Angular

Formation débutant

https://angular.io

Présentation

- Début de l'année 2010, sortie d'AngularJS
  Créateur Misko Hevery (Google)
  But : Faciliter le développement des applications web
- En 2014, sortie d'Angular 2
  Réécriture **totale**, apparition du **TypeScript**, mise en place des versions **SEMVER** (x.x.x)
- Aujourd'hui en ***version 14***,
  une version **majeure** tous les **six mois**
  une à trois versions **mineures** pour **chaque** version **majeure**
  un patch par **semaine**

Le typescript

Un javascript dopé

Les caractéristiques principales

- Cross-platform
- Langage orienté objet
- Langage typé

Par où commencer

https://www.typescriptlang.org/play

Installer NodeJS

Vérifions si NodeJS est déjà installé.

```
node --version
```

Si la version est affiché, tout va bien.
Sinon, il va falloir l'installer...

https://nodejs.org/en/download/

Installer Typescript en mode global

Vérifions si Typescript est déjà installé.

```
tsc --version
```

Pour l'installer,

```
npm install -g typescript
```

Installer Typescript dans un projet

Création d'un projet

```
npm init
```

Ajout de Typescript

```
npm install --save-dev typescript
```

Initialiser Typescript

Typescript a besoin d'un fichier de configuration pour fonctionner.

```
tsc --init
```

Exemple de tsconfig.json:

```
{
  "compilerOptions": {
    "target": "es2016",
    ...
  }
}
```

Syntaxe

Types de données

Number
String
Boolean
Array
Tuple

Enum
Union
Any
Void
Never

Les nombres

Exemple:

```
let int: number = 123;
let float: number = 123.456;
let hexa: number = 0x37CF;
let octal: number = 0o377;
let binary: number = 0b111001;

console.log(int); // 123
console.log(float); // 123.456
console.log(hexa); // 14287
console.log(octal); // 255
console.log(binary); // 57
```

Les chaines

Exemple:

```
let single_quote: string = 'I\'m Groot!';
let double_quote: string = "I'm Groot!";
let back_quote: string = `I'm Groot!`;

console.log(single_quote); // I'm Groot!
console.log(double_quote); // I'm Groot!
console.log(back_quote); // I'm Groot!
```

Les booléens

Exemple:

```
let IAmTrue: boolean = true;
let IAmFalse: boolean = false;

console.log(IAmTrue); // true
console.log(IAmFalse); // false
```

Les tableaux

Exemple:

```
let tab1: string[] = ['I', 'am', 'Groot', '!'];
let tab2: Array<string> = ['I', 'am', 'Groot', '!'];
let tab3 = [1, 2, 'I', 'am', 'Groot', '!', true];

console.log(tab1); // [ 'I', 'am', 'Groot', '!' ]
console.log(tab2); // [ 'I', 'am', 'Groot', '!' ]
console.log(tab3); // [ 1, 2, 'I', 'am', 'Groot', '!', true ]
```

Les tableaux multi types

Exemple:

```
let tab4: (string | number)[] = ['I', 1, 'am', 2, 'Groot', 3, 4, '!'];
let tab5: (string | number)[] = ['I', false]; // Error !

console.log(tab4); // [ 'I', 1, 'am', 2, 'Groot', 3, '!', 4 ]
```

Les tableaux Map

Exemple:

```
let map = new Map();
map.set("key1", "value1");

console.log(map.get("key1")); // value1
```

Les tuples

Un tuple est un regroupement de données

Exemple:

```
let tuple1: [number, string] = [1, "I am Groot!"];

console.log(tuple1); // [ 1, 'I am Groot!' ]
```

Les tableaux de tuples

Exemple:

```
let kp: IKeyPair;
kp["one"] = 1;
kp["two"] = 2;
let tuple2: [number, string][] = [[1, "I"], [2, "am"], [3, "Groot"], [4, "!"]];

console.log(tuple2); // [ [ 1, 'I' ], [ 2, 'am' ], [ 3, 'Groot' ], [ 4, '!' ] ]
```

Les énumérations

Exemple:

```
enum compter {
    zero,
    one,
    ten = 10,
    more = "MORE"
}

console.log(compter.zero); // 0
console.log(compter.one); // 1
console.log(compter.ten); // 10
console.log(compter.more); // MORE
```

Les unions

Exemple:

```
let union: string | number;
union = "First";

console.log(union); // First

union = 1;

console.log(union); // 1
```

Any

Exemple:

```
let all_types: any;
all_types = "First";
all_types = 1;
all_types = true;

console.log(all_types); // true
```

Void

Exemple:

```
let nada: void;

let kp: IKeyPair;
kp["one"] = 1;
kp["two"] = 2;
function rien(): void {
    console.log("Aucun return");
}

nada = rien();
console.log(nada); // undefined
```

Never

Ce type est un peu particulier, il annonce qu'un fonction n'aura jamais de retour.

