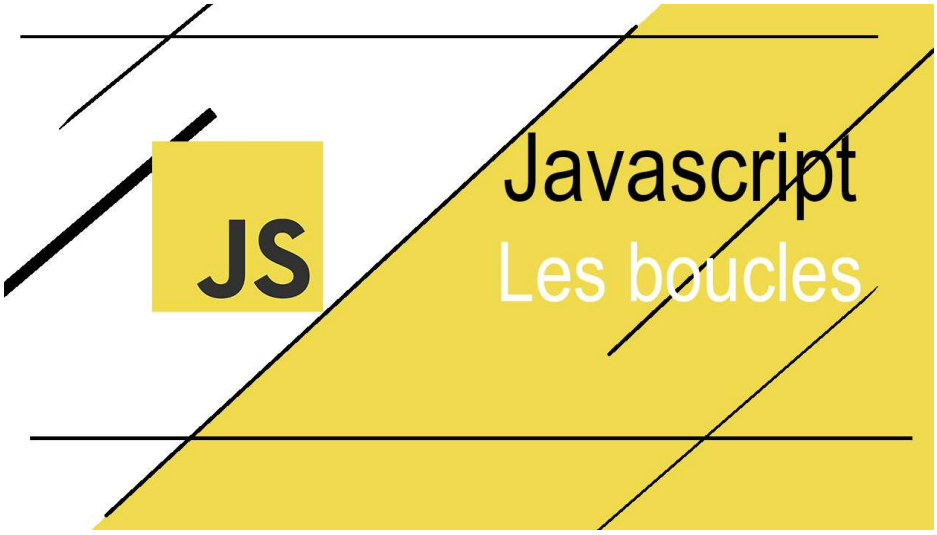

Comment fonctionnent la boucle for en javascript

10 min read

Redkire (https://leblogducodeur.fr/author/redkire/) - 14 février 2020 -
javascript (https://leblogducodeur.fr/category/javascript/) -
0 commentaire (https://leblogducodeur.fr/boucle-for-javascript/#respond)

Rechercher

CATÉGORIES

Sélectionner une catégorie

SUIVEZ-NOUS

(https://www.youtube.com/channel/UCBtChUM0xHu0d_FubGu_iIQ?view_as=subscriber) (https://twitter.com/CollignonGailen) (https://www.instagram.com/leblog_ducodeur/?hl=fr) (https://www.facebook.com/Le-Blog-Du-Codeur-104644620907623/)

Dans cet article, nous allons voir comment fonctionne la boucle for en Javascript et comment l'utiliser.

C'est quoi une boucle for ?

Une boucle for est un bloc de programmation fondamental permettant d'exécuter une action un certain nombre de fois et selon certaines conditions.

Les boucles for existent dans tout les langages de programmation. Elles sont utilisées dans tout les programmes.

Il est essentiel de les maîtriser. Elles permettent de créer des variables complexes, de gérer des données, d'effectuer des actions spécifiques etc..

Comment utiliser une boucle for

La boucle for Javascript basique dérive du langage C. On utilise donc la même syntaxe. Il y a 3 éléments distincts dans une boucle for. L'initialisation, la condition et le modificateur :

```
1 for (let i =0; i < 100; i++){
2
3 }
```

Ici, on déclare notre variable d'initialisation à 0. On utilise let et pas const car cette variable sera modifiée.

Ensuite, on sépare la seconde partie avec un ";". C'est notre condition. Ici on précise que notre boucle tournera tant que notre variable i initialisée à 0 sera inférieure à 100. Enfin, on as notre modificateur. Ici on explique qu'à chaque passage de la boucle on augmente la valeure de i de 1.

On as donc un système plutôt simple. On initialise une variable à 0. Tant qu'elle est inférieure à 100, notre boucle tourne. Et à chaque passage, on augmente la valeure de 1. On aura donc 100 passages de boucle.

Je vais ajouter une instruction simple à l'intérieur :

```
1 for (let i =0; i < 100; i++){
2     console.log('ma boucle passe 100 fois')
3 }
```

Ici, on passe les instructions dans les crochets, comme les fonctions ou les blocs conditionnels.

Si je lance mon code, j'aurais un console.log() 100 fois, notre boucle fonctionne.

Utiliser la variable i

Bien sûr, il est possible d'aller un peu plus loin. Faire une boucle pour répéter 100 fois quelque chose ça ne sers pas à grand chose.

Ce qui va vraiment nous servir dans cette boucle, c'est la variable i.

Si j'exécute ce code, vous allez voir quelque chose de magique :

```
for (let i =0; i < 100; i++){
    console.log('passage numéro ' + i)
}
```

Ici, notre code nous indique à quel passage il est. Notre variable i est augmentée à chaque passage de boucle.

Avec ça, on peux faire énormément de choses. Je vais vous donner un exemple que j'utilise dans mes cours en ligne.

