



## TP2 : INITIATION AU LANGAGE TYPESCRIPT

Matière : Atelier Framework Coté Client

Enseignante : Sonia Guerbouj

Classes : DSI2, MDW2

Durée : 3h

### Objectif

Le but de ce TP est de présenter le langage de programmation TypeScript aux étudiants et de les initier à son utilisation (variables, fonctions, modules).

### 1- Présentation de TypeScript

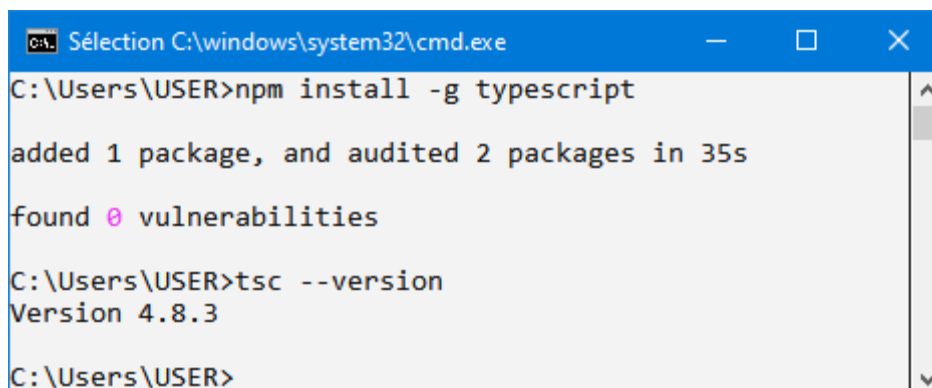
C'est un langage compilé récent (février 2012) conçu par Microsoft, et précisément par Anders Hejlsberg, le concepteur du langage C#. Le but premier de TypeScript est de rendre plus facile et plus fiable l'écriture de code en JavaScript pour des applications de grande ampleur.

En effet, JavaScript, étant un langage faiblement typé et interprété, il ne permet pas de détecter les erreurs au moment du codage. De son côté TypeScript qui est un langage fortement typé, compilé et orienté objet, dispose d'un compilateur dédié qui résout ce problème. Cela permet d'éviter les comportements inattendus de l'application.

### 2- Installation de l'environnement de développement

Pour travailler avec TypeScript, il convient d'abord d'installer *NodeJS* (qui fournit l'outil *npm*). Puisque cette étape a été réalisée lors du TP1, on passe directement à l'installation du package typescript.

Ouvrez une fenêtre d'éditeur de commande et tapez : `node install -g typescript`  
Vérifiez aussi la version de typescript en tapant : `tsc --version`



```
Sélection C:\windows\system32\cmd.exe
C:\Users\USER>npm install -g typescript
added 1 package, and audited 2 packages in 35s
found 0 vulnerabilities

C:\Users\USER>tsc --version
Version 4.8.3

C:\Users\USER>
```

### 3- Premier exemple

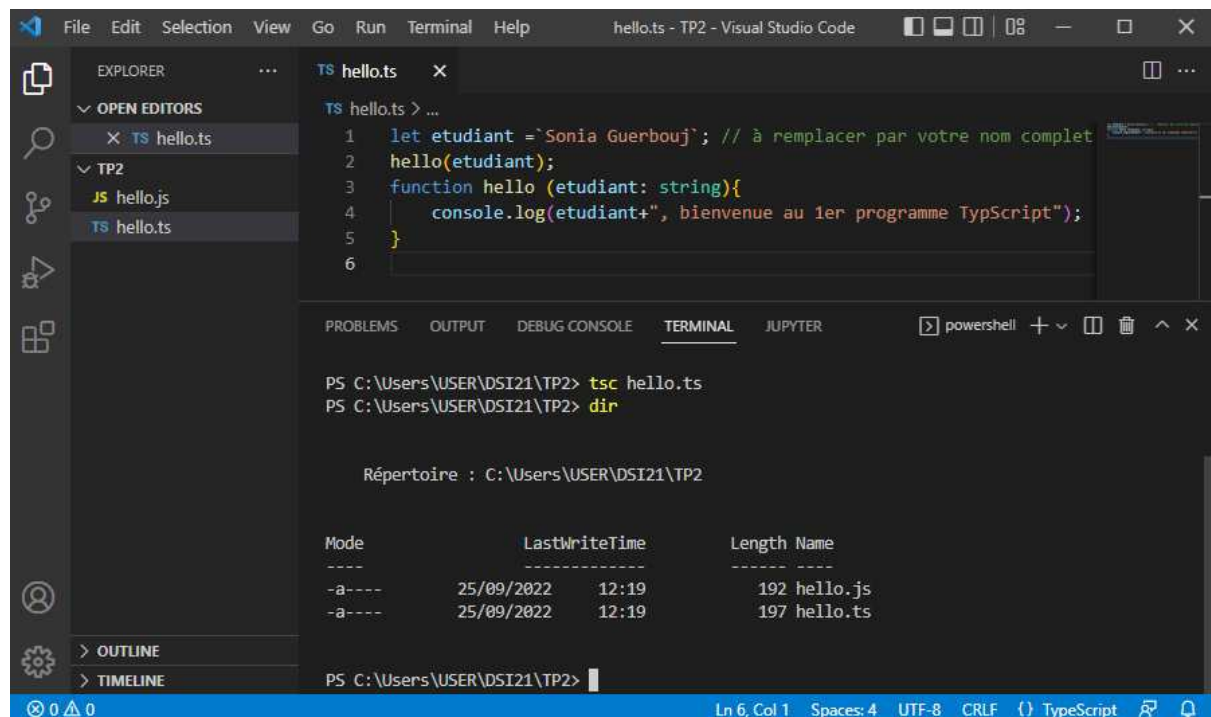
Créez un répertoire « TP2 » sous « DSI21 ». A l'aide de VScode créez un fichier « hello.ts » dans lequel tapez l'exemple suivant :

```
let etudiant = `Sonia Guerbouj`; // à remplacer par votre nom complet
hello(etudiant);
function hello (etudiant: string){
    console.log(etudiant+", bienvenue au 1er programme TypeScript");
}
```

