esprit
Se former autrement

TypeScript

Année universitaire

2020-2021

CONFÉRENCE DES GRANDES ÉCOLES

cdio
CONCEIVE DESIGN IMPLEMENT OPERATE

AUF Agence UNIVERSITAIRE DE LA FRANCOPHONIE
IFGU INSTITUT DE LA FRANCOPHONIE POUR LA GOUVERNANCE UNIVERSITAIRE

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

Plan:

- TypeScript
- Déclaration de variables
- Les types de bases
- Boucle For
- Les classes
- Les interfaces
- Les fonctions
- Les modules
- Les décorateurs

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.
- Dernière version au début de septembre 2020 :

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.

Déclaration de variables (1/4)

1. Déclaration avec "var"

var maVariable :type;

C’est la syntaxe de JavaScript.

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 (2/4)

2. Déclaration avec "let"

let maVariable :type;

Avec let:

- On ne peut pas créer plusieurs variables ayant le même nom

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

- Interdit d'utiliser une variable avant qu'elle soit déclarée

```
++a; //illégal
let a;
```

Déclaration de variables (3/4)

- 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;
}
```

Déclaration de variables (4/4)

3. Déclaration avec "const"

```
const maVariable = valeur;
```

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

Les types de bases (1/4)

Les types de bases sont les même qu'en Javascript:

Type	Syntaxe

Number	let val1: **number** = 6;
	let val2: **number** = 0xf00d;
	let big: **bigint** = 100n; (for Big integers)

String	let color: **string** = "blue";
Array	let list: **number[]** = [1, 2, 3]; ⇔syntaxe : **type[]** ou bien let list: **Array<number>** = [1, 2, 3]; ⇔syntaxe : **Array<type>**

Les types de bases (2/4)

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 de bases (3/4)

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 de bases (4/4)

Type	Syntaxe

Never	// fonction qui retourne toujours des exceptions
	error(message: string): never {
	throw new Error(message);
	}
	
	// le type retourné est never
	function fail() { return error("Something failed"); }
	
	//pas de retour (ne se termine pas
	function infiniteLoop(): never { while (true) {} }

Boucle For

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.

```
let list = [4, 5, 6];

for (let i in list) {
 console.log(i);}
```

→ "0", "1", "2",

```
for (let i of list) {
 console.log(i);}
```

→ "4", "5", "6"

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

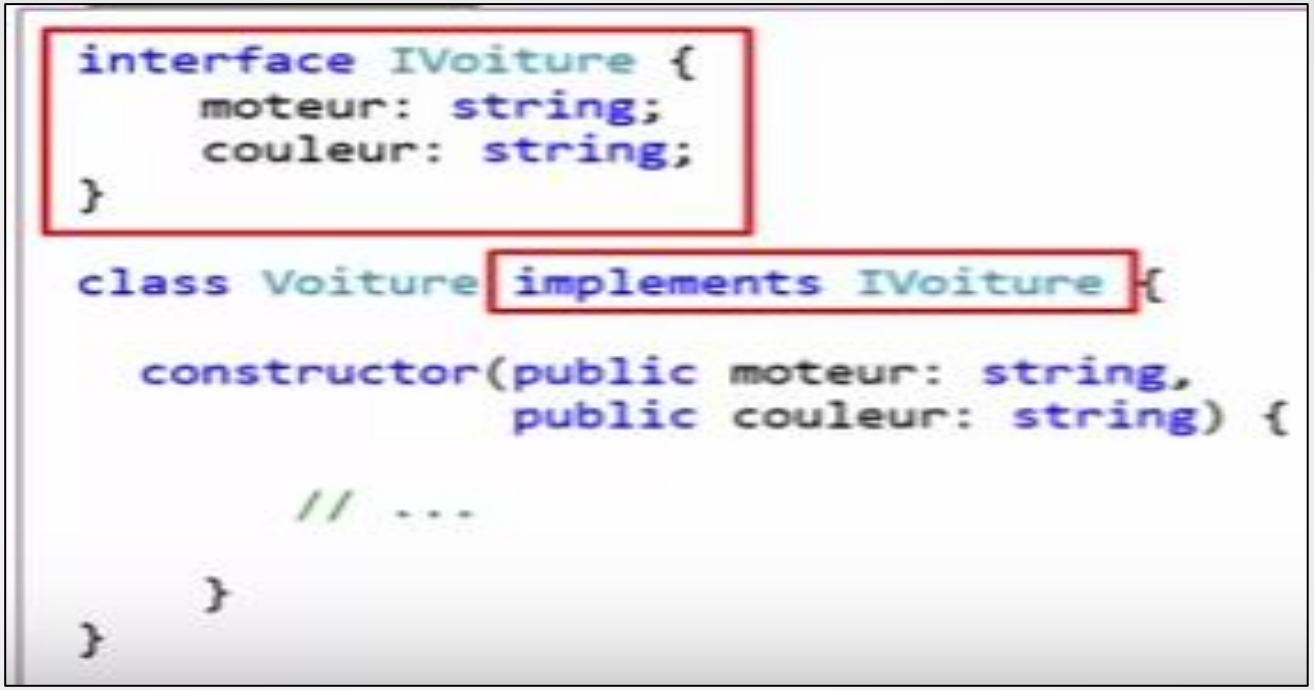
```
class Etudiant extends Personne{ // class Etudiant herite de Personne
  classe: string; // attribut

  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 interfaces



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

class Voiture implements IVoiture {

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

      // ...

    }
}
```

Les fonctions – paramètres (1/2)

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 – paramètres (2/2)

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 – types 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 modules (1/4)

- 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.

Les modules (2/4)

- 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 (3/4)

- 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).

Les modules (4/4)

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

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

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

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 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.
- Il y a des décorateurs prédéfinis et on peut créer des décorateurs personnalisés.

Les décorateurs de 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

Néthographie

- https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html (dernière consultation le 04/09/2020)