Exemple:

```
function error(msg: string): never {
    throw new Error(msg);
}
function turnAround(): never {
    while(true) {
        console.log("You shall not pass !!!");
    }
}
```

Les constantes

```
const ma_constante: string = "sa valeur";
ma_constante = "sa nouvelle valeur"; // Uncaught TypeError: Assignment to constant variable.
```

Les variables

```
let ma_variable: string = "sa valeur";
ma_variable = "sa nouvelle valeur";
```

Les fonctions

Fonction nommée

```
function ma_fonction(param: String): void {
    // mes actions
}
```

Fonction anonyme

```
const ma_fonction: Function = function(param: string): void {
    // mes actions
};
```

Fonction fléchée

```
const ma_fonction: Function = (param: string): void => {
    // mes actions
};
```

```
const ma_fonction: Function = (param: String, ...names: string[]): void => {
    // mes actions
};
```

Les interfaces

```
interface IEmployee {
    code: number;
    name: string;
    managerCode?: number;
    getSalary: () => number;
    getManager: () => number;
};
```

```
interface IKeyPair {
    key: number;
    value: string;
};

let kp: IKeyPair = {key: 1, value: "valuer"};
```

```
interface IKeyPair {
    [key: string]: number;
};

let kp: IKeyPair;
kp["one"] = 1;
kp["two"] = 2;
```

```
interface IVehicle {
    type: string;
};
interface ICar extends IVehicle {
    brand: string;
    model: string;
    color: string;
    nbSeats: number;
};
```

Ici, l'interface `ICar` étend l'interface `IVehicle`.

Les classes

```
class Employee {
    private code: number;
    private name: string;

    constructor(code: number, name: string) {
        this.code = code;
        this.name = name;
    }

    getName(): string {
        return this.name;
    }
}

let emp1: Employee = new Employee(1, "Toto");
```

```
class Vehicle {
    protected type: string;
    constructor(type: string) {
        this.type = type;
    }
}
class Car extends Vehicle {
    private brand: string;
    constructor(type: string, brand: string) {
        super(type);
        this.brand = brand;
    }
    getType(): string {
        return this.type;
    }
}

let car1: Car = new Car("Voiture", "Peugeot");
console.log(car1.getType());
```

```
interface IVehicle {
    type: string;
    brand?: string;
    nbSeats: number;
    getNbSeats: () => number;
}
class Car implements IVehicle {
    type: string;
    nbSeats: number;
    constructor(type: string, nbSeats: number) {
        this.type = type;
        this.nbSeats = nbSeats;
    }
    getNbSeats(): number {
        return this.nbSeats;
    }
}

let car1: Car = new Car("Voiture", 5);
console.log(car1.getNbSeats());
```

L'abstraction

TypeScript propose de créer des classes abstraites.
Une classe abstraite est une classe qui n'est pas instanciable directement.

```
abstract class Person {
    name: string;
    constructor(name: string) {
        this.name = name;
    }
    abstract find(name: string): Person;
}
```

```
class Employee extends Person {
    code: number;
    constructor(code: number, name: string) {
        super(name);
        this.code = code;
    }
    find(name: string): Person {
        return new Employee(1, name);
    }
}

let emp1: Person = new Employee(1, "Toto");
console.log(emp1.find("Titi"));
```

Les qualifiers d'accès

- **public**
  membre(s) accessible par tout le monde
- **private**
  membre(s) accessible uniquement par les éléments de la classe/instance
- **protected**
  membres(s) idem que private mais les classes dérivées peuvent y accéder

Les contraintes

- **read-only**
  propriété accessible uniquement en lecture
- **static**
  propriété associée à une classe (scope classe)

Les modules

L'exportation d'un élément sera réalisé par l'utilisation du mot clef **export**

L'importation d'un élément sera réalisé par l'utilisation du mot clef **import**

```ts
// export.ts

export const AGE: number = 25;
```

```ts
// index.ts

import { AGE } from "./export";

console.log(AGE); // 25
```

```
// index.ts

import * as all_export from "./export";

console.log(all_export.AGE); // 25
```

Les namespaces

```ts
// export.ts

namespace Export {

    export const AGE: number = 25;

}
```

```ts
// index.ts

/// <reference path="export.ts" />

console.log(Export.AGE); // 25
```

L'asynchronisme

Il y a trois types d'asynchronisme.

- callback
- promise
- await/async

Le callback

Il consiste à appeler une fonction passée en paramètre.

```
function main(cb: Function): void {
    cb();
}
main(() => console.log("I am Groot !"));
```

La promesse

```
function faireQqc() {
  return new Promise((successCallback, failureCallback) => {
    console.log("C'est fait");
    if (Math.random() > .5) { // réussir une fois sur deux
      successCallback("Réussite");
    } else {
      failureCallback("Échec");
    }
  })
}

const promise = faireQqc();
promise.then(successCallback, failureCallback);
```

Le async/await

```
async function fun() {

    let promise = new Promise((resolve, reject) => {
        setTimeout(() => {
            resolve("Done!");
            console.log("Inside promise");
        }, 1000);
    });

    let result = await promise; // on attend la résolution de la promesse

    console.log(result); // "Done!"
}

fun();
```

Angular

Angular CLI

Angular dispose d'une série d'outil.