1) Compilez le fichier hello.ts, il faut d'abord se placer dans le répertoire qui le contient

```
PS C:\Users\sonya> cd 'DSI21\TP2'
PS C:\Users\sonya\DSI21\TP2> tsc hello.ts
```

S'il n'y a pas d'erreurs, la compilation génère un fichier javaScript. En effet, les fichiers typeScript (.ts) ne sont pas utilisés directement sur le navigateur.



2) Ouvrez le fichier généré « hello.js ». Quelles différences avec le fichier « hello.ts » ?

3) Exécutez à présent le fichier javaScript à l'aide de NodeJS : `node hello.js`

4) Modifiez le code du fichier « hello.ts » pour que la fonction « hello() » retourne une chaîne à afficher dans la console lors de l'appel de la fonction. Testez le programme.

### 4- Exercice 1 : Les fonctions

1) Créez un fichier « fct.ts » sous « TP2 », dans lequel écrivez les fonctions suivantes :

- estPremier (a: number): boolean retourne vrai si le nombre a est premier (uniquement divisible sur lui-même et 1).
  - afficherPremier (a: number): void affiche à la console si un nombre a est premier ou pas.
- 2) Ajoutez un appel à la fonction « afficherPremier » pour les entiers dans l'intervalle [1..20], puis compilez et exécutez.
- 3) Créez un nouveau fichier « fctParam.ts » dans lequel écrire le code suivant :

```
function infoPays(nom: string, superficie?: number, langue='Arabe'): void {
    let infos = nom;
    if (superficie)
        infos = infos + ' : ' + superficie + 'Km²';
    if (langue)
        infos += ' parle ' + langue ;
    return(infos);
}
```

- 4) Ajoutez 2 lignes dans lesquelles appelez la fonction infoPays, une fois avec les 2 paramètres « Tunisie » et « 163 610 » et une autre en ajoutant un 3ème paramètre « Tunisien ». Testez le programme. Qu'affiche-t-il ?
- 5) On voudrait maintenant calculer le total des notes obtenu par un etudiant mais on ignore combien de note possède-t-il. Créez le fichier « nbParamFct.ts » suivant :

```
function additionner(total, ...notes) {
    for (var i = 0; i < notes.length; i++) {
        total += notes[i];
    }
    console.log (total);
}
```

Testez avec 3 puis 5 notes différentes.

## 5- Exercice 2 : Les classes

TypeScript permet d'utiliser les notions de la programmation orientée objet et notamment les classes. Soit la classe voiture définie comme suit :

```
class voiture {

    marque : string;    // propriété 1
    couleur : string;   // propriété 2

    constructor (m:string, c:string) { // constructeur permet
        this.marque = m;                // de créer des objets
        this.couleur = c;
    }
    quiSuisJe() { // méthodes 1
        return ("Voiture "+this.marque+" "+this.couleur);
    }
}
```

```
        klaxonner(son:string) { // méthode 2
            console.log (son);
        }
    } // fin class voiture

let v = new voiture ("BMW","bleu");
console.log(v.quiSuisJe());
v.klaxonner("Bip Bip");
```

- 1) Créez un fichier : « **exClass.ts** » dans lequel définir une classe qui décrit un animal de votre choix de la même manière qu’avec la classe voiture : il doit y avoir 2 attributs, un constructeur et 2 méthodes.
- 2) Créez un objet de cette classe et faites appel à ses méthodes comme dans l’exemple.

