esprit
Se former autrement
HONORIS UNITED UNIVERSITIES

Introduction à TypeScript

Année universitaire

2022-2023

United Nations Educational, Scientific and Cultural Organization
UNESCO Chair "Project-based learning" ESPRIT School of engineering, Tunisia
uniTwin

EUR-ACE® Délivrée par la Commission des Titres d'Ingénieur

CONFÉRENCE DES GRANDES ÉCOLES

cdio
CONCEIVE DESIGN IMPLEMENT OPERATE

- Introduction à TypeScript
- Rappel sur JavaScript et ES6
- Nouveautés de TypeScript

Introduction à TypeScript

TypeScript - Présentation

- Langage de script typé, un sur-ensemble de Ecmascript (JavaScript)
- Support d'EcmaScript 3 et 5 et ES6.
- Release 1.0 en Avril 2014
- Compilé en JavaScript.
- Opensource, source disponible sur github

TypeScript - Caractéristiques

- Vérificateur statique : La détection d'erreurs dans le code sans l'exécuter est appelée vérification statique.
- Langage typé
- Préserve le comportement d'exécution de JavaScript
- Une fois votre code TS est compilé, le code JS brut résultant n'a aucune information de type.

TypeScript - Téléchargement

TypeScript peut être téléchargé de 3 façons:

- Npm module
- NuGet package
- Visual Studio Extension

Vous pouvez consulter https://www.typescriptlang.org/download

Rappel sur JavaScript et ES6

Déclaration de variable

C’est la même déclaration de JavaScript

On peut utiliser :

1. Var : définir une variable globale
2. Let: définir une variable dans un block de code
3. Const: définir une constante

Déclaration de variables - Var

Syntaxe: **var** maVariable :type;

Avec var:

- On peut créer plusieurs variables ayant le même nom
- La variable déclarée est accessible partout

=> Ceci peut engendrer plusieurs erreurs

Déclaration de variables - Let

Syntaxe : let maVariable :type;

- Block scoping : La variable déclarée dans un block, n'est pas visible à l'éxtérieur

```
function f(input: boolean) {
        let a = 100;
        if (input) {
                let b = a + 1;
                 return b; }
                // Error: 'b' doesn't exist here
                 return b; }
```

```
function f(x) { let x = 100; } => erreur
 function g() {
 let x = 100;
 var x = 100;} => erreur
```

- On ne peut pas créer plusieurs variables ayant le même nom
- Interdit d'utiliser une variable avant qu'elle soit déclarée

```
++a; //illégal
let a;
```

Déclaration de variables - const

```
const maVariable = valeur;
```

- Nous utilisons const pour des variables où la valeur ne change pas.
- La visibilité des variables crées avec const est comme celles crées avec let.

Les conditions

```
if (condition) {
.....traitement....
}
elsif(condition){
.....traitement....
}
else (condition){
.....traitement....
}
```

Les Boucles

Do {

…..traitement….

}

While (condition)

While (condition){

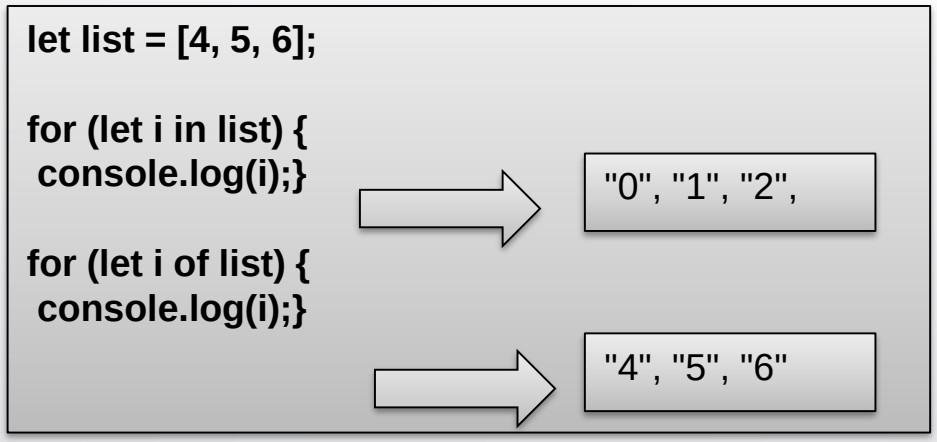
…..Traitement……

};

