeur de x à y.

```
let x = 0;
let y = 0;

while (x < 3) {
  x++;
  y += x;
}
```

Ainsi, x et y prennent les valeurs suivantes :

- **itération 1 :**
  x = 1 //incrémentation de +1 à valeur originale de X (0 + 1)
  y = 0 //la valeur originale de la variable
  y = 1 // on additionne x à y (1 + 0)
  après l'itération 1, nous obtenons x = 1 et y = 1

- **itération 2** :
  x = 2 //incrémentation de +1 à valeur de x stockée après l'itération 1 (1 + 1)
  y = 1 //la valeur de y stockée après l'itération 1
  y = 3 // on additionne x à y (2 + 1)
  après l'itération 2, nous obtenons x = 2 et y = 3

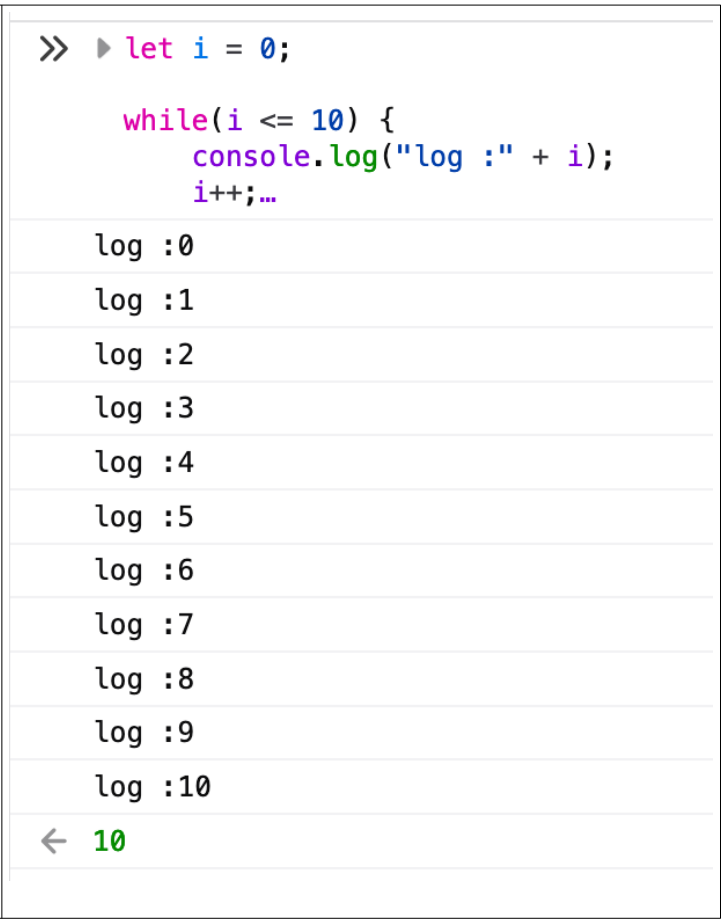
- **itération 3** :
  x = 3 //incrémentation de +1 à valeur de x stockée après l'itération 2 (2 + 1)
  y = 3 //la valeur de y stockée après l'itération 2
  y = 6 // on additionne x à y (3 + 3)
  après l'itération 3, nous obtenons x = 3 et y = 6
  la condition x < 3 n'est plus vraie et donc la boucle s'arrête.

Exemple 2 :

```
let i = 0;

while(i <= 10) {
    console.log("log :" + i);
    i++;
}
```

1) on initialise la variable i à 0
2) notre boucle itérera jusqu'à que la variable soit plus petite ou égale à 10
3) à chaque itération, on concatène le mot « log : » + la valeur de la variable
4) et on incrémente +1 à sa valeur
5) ensuite on recommence l'itération tant que la condition i<=10 soit vraie
5) une fois que i est plus grand que 10, la boucle s'arrête



Exemple 3 : boucle while affichera uniquement les nombres pairs de 2 à 20

```
let i = 2;

while(i <= 20) {
    console.log("log :" + i);
    i += 2;
}
```

1) on initialise la variable i à 2
2) notre boucle itérera jusqu'à que la variable soit plus petite ou égale à 20
3) à chaque itération, on concatène le mot « log : » + la valeur de la variable
4) et on incrémente +2 à sa valeur, pour afficher uniquement les valeurs paires
5) ensuite on recommence l'itération tant que la condition i<=20 soit vraie
5) une fois que i est plus grand que 20, la boucle s'arrête

```
» i = 2;

    while(i <= 20) {
        console.log("log :" + i);
        i += 2;…
log :2
log :4
log :6
log :8
log :10
log :12
log :14
log :16
log :18
log :20
← 22
```

