Francis Munabeno

Async / Await

Les functions **asynchrones** permettent de créer et d'utiliser des **promesses** avec Un code plus **intuitif**, ce code ressemble davantage à la syntaxe classique de **JS**.

Ils sont apparus avec la version **2017** de JS et sont très utilisés par les **API**.

Async / Await

Une fonction définie avec le mot clé **async** renvoie systématiquement une **promesse**. Si une fonction retourne explicitement une valeur qui n'est pas une **promesse**, alors cette valeur sera **automatiquement** enveloppée dans une **promesse**.

Du coup, une fonction async retournera **toujours** une **promesse** qui sera résolue avec la valeur renvoyée par la fonction **asynchrone** ou qui sera **rompue** s'il y a une exception non interceptée émise depuis la fonction **asynchrone**.

Async / Await

Await ne peut être utilisé que dans une fonction **async**, Il permet d'interrompre l'exécution d'une fonction asynchrone tant qu'une promesse n'est pas résolue ou rejetée. Si **await** est utilisé en dehors d'une function **asynchrone**, cela provoquera une exception SyntaxError.

Await se met en **pause** tant qu'une promesse n'est pas **consommée**, puis return le résultat De la promesse. Cela ne **consomme aucune ressource supplémentair**e puisque le moteur Peut effectuer d'autres tâches en attendant (exécute d'autres scripts, gérer des événements, etc)

Conclusion, **await** est une alternative à **then()**, plus facile à lire, à comprendre et à écrire.

Async / Await

Async / Await

Grâce au **Async / Await** on gagne
en **lisibilité**, on a un code plus **facile**
à comprendre et qui n'est pas **verbeux**.

```
function getNb() {
    return number1()
    .then(nombre1 => {
    return number2()
    .then(nombre2 => nombre1 + nombre2);
})};
```

Promesse function

```
async function getNb() {
    const nombre1 = await number1();
    const nombre2 = await number2();
    return nombre1 + nombre2;
}
```

Async function

Francis Munabeno

Async / Await

```
function getNb() {
    return number1()
    .then(nombre1 => {
    return number2()
    .then(nombre2 => nombre1 + nombre2);
})};
```

Promesse function

Comme on peut le voir, cette équivalents
Est beaucoup moins clair que la version
async/await.

```
async function getNb() {
    const nombre1 = await number1();
    const nombre2 = await number2();
    return nombre1 + nombre2;
}
```

Async function

Comme on peut le voir, cette équivalent
Est beaucoup plus lisible que la version
avec la promesse.

Async / Await

Si une **promesse** est résolue (opération effectuée avec succès), alors **await** promise retourne le résultat.

Dans le cas d'un **rejet**, une erreur va être lancée de la même manière qui si on utilisait **throw**.

Pour **capturer** une erreur lancée avec **await**, on peut tout simplement utiliser une structure **try** / **catch**.

```
async function getNb() {
    try{
        const nombre1 = await number1();
        const nombre2 = await number2();
        return nombre1 + nombre2;
    }catch(error){
        console.log(error)
    }
}
```

Async / Await

Conclusion

Les mots clefs **async** / **await** permettent d'écrire du code **asynchrone**, ils n'ajoutent aucune **fonctionnalité** !

Mais fournissent une syntaxe plus **intuitive** et plus **claire** et utiliser des **promesse**.

Utiliser le mot clef **async** devant une fonction force la fonction à retourner une **promesse** et nous permet d'utiliser **await** dans celle-ci.