Les Boucles

Deux boucles for possibles **for … of** et **for ….in**

1. **For … in** : Renvoie une liste de clés sur l'objet en cours d'itération
2. **For … of :** Renvoie une liste de valeurs des propriétés de l'objet en cours d'itération



Les types

Il existe 6 types primitifs en Java script :

- String
- Number
- Null (no value)
- Undefined (une variable déclarée mais sans aucune initialisation)
- Boolean
- Symbol

Le reste est objet : objet classique, function, Array, Date

Les types - primitifs

Type	Syntaxe	Fonctions relatives

Number	let val1: number = 6;
	let val2: number = 0xf00d;
	let big: bigint = 100n; (for Big integers)
		Fonctions sur les nombres :
		Number() - Number.isNaN() - Number.parseInt() - Number.parseFloat(

String	let color: string = "blue";	Il y a beaucoup de propriétés et de méthodes sur la manipulation de chaînes de caractères. Exemples : length - includes() - indexOf() - slice() - split() - substring() - toLowerCase() / toUpperCase()
Boolean	Chaque variable sans valeur est évaluée à false 0 / “”/ undefined / null Chaque variable avec valeur est évaluée à true 100 / 3.14 / -15 / "Hello" / "false" / 7 + 1 + 3.14	

Les types - objet

Type	Syntaxe
Tuple : tableau avec un nombre fixe d'éléments ayant des types connus mais différents	Déclaration: let x: [string, number]; Initialization: x = ["hello", 10]; // OK
Enum	enum Color { Red, Green, Blue, } let c: Color = Color.Green;
Unknown (le type est non reconnu à l'avance)	llet notSure: unknown = 4; notSure = "maybe a string instead"; // OK, definitely a boolean notSure = false;

Les types - objet

Type	Syntaxe
Object : aucun des ces types (number, string, boolean , null, undefined)	declare function create(o: object | null): void; create({ prop: 0 }); ⇔ OK create(42); create("string"); ⇔ ce n'est pas le type Object
Any (n'importe quel type ⇔ JS)	let looselyTyped: any = 4;
Void : pour une fonction qui ne retourne rien	function warnUser(): void { console.log("This is my warning message"); }
Null / Undefined	let u: undefined = undefined; let n: null = null;

Les types - Les fonctions - syntaxes

```
// syntaxe classique
function myFunction (param1, param2){
...
}
```

```
// arrow function (ES6)
const myFunction= (param1, param2)=>{
...
}
```

```
// fonction anonyme
function(param1, param2){
.....
}
```

Les types - Les fonctions - syntaxes

```
// arrow function (ES6)
const myFunction= (param1, param2)=>{
...
}
```

```
// arrow function (ES6)
const myFunction= ()=>{
...
}
```

```
// arrow function (ES6)
const myFunction= (param1)=>{
...
}
```

```
// arrow function (ES6)
const myFunction= param1=>{
...
}
```

Les types - Les fonctions - syntaxes

- Les fonctions anonymes sont déjà utilisés comme des callback

```
// fonction anonyme
[1,2,3].forEach(function(element){
    console.log(element)
  }
)
```

```
// fonction anonyme
[1,2,3].forEach((element)=>{
    console.log(element)
  }
)
```

Les fonctions paramétrées

Paramètre optionnel

TypeScript permet de rendre des paramètres optionnels. Ces paramètres doivent être placés à la fin

```
function afficheNom(nom: string, prenom?: string): void {
let texte = nom;
if (prenom) {
texte += ' ' + prenom;
}
alert(texte);
}
afficheNom('Durand');
afficheNom('Dupont', 'Marcel');
```

Les fonctions paramétrées

Paramètres du reste

Des fois on ne sait pas de combien de paramètres on a besoin. On utilise alors les paramètres du reste

```
function ajouter(base, ...elements) {
for (var i = 0; i < elements.length; i++) {
base += elements[i];
}
return base;
}
var resultat = ajouter(10, 1, 2);
alert(resultat);
```

- On ajoute au premier paramètre tous les autres transmis, quel qu'en soit le nombre.

