03_Concept de Synchrone et Asynchrone

Introduction

En JavaScript, la gestion de l'exécution du code peut être synchrone ou asynchrone. Comprendre la différence entre ces deux modes d'exécution est important pour écrire du code efficace et performant, surtout dans des applications web où les opérations peuvent prendre du temps (comme les appels réseau ou les accès aux bases de données).

Table des Matières

Exécution Synchrone

Qu'est-ce que l'Exécution Synchrone ?

L'exécution synchrone signifie que le code est exécuté ligne par ligne, dans l'ordre où il apparaît. Chaque opération doit se terminer avant que la suivante ne commence.

Exemple d'Exécution Synchrone

```
console.log("Début");

for (let i = 0; i < 3; i++) {
    console.log(i);
}

console.log("Fin");
```

Dans cet exemple, les messages "Début", "0", "1", "2", et "Fin" sont affichés dans cet ordre précis.

Problème de l'Exécution Synchrone

L'exécution synchrone peut causer des problèmes de performance lorsque des opérations longues sont impliquées, comme les appels réseau ou les opérations de fichier. Pendant qu'une opération synchrone est en cours, le reste du programme est bloqué et doit attendre que l'opération se termine.

Exécution Asynchrone

Qu'est-ce que l'Exécution Asynchrone ?

L'exécution asynchrone permet au programme de continuer à fonctionner pendant que d'autres opérations sont effectuées en arrière-plan. Cela est particulièrement utile pour les opérations qui prennent du temps.

Exemple d'Exécution Asynchrone

```
console.log("Début");

setTimeout(() => {
    console.log("Opération asynchrone terminée");
}, 1000);
```

```
console.log("Fin");
```

Dans cet exemple, les messages "Début" et "Fin" sont affichés immédiatement, tandis que "Opération asynchrone terminée" s'affiche après une seconde.

Callbacks

Qu'est-ce qu'un Callback ?

Un callback est une fonction passée en argument à une autre fonction qui est exécutée après que l'opération principale soit terminée.

Exemple avec Callback

```
function operationAsynchrone(callback) {
    setTimeout(() => {
        console.log("Opération terminée");
        callback();
    }, 1000);
}

console.log("Début");
operationAsynchrone(() => {
    console.log("Callback exécuté");
});
console.log("Fin");
```

Promises

Qu'est-ce qu'une Promise ?

Une Promise est un objet représentant l'achèvement (ou l'échec) d'une opération asynchrone et sa valeur résultante.

Exemple avec Promise

```
let promesse = new Promise((resolve, reject) => {
    setTimeout(() => {
        resolve("Opération réussie");
    }, 1000);
});

console.log("Début");

promesse.then(message => {
    console.log(message);
}).catch(error => {
    console.error(error);
});

console.log("Fin");
```

Async/Await

Qu'est-ce que Async/Await ?

Async/Await est une syntaxe qui simplifie l'écriture et la lecture de code asynchrone. `async` marque une fonction comme asynchrone, et `await` est utilisé pour attendre qu'une Promise soit résolue.

Exemple avec Async/Await

```
function attendreUneSeconde() {
    return new Promise(resolve => {
        setTimeout(() => {
            resolve("Opération réussie");
        }, 1000);
    });
```

```
}

async function executionAsynchrone() {
    console.log("Début");
    let resultat = await attendreUneSeconde();
    console.log(resultat);
    console.log("Fin");
}

executionAsynchrone();
```

Ressources Utiles

- MDN Web Docs: Asynchronous JavaScript
- JavaScript Info: Callbacks
- JavaScript Info: Promises
- JavaScript Info: Async/Await