Exemple 4 : une instruction "if else" à l'intérieur d'une boucle while

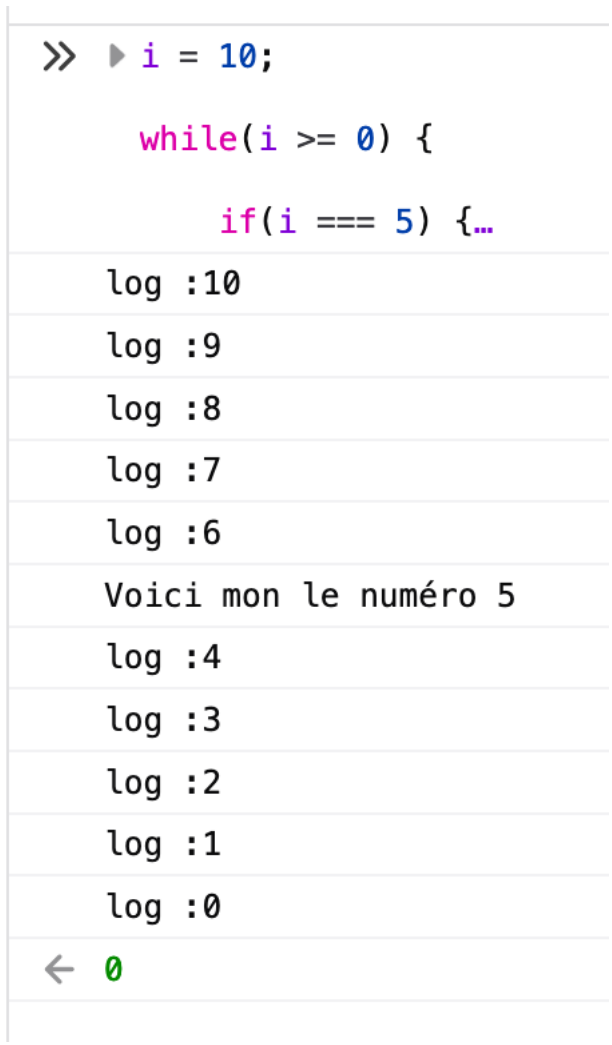
```
let i = 10;

while(i >= 0) {

    if(i === 5) {
        console.log("Voici mon
le numéro 5");
    }
    else{
        console.log("log :" +
i);
    }

    i--;
}
```

1) on initialise la variable i à 10
2) notre boucle itérera jusqu'à que la variable soit plus grande ou égale à 0
3) nous donnons les instructions suivantes : si la valeur de la variable est **strictement égale à 5**, nous signalons avec le string «Voici mon le numéro 5», sinon on concatène le mot « log : » + la valeur de la variable
4) et on décrémente -1 à sa valeur, pour afficher uniquement un compte à rebours
5) ensuite on recommence l'itération tant que la condition i>=0 soit vraie
6) une fois que i est plus petit que 0, la boucle s'arrête



Exemple 5 : boucle while infinie accidentelle

```
let i = 20;

while(i >= 0) {

    if(i === 10) {
        console.log("Dix");
    }
    else{
        console.log("log :" +
i);
         i--;
    }
}
```

1) on initialise la variable i à 20
2) notre boucle a l'intention d'itérer jusqu'à que la variable soit plus grande ou égale à 0
3) nous donnons les instructions suivantes : si la valeur de la variable est strictement égale à 10, nous signalons avec le string «Dix», sinon on concatène le mot « log : » + la valeur de la variable **et on décrémente -1 à sa valeur.**
ATTENTION : **la décrémentation se trouve à la porte de l'instruction else** - *contrairement à l'exemple intérieur, ou la décrémentation était lue **après** les conditions if et else.*
La décrémentation ne sera donc pas lue.
POURQUOI : l'instruction if (i===10) est vraie, à ce moment de l'itération. Vu que l'if est donné vrai, l'else n'est pas lu et **la boucle itère.** Et ensuite le cycle se répète :