Les fonctions – Type de retour

- Une fonction qui retourne une valeur ayant un type.

```
function carre(x: number): number {
        Return x * x ;
        }
```

- Si la fonction ne renvoie pas de valeur on lui donne le type **void** :

```
function affiche(texte: string): void {
        alert(texte);
        }
```

Les classes - Définition

- Comme dans ES6, il y a la notion de classes

```
class Salutation {
    saluer: string;

    Constructor(message: string) {
        this.saluer = message;
    }
    greet() {
        return "Hello, " + this.saluer;
    }
}
let Salut = new  Salutation("world");
```

Les classes - Héritage

```
class Etudiant extends Personne{ // class Etudiant herite de Personne
  classe: string; // attribut

  constructor(name: string, classe: string) { // contructeur
    super(name); // appel constructeur de la class mère
    this.classe= classe;
  }

  initiation(): string { // une methode
    return “Bonjour!"
  }
}
//instanciation
var MonEtudiant = new Etudiant(Mohamed',5GL');

//appel d'un methode
MonEtudiant,initiation();
```

Les modules

- À partir d'ECMAScript 2015, JavaScript a un concept de modules. TypeScript partage ce concept.
- Les modules sont exécutés dans leur propre étendue, pas dans la portée globale; cela signifie que les variables, fonctions, classes, etc. déclarées dans un module ne sont pas visibles à l'extérieur du module à moins qu'elles ne soient explicitement exportées en utilisant l'un des formes d'exportation.
- Inversement, pour consommer une variable, une fonction, une classe, une interface, etc. exportée depuis un module différent, il faut l'importer à l'aide de l'un des formulaires d'importation.

Les modules

- Dans TypeScript, tout comme dans ECMAScript 2015, tout fichier contenant une importation ou une exportation de niveau supérieur est considéré comme un module. Inversement, un fichier sans aucune déclaration d'import ou d'export de niveau supérieur est traité comme un script dont le contenu est disponible dans la portée globale (et donc aux modules également).

```
import { StringValidator } from "./StringValidator";

export const numberRegexp = /^[0-9]+$/;

export class ZipCodeValidator implements StringValidator {
  isAcceptable(s: string) {
    return s.length === 5 && numberRegexp.test(s);
  }
}
```