Imaginez une liste de prénoms. Vous voulez afficher chaque prénoms individuellement mais de manière automatique, c'est à dire sans accéder aux index. Comment faire ?

Et bien les boucles nous permettent de le faire assez simplement

```
let prenoms = ['Michel', 'Jean', 'Pierre', 'Paul']
for (let i =0; i < 4; i++){
    console.log(prenoms[i])
}
```

Si je lance ce code, je vais obtenir ce résultat :

```
Michel
Jean
Pierre
Paul
```

Ici, il va m'afficher individuellement chaque élément de ma liste.

Mais pourquoi ?

Et bien parce que i change de valeur. On passe i dans notre liste comme si on passait un index. Au début il est à 0 et il augmente à chaque fois. On récupère donc tout les éléments de notre liste.

Il y a un soucis à ça... Comment faire si vous ne connaissez pas la longueur de votre liste ? Et bien on peux utiliser la méthode length :

```
let prenoms = ['Michel', 'Jean', 'Pierre', 'Paul']
for (let i =0; i < prenoms.length; i++){
    console.log(prenoms[i])
}
```

Ici, on lui dis de boucler tant que notre variable i est inférieure à la longueur total de notre liste. Dis comme ça, on pourraît se dire que l'on va manquer le dernier élément mais non.

En programmation, on accède aux éléments à partir de 0 mais la valeur de la méthode length commence à partir de 1. Le système est parfait.

On as donc une boucle dynamique. Peu importe la longueur de la liste on accède à chaque éléments de manière individuelle.

Sauf qu'on peux faire encore mieux...

La boucle for in

On as deux soucis avec notre boucle de base. Elle est longue à écrire et elle ne fonctionne pas sur les objets. Une nouvelle façon de faire des boucles for sortie il y a quelques années, la for in nous permets de palier à ces deux problèmes.

Le concept est extrêmement simple :

```
let prenoms = ['Michel', 'Jean', 'Pierre', 'Paul']
for (i in prenoms){
    console.log(prenoms[i])
}
```

Ici on lui dis d'aller nous récupérer la position de l'élément de manière dynamique. Plus besoin d'utiliser length.

i va prendre successivement la valeur de chaque index. Il nous suffit ensuite de passer cet variable comme index de notre liste pour récupérer la valeur.

Le truc génial, c'est que ça fonctionne aussi avec les objets :

```
let magicien = {
    magie : 300,
    force : 20,
    prenom : 'Gandalf',
    taille : 200
};

for (key in magicien){
    console.log(key, ': ', magicien[key])
}
```

Ce qui donne :

```
magie :  300
force :  20
prenom :  Gandalf
taille :  200
```

Ici, on récupère la clé de chaque élément au lieu de son index. On passe ensuite la clé dans notre objet et on récupère les valeurs. Rien de plus simple.

Avec ça, on peux très facilement récupérer des éléments d'objets et de listes en quelques lignes.

Sauf que l'on doit passer l'index dans notre liste pour que ça fonctionne. On peux donc aller encore plus loin

La boucle for of

La boucle for of nous permets de récupérer directement les éléments sans passer par l'objet de base, elle ne fonctionne que pour les listes par contre. Puisque les objets ne sont pas ittérable de base, vous allez causer un bug.

Voici la marche à suivre :

```
let prenoms = ['Michel', 'Jean', 'Pierre', 'Paul']
for (let prenom of prenoms) {
    console.log(prenom)
}
```

Ici on récupère directement l'élement et on l'assigne à une variable, prenom. Voici le résultat :

```
Michel
Jean
Pierre
Paul
```

Comme vous le voyez, on as pu récupérer tout nos éléments sans passer par l'index. Tout est direct.

Un exemple d'utilisation de boucles

Faisons un petit exercice, essayons de créer un programme permettant de compter les nombres pairs et impairs dans une liste :

Avec les boucles, c'est très simple.

D'abord, on veux passer à travers tout nos élements. La manière la plus simple de le faire, c'est la for of

```
let nombres = [1,12,47,25,8932,85719,1818,17,139];
for (let nombre of nombres){

}
```

Ensuite, on ajoute une condition pour vérifier si notre nombre est pair. Pour ça, on divise le nombre par deux avec le modulo et on vérifie que le reste = 0. C'est assez logique, tout les nombres pairs sont divisible par 0 en conservant des entiers.

```
let nombres = [1,12,47,25,8932,85719,1818,17,139];
for (let nombre of nombres){
    if (nombre % 2 === 0){

    } else {

    }
}
```

On ajoute ensuite deux variables qui seront nos compteurs de nombres pairs et impairs :

```
let nombres = [1,12,47,25,8932,85719,1818,17,139];
let pairs = 0;
let impairs = 0;
for (let nombre of nombres){
    if (nombre % 2 === 0){

    } else {

    }
}
```

Pour finir, on ajoute 1 au compteur correspondant et on console.log les résultats :

```
let nombres = [1,12,47,25,8932,85719,1818,17,139];
let pairs = 0;
let impairs = 0;
for (let nombre of nombres){
    if (nombre % 2 === 0){
        pairs += 1
    } else {
        impairs += 1
    }
}

console.log('pairs', pairs);
console.log('impairs', impairs)
```

Ce qui donne :

```
pairs 3
impairs 6
```

Si vous comptez manuellement, vous allez arriver au même résultat.