1. i === 10
2. le string "Dix" s'affiche à la console
3. la boucle itère
4. i === 10
5. le string "Dix" s'affiche à la console
6. la boucle itère
7. **...à l'infini.**

```
» i = 20;
    while(i >= 0) {
        if(i === 10) {…
log :20
log :19
log :18
log :17
log :16
log :15
log :14
log :13
log :12
log :11
Dix                                30946
```

Boucle infinie

Pour éviter des boucles infinies, il faut bien vérifier les points suivant :
- la condition de sortie de boucle est réalisable et correctement écrite,
- la variable « compteur » est bien incrémentée ou décrémentée,
- l'incrémentation ou décrémentation est à la bonne portée, de façon qu'elle permet réaliser un cycle d'itération

3) La Boucle *do..while*

Dans une boucle do...while, la condition vient après les instructions qu'on souhaite exécuter.

Synthase

```
do{
    instruction}
while (condition);
```

instruction
: L'instruction est exécutée au moins une fois et ré-exécutée chaque fois que la condition est évaluée à true.

condition
: Si la condition donne true, instruction sera exécutée à nouveau. Noter que l'instruction à l'intérieur d'une boucle do... while sera toujours exécutée au moins une fois, même si la condition n'est pas vraie.

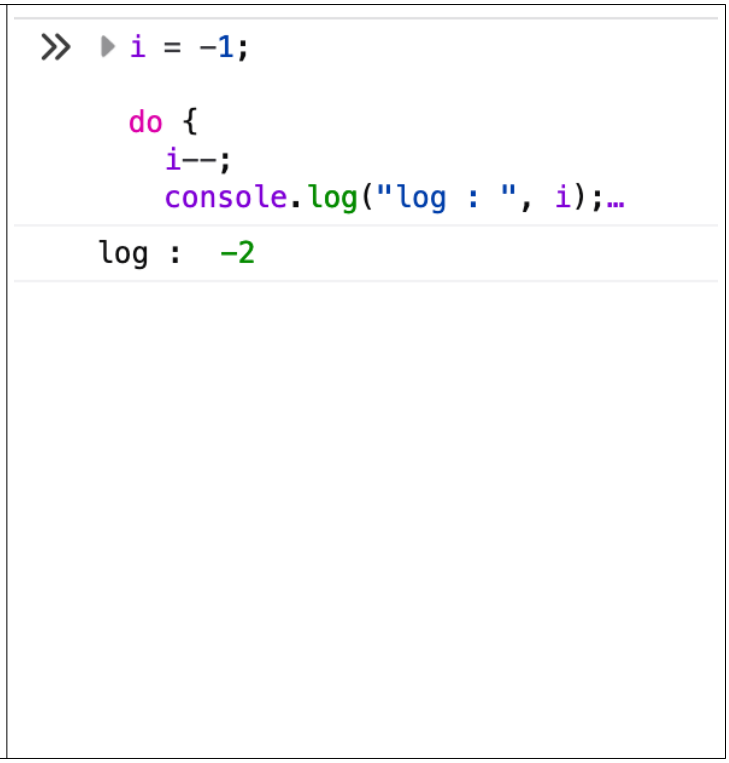
// Il est important de terminer la boucle avec un point-virgule.

Exemple

```
let i = -1;

do {
  i--;
  console.log("log : ", i);
} while( i >= 0);
```

1) on initialise la variable i à -1
2) on décrémente -1 à i. I = -2
3) on concatène le mot « log : » + la valeur de la variable i
4) la condition n'est pas remplie (i n'est pas égal ou supérieur à 0), la boucle s'arrête.
Cependant, toutes les instructions comprises dans do ont déjà été exécutées



4) La Boucle *for*

Synthase

```
for (initialisation; condition; expression_finale)
   { instruction }
```

`initialisation`
: Une expression d'affectation (exemple : *x* += *y* ou *x* >= *y*) ou une déclaration de variable (exemple : *let i = 0*). Cette expression est évaluée **une unique fois avant que la boucle démarre**. On utilise généralement **la variable nommée i** qui agit comme un compteur.

`condition`
: Une condition qui est évaluée avant chaque itération de la boucle. Si cette expression retourne « true », l'`instruction` est exécutée.

`expression_finale`
: Une expression qui est évaluée à la fin de chaque itération. **Cela se produit avant l'évaluation de la** `condition`. Cette expression est généralement utilisée pour mettre à jour, incrémenter ou décrémenter le compteur (`variable d'initialisation`).

