



# TypeScript

Par MIHAINGOHERILANTO Manambintsoa

ITEAM-\$

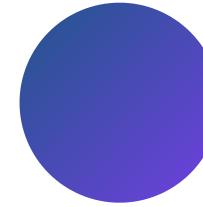


# Plan

1. Introduction
2. Installation et configuration
3. Types de données
4. Typage des fonctions
5. Interfaces
6. Conclusion



# Introduction



TypeScript c'est JavaScript avec une syntaxe pour les types

Why ?

- Fortement typé
- Facilite la compréhension
- Amélioration de la maintenance

```
type Result = "pass" | "fail"

function verify(result: Result) {
    if (result === "pass") {
        console.log("Passed")
    } else {
        console.log("Failed")
    }
}
```



# Les différences entre TypeScript et JavaScript



<b>CRITERE</b>	<b>TYPESCRIPT</b>	<b>JAVASCRIPT</b>
Typage statique	Oui	Non
Compilation	Nécessaire	Non
Performance	Légèrement plus lent que Js	Plus rapide que TS



# Installation

```
$ npm install -g typescript
```

ou

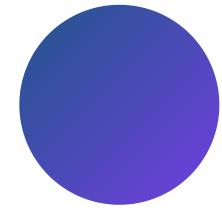
```
$ yarn global add typescript
```

```
$ tsc --version
```





# Environnement de développement



```
$ mkdir projet && cd projet
```

```
// Initialiser un nouveau projet Node.js, cela créera un fichier package.json  
$ npm init
```

```
$ npm install typescript
```

```
// Créez un fichier de configuration TypeScript, Cela créera un fichier tsconfig.json  
$ npx tsc --init  
//Ouvrez le fichier tsconfig.json et modifiez les options en fonction de vos besoins
```

```
// Créez un fichier TypeScript en utilisant l'extension de fichier .ts. Vous pouvez  
écrire votre code TypeScript dans ce fichier  
// Compilez votre fichier TypeScript en JavaScript  
$ npx tsc
```

# TypeScript: The extends keyword

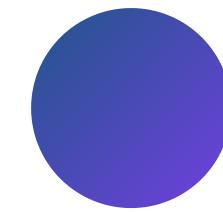


```
interface User {  
    firstName: string;  
    lastName: string;  
    email: string;
```

## Types de données



# Les types de base de TypeScript

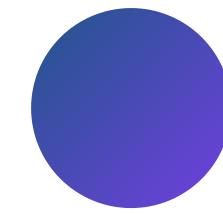


- number : représente les nombres, qu'ils soient entiers ou à virgule flottante.
- string : représente les chaînes de caractères.
- boolean : représente les valeurs booléennes true ou false.
- null et undefined : représentent respectivement les valeurs null et undefined.
- void : représente l'absence de valeur.
- any : représente n'importe quel type de donnée.
- never : représente un type qui ne peut jamais avoir de valeur.
- object : représente tout type d'objet

Exemple: voir monfichier.ts



# Les types avancés de TypeScript



- Union types : var qui peut contenir plusieurs types différents. Par exemple, une variable de type string | number
- Intersection types : les intersection types permettent de combiner plusieurs types en un seul. Par exemple, type A = { x: number } & { y: number }
- Literal types : les literal types permettent de définir des types qui ne peuvent prendre qu'une seule valeur. Par exemple, type Color = "red" | "green"
- Mapped types : les mapped types permettent de créer de nouveaux types en fonction d'un type existant. Par exemple, type Readonly<T> = { readonly [P in keyof T]: T[P] } définit un type qui rend toutes les propriétés d'un objet en lecture seule.

Exemple: voir monfichier.ts

Recherche: Les types génériques de Typescript (fonction, interface, classe générique)

# Le typage des fonctions de TypeScript



Le typage des fonctions en TypeScript est un moyen de spécifier le type des paramètres et de la valeur de retour d'une fonction

Le typage des fonctions permet à TypeScript de détecter les erreurs de type lors de la compilation plutôt qu'à l'exécution



# Exemples de typage des fonctions

- Typage de fonction simple en TypeScript:

```
function addition(a: number, b: number): number {  
    return a + b;  
}  
  
function concat(a: string, b: string, c?: string, d = 'd'): string {  
    return a + b + (c ? c : '') + d;  
}
```

- Typage de fonction en TypeScript qui retourne un objet:

```
function createUser(name: string, age: number): {name: string, age: number} {  
    return {  
        name: name,  
        age: age  
    };  
}
```



# Exemples de typage des fonctions

- Typage de fonction en TypeScript qui retourne un tableau contenant des objets:

```
interface Person {  
    name: string;  
    age: number;  
}
```

```
function createPeople(names: string[], ages: number[]): Person[] {  
    const people: Person[] = [];  
    for (let i = 0; i < names.length; i++) {  
        people.push({ name: names[i], age: ages[i] });  
    }  
    return people;  
}
```

# Interfaces

En TypeScript, une interface est une structure qui définit la forme des objets. Les interfaces sont souvent utilisées pour décrire la structure des données échangées entre différents modules d'une application

```
interface Book {  
    title: string;  
    authors: string[] | string;  
    publisher: string;  
    publishDate: Date;  
}
```

```
interface Employee {  
    id: number;  
    firstName: string;  
    lastName: string;  
    getFullName: () => string;  
}
```

```
interface Person {  
    firstName: string;  
    lastName: string;  
    age: number;  
    address?: string;  
}
```

# Conclusion

01

TypeScript ajoute une couche de typage statique au JavaScript

02

TypeScript améliore la qualité et la lisibilité de notre code

03

TypeScript est un langage de programmation en constante évolution

04

N'hésitez pas à utiliser TypeScript dans vos projets futurs



# Thank You

Par MIHAINGOHERILANTO Manambintsoa