Comme vous le voyez, il est très simple de créer des morceaux de programmes utilisant les boucles for pour faire des calculs.

L'utilisation des mots clés break et continue

Pour finir, nous allons voir comment utiliser break et continue pour améliorer les fonctionnalités de nos boucles.

L'utilisation de break

Break nous permets de casser une boucle. Le fonctionnement est plutôt simple, dès que le mot clé est activé, notre boucle est stopée nette.

Pour vous illustrer ce fonctionnement, je vais créer un petit programme, on possède une liste de nombres et dès que le nombre 10 est trouvé, le programme nous indique sa position et casse la boucle.

Bien sûr, il est préférable de caler ce code dans une fonction afin de l'automatiser et de retourner une valeur.

Il est probable que certains de mes lecteurs ne comprennent pas le concept des fonctions, je ne vais donc pas aller aussi loin :

```
let nombres = [1,58,63,95,74,10,18,78,36];
let position = -1;
for (let i in nombres){
    console.log(nombres[i])
    if (nombres[i] === 10){
        position = i;
        break
    }
}

console.log('position',position);
```

Ici, on utilise for in parce que l'on as besoin de l'index de notre nombre pour le stocker. On applique une condition basique, si elle es vrai, on assigne la position et on break.

Dans ce cas, il y a bel et bien un 10. Il nous renvoi donc sa position (on compte à partir de 0) et il break. Puisque j'ai console.log() les éléments, vous allez voir que notre boucle s'arrête.

```
1
58
63
95
74
10
position 5
```

On as maintenant sa position, je peux donc accéder à l'élément de manière individuelle. Dans le cas ou il n'y à pas de 10, notre variable position est égale à -1

L'utilisation de continue

Continue nous permets de passer un élément dans notre boucle. Nous allons faire quelque chose d'un peu différent, essayons de créer un code qui nous permets de filtrer les

nombres pairs :

On va simplement console.log() les nombres pairs de notre liste :

```
let nombres = [1,58,63,95,74,10,18,78,36];

for (let nombre of nombres){
    if (nombre % 2 !== 0){
        continue
    }
    console.log(nombre)
}
```

Ici, continue va simplement aller au prochain élément de notre liste si il es tapé. Le console.log() ne sera donc exécuté que sur les nombres pairs :

```
58
74
10
18
78
36
```

En conclusion

Les boucles for nous permettent de passer à travers des données et d'exécuter des actions un certain nombre de fois. Il y a la for classique à condition, la for in à index et la for of à valeur.

Chaque boucle as ses avantages et vous devez apprendre à les choisirs. Il existe aussi des mots clés comme break et continue qui permettent d'ajouter de la logique dans vos boucles.

Je pense que cet article vous as enseigné tout ce qu'il y a à savoir sur les boucles en Javascript. Vous devriez être préparé pour créer vos boucles dans vos programmes.

Si cet article vous as plu, n'hésitez pas à télécharger mon livre PDF qui vous apprendra à bien débuter la programmation.

VOUS ABONNER

Articles similaires

Les boucles en Python (https://leblogducodeur.fr/les-boucles-en-python/)
2 mai 2019
Dans "python"

Les arrays en PHP (https://leblogducodeur.fr/les-arrays-en-php/)
11 mars 2019
Dans "PHP"

Comment créer une animation en Javascript (https://leblogducodeur.fr/comment-creer-une-animation-en-javascript/)
30 décembre 2019
Dans "animation"

VOUS DEVRIEZ ÉGALEMENT AIMER

(https://leblogducodeur.fr/comment-creer-une-animation-en-javascript/)

Comment créer une animation en Javascript
7 min read
(https://leblogducodeur.fr/comment-creer-une-animation-en-javascript/)

30 décembre 2019

(https://leblogducodeur.fr/fetch-javascript/)

Comment utiliser fetch en Javascript
7 min read
(https://leblogducodeur.fr/fetch-javascript/)

3 janvier 2020

(https://leblogducodeur.fr/top-5-des-frameworks-web-de-programmation/)

TOP 5 des frameworks web de programmation
6 min read
(https://leblogducodeur.fr/top-5-des-frameworks-web-de-programmation/)

9 mai 2019

Laisser un commentaire

Votre commentaire ici...

Nom (nécessaire)

E-mail (nécessaire)

Site web

☐ Prévenez-moi de tous les nouveaux commentaires par e-mail.

☐ Prévenez-moi de tous les nouveaux articles par e-mail.

PUBLIER LE COMMENTAIRE