`instruction`
: Une instruction qui est exécutée tant que la `condition` de la boucle est « true ».

Comparaison de la synthase entre *while* et *for* :

while	for

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

Exemple 1 :

```
let n = 0;
for (let i = 2; i < 5; i++) {
    n += i;
  console.log("i = " + i);
  console.log("n = " + n);
}
```

```
» n = 0;
  for (var i = 2; i < 5; i++) {
     n += i;
    console.log("i = " + i);
    console.log("n = " + n);…
  i = 2
  n = 2
  i = 3
  n = 5
  i = 4
  n = 9
← undefined
```

Ainsi, n et i prennent les valeurs suivantes :

- **Avant l'itération 1 :**
  n = 0 //on déclare la valeur de n
  i = 2 //initialisation du compteur à 2
  on présente la condition (i doit être inférieur à 5)
  si la condition est true, on incrémente 1 au compteur (variable i) **dès l'itération 2**

- **itération 1** :
  i = 2 //aucune incrémentation n'a encore été faite car il s'agit de la première itération
  n = 0 // valeur déclarée de n
  n = i + n (n = 2 + 0)
  n = 2 //aucune incrémentation n'a encore été faite car il s'agit de la première itération
  après l'itération 1, nous obtenons n = 2 et i = 2

- **itération 2** :
  i = 3 // incrémentation de +1 à valeur de n stockée après l'itération 1 (2 + 1)
  n = 2 // valeur de n stockée après l'itération 1
  n = i + n (n = 3 + 2)
  n = 5
  après l'itération 3, nous obtenons n = 5 et i = 3

- **itération 3** :
  i = 4 // incrémentation de +1 à valeur de n stockée après l'itération 2 (3 + 1)
  n = 5 // valeur de n stockée après l'itération 2
  n = i + n (n = 4 + 5)
  n = 9
  après l'itération 3, nous obtenons n = 9 et i = 4

  la condition i < 5 ne peut plus être vraie car si la boucle itère une 4ème fois, i sera = à 5. La boucle s'arrête

Exemple 2 :

```
for(let i = 10; i >= 0; i--) {
    console.log("for : " + i);
}
```

1) on initialise la variable i à 10
2) notre boucle itérera jusqu'à que la variable soit plus grande ou égale à 0
3) et on décrémente -1 à sa valeur, pour afficher uniquement un compte à rebours
5) ensuite on recommence l'itération tant que la condition i>=0 soit vraie
6) une fois que i est plus petit que 0, la boucle s'arrête

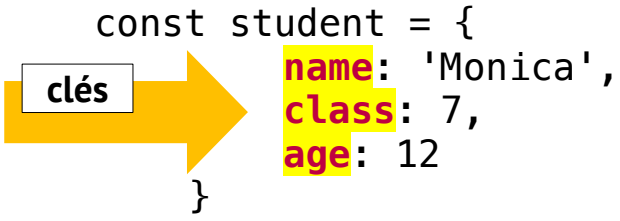
```
>> for(i = 10; i >= 0; i--) {
       console.log("for : " + i);
   }
   for : 10
   for : 9
   for : 8
   for : 7
   for : 6
   for : 5
   for : 4
   for : 3
   for : 2
   for : 1
   for : 0
<- undefined
```

5) La Boucle *for...in* et *for...of*

5.1) *for..in*

L'instruction **for...in** permet d'itérer sur les **clés** trouvées dans les propriétés d'un objet

Rappel :



Exemple

```
const persone = {
    prenom: 'Vanessa',
    nom: 'Sant'André',
    petit_nom: 'Prof'
};

for (const vanessa in persone) {
    console.log(vanessa +" : "+ persone[vanessa]);
}
```

1) on initialise la variable du type objet « persone », on assigne les propriétés (clé + valeur)
2) on crée une variable « vanessa » qui stockera les clés de « persone »
3) la boucle balaie l'objet « persone » et stocke les clés dans « vanessa »
5) l'instruction à l'intérieur de la boucle affiche dans la console la concaténation de toutes les clés + les valeurs des clés persone[vanessa]

```
» const persone = {
        prenom: 'Vanessa',
        nom: 'Sant'André',
        petit_nom: 'Prof'
    };…
prenom : Vanessa
nom : Sant'André
petit_nom : Prof
← undefined
```

5.2) *for..of*

L'instruction for...of permet d'itérer sur les **valeurs** trouvées d'un **objet itérable** : arrays, strings, arguments dans une function, etc).