### 5- Exercice 3 : Les modules

TypeScript offre la possibilité d’organiser le code dans des modules afin de faciliter la réutilisation. Chaque module est enregistré dans un fichier à part. L’appel à un module dans un autre fichier se fait à l’aide de la syntaxe import.

Soit le fichier « **moduleVoiture.ts** » qui déclare et exporte un module comme suit :

```
export const nbVoitures = 50 ;
export class voiture {
    marque : string;
    couleur : string;
    constructor (m:string, c:string) {
        this.marque = m;
        this.couleur = c;
    }
    quiSuisJe() {
        return ("Voiture "+this.marque+" "+this.couleur);
    }
    klaxonner(son:string) {
        console.log (son);
    }
}
```

Soit maintenant le fichier « **testerModule.ts** » dans lequel on importe le module précédent.

```
Import {voiture} from './moduleVoiture' ;
let v = new voiture ("BMW","bleu");
console.log(v.quiSuisJe());
v.klaxonner("Bip Bip");
```

- 1) Créez 2 fichiers : « **moduleAnimal.ts** » et « **testerAnimal.ts** » dans lesquels séparez la déclaration d’un module, contenant votre classe de l’exercice1, de son utilisation comme dans l’exemple expliqué ci-dessus.
- 2) Compilez et testez.

## ANNEXE

### ▪ Quelques nouveautés de typeScript (ECMAScript6 ou JavaScript 2015)

<b>let</b>	"let" permet de définir les variables comme le mot-clé <b>var</b> mais la portée de la variable est restreinte au bloc courant.
<b>const</b>	Un mot-clé permettant de définir une variable dont la valeur ne change pas et la portée de <b>const</b> est similaire à celle de <b>let</b> .
<b>Template strings</b>	C'est le nouveau format qui permet d'écrire les chaînes de caractères en utilisant des anti-quotes (``) au lieu des guillemets ou des apostrophes.
<b>Classes</b>	Comme la plupart d'autres langages de programmation orientés objet, elles permettent de créer des objets.
<b>Map</b>	<b>Map</b> est une collection des paires de clé et valeur.
<b>Set</b>	<b>Set</b> est un ensemble (tableau) des valeurs uniques.
<b>For-of</b>	<b>For-of</b> est une nouvelle syntaxe qui remplace <b>foreach</b> .
<b>Arrows</b>	<b>Arrow</b> (=>) permet d'écrire les fonctions fléchées qui sont souvent utilisées pour créer des fonctions anonymes ou remplacer des fonctions normales.

### ▪ Les types prédéfinis

Type	Valeurs possibles	Exemple
<b>boolean</b>	true ou false	let exist = false ;
<b>number</b>	Décimal, Hexadécimal, Binaire, Octet	let entier : number = 123 ; let decimal : number = 3.14; //un réel let hexa : number = 0xf00a; // un hexa // un binaire let binaire : number = 0b0011; let octet : number = 0o744; //un octet
<b>string</b>	Chaînes de caractères	let pays : string = 'Tunisie' ;
<b>any</b>	N'importe quelle valeur	let joker : any = 'ordi'; joker = 100; joker = 15.5; joker = true;
<b>array</b>	Tableau d'un type de donné	let entiers : number[] = [5,22,3]; let chaines:string[]=['bonjour','toi']; // pour avoir un mélange des types nous allons utiliser le tableau de type any let inconnu:any[]=['costume',44,1.80];
<b>enum</b>	énumérateur	enum Color {Rouge, Vert, Bleu} let c: Color = Color.Vert;

<b>tuple</b>	Tableau composé de couple de clés et valeurs	<code>let voitures : [string, number] = ["Renault", 2022];</code>
<b>void</b>	undefined ou null	<code>function msgErr (): void { console.log ("Erreur !"); }</code>
<b>undefined</b>	undefined	<code>let ovni : undefined = undefined ;</code>
<b>null</b>	null	<code>let vide : null = null ;</code>

### ▪ Les fonctions

<b>Fonction nommée</b>	<code>function carre(x) { return x * x; } function carre2(x) : void {     console.log(x * x); }</code>
<b>Fonction anonyme</b>	<code>let maFonction = function(x) { return x * x; }</code>
<b>Fonction fléchée</b>	<code>function afficherMsg(msg: string) {     this.msg = msg;     this.timer = setInterval(         () =&gt; {             console.log(this.msg);         },         2000); }</code>