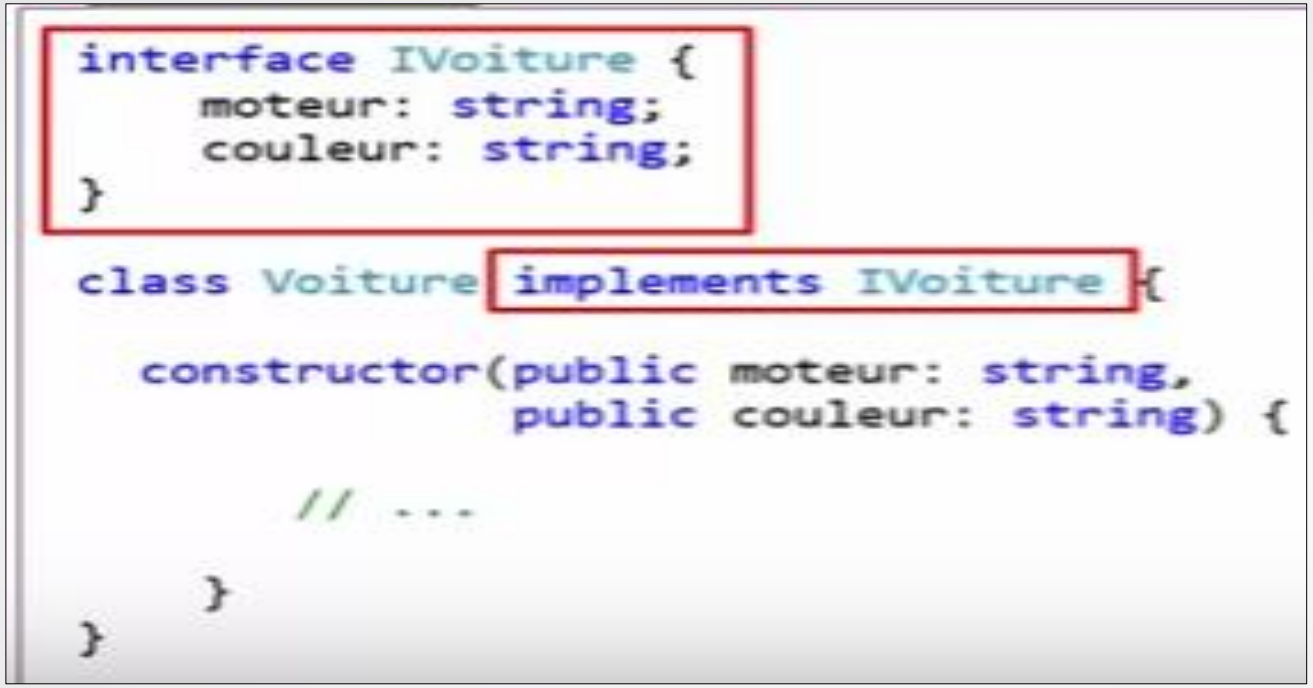
Nouveautés de TypeScript/ES6

Nouveautés

Les nouveautés de TypeScript par rapport à JavaScipt lors de son apparition

- un typage statique optionnel des variables et des fonctions
- la création de classes et d'interfaces
- l'import de modules

Les interfaces



```
interface IVoiture {
    moteur: string;
    couleur: string;
}

class Voiture implements IVoiture {

  constructor(public moteur: string,
              public couleur: string) {

      // ...

    }
}
```

Les Enumérations

- Les énumérations sont l'une des rares fonctionnalités de TypeScript qui ne sont pas une extension au niveau du type de JavaScript.
- Les énumérations permettent à un développeur de définir un ensemble de constantes nommées. L'utilisation d'énumérations peut faciliter la documentation de l'intention ou la création d'un ensemble de cas distincts. TypeScript fournit à la fois des énumérations numériques et basées sur des chaînes.

```
enum Direction {
  Up = 1,
  Down,
  Left,
  Right,
}
```

```
enum Direction {
  Up,
  Down,
  Left,
  Right,
}
```

Exemple de transpilation de TS vers JS

```
interface IPersonne{
  parler();
  estMajeur():boolean;
}
public class Personne implements IPersonne{
  private _nom:string;
  private _prenom:string;
  private _age:number;

  constructor(nom:string,prenom:string){
    this._nom=nom;
    this._prenom=prenom; }

    public parler(){
      console.log("");
    }

    public estMajeur():boolean{
      return this._age >= 18;
    }

}
```

```
var Personne = /** @class */ (function () {
    function Personne(nom, prenom) {
        this._nom = nom;
        this._prenom = prenom;
    }
    Personne.prototype.parler = function () {
        console.log("");
    };
    Personne.prototype.estMajeur = function () {
        return this._age >= 18;
    };
    return Personne;
}());
```

Les décorateurs

- Un décorateur est un type spécial de déclaration qui peut être attaché à une déclaration de classe, une méthode, un accesseur, une propriété ou un paramètre.
- Les décorateurs permettent d'ajouter à la fois des annotations et une syntaxe de méta-programmation pour les déclarations de classe et les membres.
- Les décorateurs utilisent la forme @expression, où expression doit s'évaluer en une fonction qui sera appelée au moment de l'exécution avec des informations sur la déclaration décorée.

Evaluation d’un décorateur

Il existe un ordre bien défini sur la façon dont les décorateurs appliqués à diverses déclarations à l'intérieur d'une classe sont appliqués :

- *Les décorateurs de paramètre* , suivis de *méthode*, d' *accesseur* ou de décorateurs de propriété sont appliqués pour chaque membre d'instance ou bien chaque membre statique.
- *Les décorateurs de paramètres* sont appliqués pour le constructeur.
- *Les décorateurs de classe* sont appliqués pour la classe.

Les décorateurs dans Angular

Décorateur Angular	Exemple
Décorateur de classe	@Component, @NgModule
Décorateur de propriété pour les propriétés dans une classe	@Input, @Output
Décorateur de méthode pour les méthodes dans une classe	@HostListener
Décorateur de paramètre pour les paramètres dans une classe	@Inject

Références

- https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html

Merci de votre attention