Définition d'un objet itérable

Comme nous avons vu précédemment, « itérer » en français signifie « répéter ». En JavaScript, un objet est dit « itérable » si celui-ci a été créé de façon à ce qu'on puisse parcourir ses valeurs une à une.

Exemple d'un array (objet itérable), où nous pouvons parcourir ses valeurs à partir de ses index :

```
let monTableau = ['Vanessa', 'prof', 'admin']
```

Vanessa	prof	admin
0	1	2

Indice (ou index en anglais)

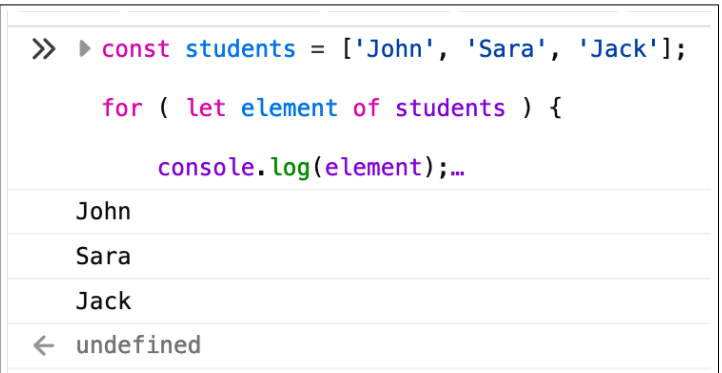
Exemple - la boucle for ... of est utilisée pour parcourir la variable array « students » et afficher toutes ses valeurs.

```
const students = ['John', 'Sara', 'Jack'];


for ( let element of students ) {

    console.log(element);
}
```

1) on initialise la variable du type array « students », on assigne les valeurs. On crée une variable « element» qui stockera les valeurs de « students »
3) la boucle balaie l'array « students » et stocke les valeurs dans « element »
5) l'instruction à l'intérieur de la boucle affiche dans la console les valeurs du tableau students



for of versus for if

```
let toto = [3, 5, 7];
```

for...of

```
for (const i of toto) {
  console.log(i);
}
//affiche les valeurs de l'array
```

```
toto = [3, 5, 7];

for (let i of toto) {
  console.log(i); …
3
5
7
```

for...in

```
for (let i in toto) {
  console.log(i);
}
//affiche les index de l'array
```

```
toto = [3, 5, 7];

for (let i in toto) {
  console.log(i); …
0
1
2
```

La boucle for ... of est utilisée pour parcourir les valeurs d'un objet itérable (tels arrays et strings, etc).

La boucle for ... of ne peut pas être utilisée pour itérer sur un objet général (donc non itérable). Exemple :

```
const profs = {
    prenom: 'Vanessa',
    nom: 'Sant'André',
};

for (const vanessa of profs) {
    console.log(vanessa);
}
```

```
» const profs = {
      prenom: 'Vanessa',
      nom: 'Sant'André',
  };
  …
Uncaught TypeError: profs is not iterable
    <anonymous> debugger eval code:6
  [En savoir plus]
```

La boucle for ... in est utilisée pour parcourir les clés d'un objet.

On peut utiliser for ... in pour itérer sur un objet itérable de tels arrays et strings,

for...in doit être évité si nous voulons parcourir un Array lorsque l'ordre des valeurs est important.

Les indices d'un tableau sont des « clés » composées par des **numéros entiers** :
Exemple :

- dans toto = [3, 5, 7], la « clé » de la première valeur est égale à toto**[0]**;
- dans toto = {age : 3}, la clé de la première valeur est **age**;

(A part ça, les indices sont en tout point identiques aux clés des objets en général.)

Il n'y a donc aucune certitude que for...in renvoie les indices dans un ordre particulier. L'ordre dans lequel le parcours est effectué dépend de l'implémentation (navigateur, version...).

for..in n'est pas garanti de conserver l'ordre des éléments

L'autre motif : c'est lent car vous devez parcourir toutes les propriétés de l'objet tableau et n'obtiendrez toujours que la clé. C'est-à-dire que pour obtenir la valeur, une recherche supplémentaire sera nécessaire.

Exemple :

```
for (const vanessa in persone) {
    console.log(vanessa +" : "+ persone[vanessa]);
}
```