Typescript /TS/

Javascript pour les moins nuls

Quésako ?

Typescript est une extension du langage Javascript. Il est libre et Open source et fut créé par le principal contributeur de C# chez Microsoft.

Il regroupe nombre des caractéristiques de ES6 (classes, modules) et en ajoute quelques unes (typage strict, interfaces). Transpilé, il peut être traduit en ES5 pour être rendu accessible sur tous les navigateurs.

Nous en ferons usage dans nos projets Angular qui le recommande.

Dans ce cours nous aborderons ses principales fonctions et modalités de mise en oeuvre.

Comment faire ?

Plusieurs IDE ont intégré Typescript. Pour faciliter notre approche et permettre au développement partagé, nous avons choisi Visual Studio Code comme logiciel de production. Tous les exemples de codes et de procédures se baseront sur cette solution.

Visual Studio Code est gratuit et open source. Il intègre TS par défaut.

Installation

TS peut être intégré via NPM. NPM nécessite NodeJS aussi nous faudra-t-il avoir installé ces deux technologies avec d'utiliser Typescript.

Téléchargez et installez NodeJS puis saisissez install npm dans la console.

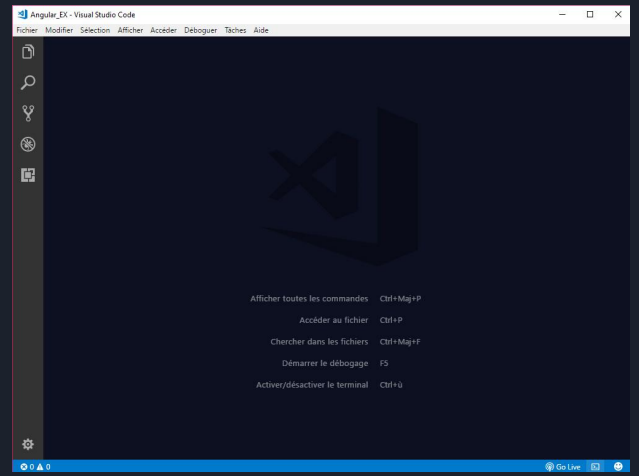
Une fois ces opérations réalisées, nous pouvons saisir cette ligne :

> npm install -g typescript // Installation en global

Puis compiler un fichier :

> tsc leNomDeMonFichier.ts

Le fichier .ts (typescript) est transpilé en Javascript.



Sur VSC, cliquer sur le flèches en bas à droite pour ouvrir la console

Les variables

Typescript est typé

Les variables

TS offre un typage strict des variables. Vu qu'il est transpilé en Javascript, ce typage est optionnel mais forcément recommandé pour la clarté du code mais surtout son débogage. En voici les principaux types.

Les traditionnelles

```
let nom:string; // Une chaîne de caractères

let phrase: string = `Salut étranger, mon nom est ${ nom}.
Je suis ${ profession } et toi ?`; // Template de chaîne

let nombre:number; // Un nombre
```

Un peu plus singulières

```
let numTab:Array<number>; // Tableau de nombres

let numTab:number[]; // Autre tableau de nombres

let numString:string | number; // Au choix
```

Types particuliers

Any

C’est étrange mais nécessaire dans certains cas. Le type any permet de définir une variable avec un type non défini. C’est notamment utile avec les promesses.

```
let maVariableNonTypee:any;
```

Typage de fonctions

Les fonctions avec return peuvent être typées :

```
function renvoieNum():number{
    return(12);
}
```

Void permet un typage sans retour :

```
function renvoieRien():void{
    console.log(12);
}
```

Public, privé, protégé

Les variables privées offrent une protection notamment lors d'héritages. Les variables protected ne peuvent pas être réécrites (avec super notamment).

Voici un exemple d'utilisation de variables privées dans des classes héritées. Dans l'exemple, la propriété 'nom' ne peut pas être réécrite.

L'utilisation de setters et getters sont une bonne pratique pour les variables privées.

```
class Animal {
  private nom: string;
  constructor(monNom: string) { this.nom = monNom; }
}

class Rhino extends Animal {
  constructor() { super("Rhino"); }
}

class Employee {
  private nom: string;
  constructor(monNom: string) { this.nom = monNom; }
}

let animal = new Animal("Goat");
let rhino = new Rhino();
let employe = new Employe("Bob");

animal = rhino;
animal = employe; // Error: 'Animal' and 'Employee' are not compatible
```

Classes & Interfaces

Les classes et les interfaces permettent de créer des types d’objets

Les classes

Typescript ajoute quelques subtilités dans l'utilisation des classes Javascript.

Classes abstraites

```
abstract class Animal {
  abstract faitDuBruit(): void;
  bouge(): void {
    console.log("crie à la lune...");
  }
}
```

Les classes abstraites ne peuvent pas être instanciées. Elles servent pour les héritages.

Un résultat similaire peut être obtenu en déclarant le constructeur protected.

Interfaces

Les interfaces permettent de définir des types d'objets. Elles possèdent un ensemble d'options spécifiques données en exemple ci-dessous.

Une interface

```
interface CarreConfig {
   c: string; // propriété strict
   l?: number; // * propriété optionnelle
   [propName: string]: any; // Tout type de propriété supplémentaire
   readonly coins:number = 4; // Equivalent de const
}

let unCarre:CarreConfig = {c:"rouge", l:20, angle:"rond"};
```

Les interfaces peuvent étendre des classes :

```
class Point {
   x: number;
   y: number;
}
interface Point3d extends Point {
   z: number;
}
let point3d: Point3d = {x: 1, y: 2, z: 3};
```

* Le point d'interrogation rend aussi optionnelle un argument de fonction