TD JS-TS

- Revoir les concepts fondamentaux de JavaScript avant d'aborder React/Next.js.
- Pratiquer la manipulation des tableaux, objets et l'asynchronisme.
- Introduire TypeScript progressivement en montrant ses avantages.

Partie 1 : Rappel des Bases de JavaScript

Exercice 1 : Portée des Variables

Objectif : Comprendre la différence entre `var`, `let` et `const`.

```
function testScope() {
    if (true) {
        var a = "var variable";
        let b = "let variable";
        const c = "const variable";
    }
    console.log(a); // Que se passe-t-il ici ?
    console.log(b); // Et ici ?
    console.log(c); // Et ici ?
}
testScope();
```

Question : Pourquoi certaines variables provoquent-elles une erreur ?

Exercice 2 : Fonctions d'Ordre Supérieur

Objectif : Utiliser `map`, `filter` et `reduce` pour manipuler des tableaux.

```
const students = [
    { name: "Alice", grade: 15 },
    { name: "Bob", grade: 9 },
    { name: "Charlie", grade: 18 }
];
```

1. Créer un tableau contenant uniquement les noms des étudiants
2. Filtrer les étudiants ayant une note supérieure ou égale à 10
3. Calculer la moyenne des notes
4. Ecrire ces fonctions avec des fonctions fléchées.

Exercice 3 : Asynchronisme

Objectif : Pratiquer `fetch` et `async/await`.

```
async function fetchUsers() {
    try {
        const response = await fetch('https://jsonplaceholder.typicode.com/users');
        if (!response.ok) {
            throw new Error('Erreur lors de la récupération des utilisateurs');
        }
        const users = await response.json();
        console.log(users);
    } catch (error) {
        console.error(error.message);
    }
}
fetchUsers();
```

Questions : Pourquoi `await` doit-il être utilisé dans une fonction `async` ? Comment gérer les erreurs de récupération des données ?

Partie 2 : Introduction à TypeScript

Exercice 4 : Ajout de Types

Objectif : Convertir un code JavaScript en TypeScript en ajoutant des types.

```
function greet(name: string, age: number): string {
    return `Bonjour ${name}, tu as ${age} ans.`;
}

console.log(greet("Alice", 25));
```

Question : Que se passe-t-il si on passe un nombre à la place d'une chaîne de caractères pour `name` ?

Exercice 5 : Interfaces et Types

Objectif : Définir des types pour mieux structurer les données.

```
interface Student {
    name: string;
    grade: number;
}

const student: Student = { name: "Alice", grade: 15 };
console.log(student);
```

Bonus : Créer un tableau de `Student` et réutiliser les mêmes fonctions d'ordre supérieur en TypeScript.

Exercice 6 : Classes en TypeScript

Objectif : Introduire les classes avec TypeScript.

```
class Person {
    name: string;
    age: number;

    constructor(name: string, age: number) {
        this.name = name;
        this.age = age;
    }

    introduce(): string {
        return `Je m'appelle ${this.name} et j'ai ${this.age} ans.`;
    }
}

const alice = new Person("Alice", 25);
console.log(alice.introduce());
```

Question : Que se passe-t-il si on oublie d'initialiser `name` ou `age` dans le constructeur ?

Réalisations

Classe HttpService

```
export default class HttpService{
    static async get(url:string):Promise<any>{
        const response=await fetch(url);
        if(response.ok) {
            return await response.json();
        }else{
            console.log(`error on ${url}`);
        }
    }
}
```

Interface

```
export default interface User{
    name: string;
    username?:string;
```

```
    email?: string;
};

const createUser=(n:string,email:string):User=>{
    return {name:n,email};
};
export {createUser};
```

Script de test

```
import HttpService from "./services/HttpService";
import User from "./interfaces/User";

HttpService.get("https://jsonplaceholder.typicode.com/users").then((users:User[])=>
{
    users.forEach(u=>console.log(u.name))
})
```