Apprendre JavaScript - Passage par référence vs passage par valeur

Introduction

En JavaScript, les variables peuvent être passées par référence ou par valeur. Il est important de comprendre la différence entre ces deux types de passage de variables pour éviter les erreurs dans votre code.

Passage par valeur

Les valeurs simples (nombre, chaîne de caractères, booléen, `null`, `undefined`, `symbol`) sont passées par valeur. Cela signifie que lorsqu'une variable est passée à une fonction, une **copie de la valeur** de la variable est passée à la fonction. Si la fonction modifie la valeur de la variable, cela n'affectera pas la valeur de la variable à l'extérieur de la fonction.

```
let a = 5;

function modifierValeur(valeur) {
    // Valeur est une copie de la variable a
    valeur = 10;
}
```

Dans cet exemple, la variable `a` ne sera pas modifiée par la fonction `modifierValeur`.

Passage par référence

Les objets (tableaux, objets, fonctions) sont passés par référence. Cela signifie que lorsqu'une variable est passée à une fonction, c'est une **référence à l'objet** qui est passée à la fonction. Si la fonction modifie l'objet, cela affectera l'objet initial à l'extérieur de la fonction.

```
let tableau = [1, 2, 3];

function modifierTableau(t) {
    // t est une référence au tableau
    t.push(4);
}

modifierTableau(tableau);

console.log(tableau); // [1, 2, 3, 4]
```

Dans cet exemple, le tableau initial est modifié par la fonction `modifierTableau`.

Meilleures pratiques

Il est recommandé de **ne pas modifier les paramètres** passés à une fonction. Ainsi, la fonction retournera toujours la même valeur pour les mêmes paramètres. On appelle cela une fonction **pure**.

Si vous devez modifier un objet, il est recommandé de créer une copie de l'objet avant de le modifier. Cela permet de garder le code plus lisible et de prévenir les effets secondaires.

Ainsi, vous évitez de modifier l'objet initial et vous pouvez toujours accéder à l'objet initial si nécessaire.

Copie de tableaux

```
let tableau = [1, 2, 3];

function modifierTableau(t) {
    // Créer une copie du tableau
    let copieTableau = t.slice();
    copieTableau.push(4);
    return copieTableau;
}

let nouveauTableau = modifierTableau(tableau);

console.log(tableau); // [1, 2, 3]
console.log(nouveauTableau); // [1, 2, 3, 4]
```

On peut aussi utiliser l'opération de décomposition (`spread`) pour copier un tableau. Les ... permettent de copier les éléments d'un tableau dans un autre tableau. Voyez cela comme "le reste des éléments" du tableau initial dans un nouveau tableau.

```
let tableau = [1, 2, 3];

function modifierTableau(t) {
    // Créer une copie du tableau
    let copieTableau = [...t];
    copieTableau.push(4);
    return copieTableau;
}

let nouveauTableau = modifierTableau(tableau);

console.log(tableau); // [1, 2, 3]
console.log(nouveauTableau); // [1, 2, 3, 4]
```

Copie d'objets

Pour copier un objet, vous pouvez utiliser la méthode `Object.assign` ou l'opérateur de décomposition (`spread`).

```
let objet = { a: 1, b: 2 };

function modifierObjet(obj) {
    // Créer une copie de l'objet
    let copieObjet = Object.assign({}, obj);
    copieObjet.c = 3;
    return copieObjet;
}

let nouvelObjet = modifierObjet(objet);

console.log(objet); // { a: 1, b: 2 }
console.log(nouvelObjet); // { a: 1, b: 2, c: 3 }
```

Ou avec l'opérateur de décomposition (`spread`).

```
let objet = { a: 1, b: 2 };

function modifierObjet(obj) {
    // Créer une copie de l'objet
    let copieObjet = { ...obj };
    copieObjet.c = 3;
    return copieObjet;
}

let nouvelObjet = modifierObjet(objet);

console.log(objet); // { a: 1, b: 2 }
console.log(nouvelObjet); // { a: 1, b: 2, c: 3 }
```