5 minutes

Les notions de valeur et de référence

Vous entendrez souvent qu'en `JavaScript`, **les primitives sont passées par valeur et les objets par référence**.

Vous entendrez également que les primitives, contrairement aux objets, sont **immuables**.

Les primitives

Les primitives sont pour rappel : les booléens, `null`, `undefined`, les nombres, les `BigInt`, les chaînes de caractères et les symboles.

Les primitives sont copiées par valeur

Lorsque l'on assigne une valeur primitive à une variable, la valeur est copiée sur un nouvel emplacement mémoire.

Nous avons donc :

```
const a = 1;
let b = a;
console.log(a, b); // 1 1

b = 2;
console.log(a, b); // 1 2
```

Copier

La valeur contenue dans `a` n'a pas été modifiée car lorsque nous faisons `b = a`, nous copions la valeur contenu dans `a` dans un nouvel emplacement mémoire pour `b`.

Lorsque nous modifions `b`, `a` n'est donc pas modifiée.

Cela entraîne le fait que les primitives sont immuables, elles ne peuvent pas être modifiées.

Exemple :

```
let a = 'bon';
let b = 'jour';
let c = a + b;
console.log(c); // "bonjour"

a = 'test';
console.log(c); // "bonjour"
```

Copier

Lorsque nous concaténons deux chaînes de caractères, les deux valeurs sont concaténées et le résultat est une nouvelle chaîne de caractères dans un nouvel emplacement en mémoire.

Les primitives sont comparées par valeur

Lorsque nous comparons deux primitives, nous les comparons par valeur, nous verrons que ce n'est pas le cas pour les objets.

```
const a = 1;
let b = 1;
console.log(a === b); // true
```

Copier

Les objets

Pour le moment, c'était évident et nous n'avons eu aucune surprise. Mais pour les objets c'est différent. Ne vous inquiétez pas nous verrons tous les objets en détails dans les prochains chapitres.

Ce qu'il est important de voir dès maintenant c'est les mécaniques d'assignation et de comparaison des objets.

Les objets sont copiés par référence

Une référence à un objet est son adresse dans la mémoire.

Comme les objets peuvent être très volumineux il est beaucoup plus efficient de ne pas les copier à chaque fois mais de passer leur référence, c'est-à-dire leur adresse, lors d'assignations :

```
const obj = {a: 1};
const obj2 = obj;
obj2.a = 2;
console.log(obj); // {a: 2}
```

Copier

Pas de panique ! Une fois que vous comprenez le mécanisme c'est en fait très logique.

Nous avons un objet qui contient une propriété qui est modifiable (les objets contrairement aux primitives peuvent être modifiés).

Dans la variable `obj` nous n'avons pas directement l'objet mais la référence de l'objet, c'est-à-dire son adresse dans la mémoire.

Lorsque nous copions la valeur de `obj` dans une nouvelle variable `obj2`, **nous copions en fait la référence vers le premier objet**.

Les deux variables contiennent donc la même référence vers le même objet.

Donc lorsque nous modifions la propriété `a` de `obj2` nous modifions en fait le même objet auquel `obj` fait référence, d'où le résultat !

Les objets sont comparés par référence

Le objets sont également comparés par référence :

```
const obj = {a: 1};
const obj2 = obj;
console.log(obj === obj2); // true
const obj3 = {a: 1};
console.log(obj === obj3); // false
// Même chose ici :
console.log( {} === {}); // false
```

Copier

La première comparaison est évidente. Les deux variables `obj` et `obj2` contiennent la même référence vers le même objet et l'égalité stricte est donc vraie.

La deuxième comparaison est moins évidente, mais c'est le même mécanisme.

Lorsque nous faisons `{a: 1}`, nous créons **un nouvel objet et qui a donc une nouvelle référence, c'est-à-dire une nouvelle adresse en mémoire**.

Donc lorsque nous comparons les valeurs de `obj` et de `obj3`, les deux références sont différentes. Peu importe que les deux objets aient les mêmes propriétés car ce sont les références qui sont comparées.

C'est une des notions les plus importantes en `JavaScript`, vous pourrez y revenir lorsque nous aurons vu en détails les objets.