Par NIDHAL JELASSI
nidhal.jelassi@fsegt.utm.tn

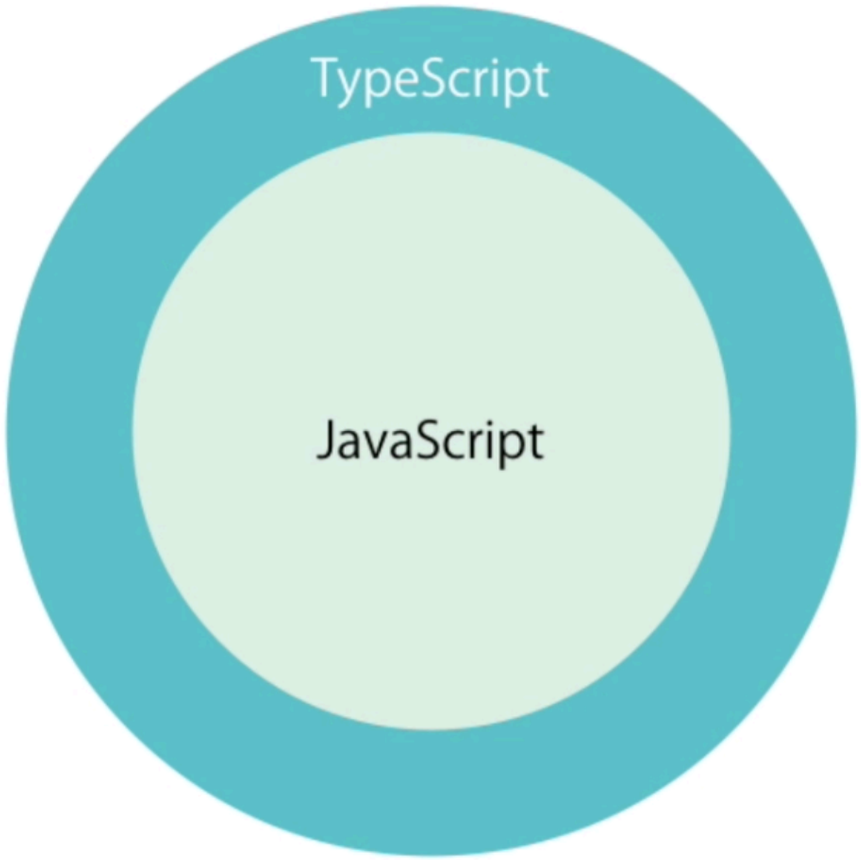
PROGRAMMATION WEB AVANCÉE

ANGULAR

Chapitre 2 : Introduction à Typescript

TYPESCRIPT, C'EST QUOI ?

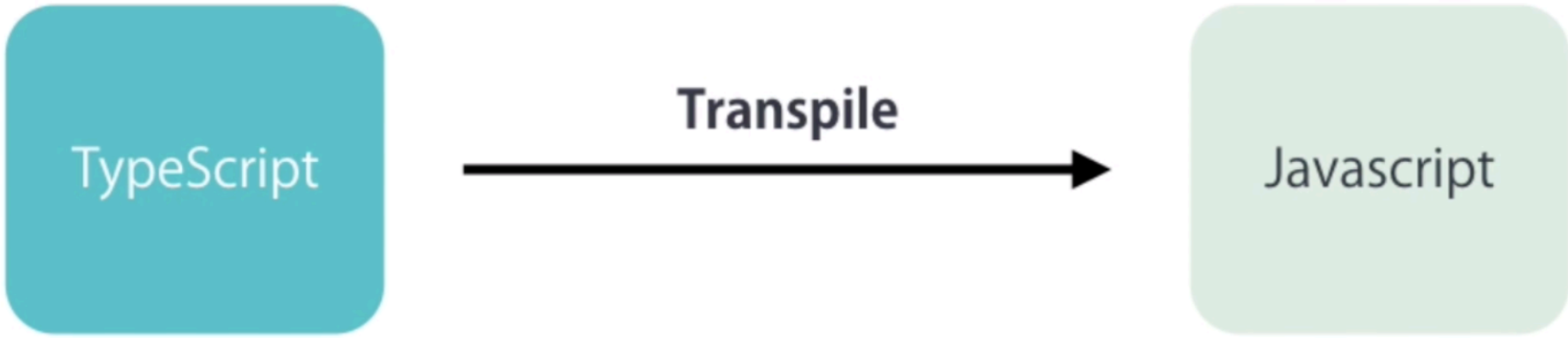
- Typescript est un sur-ensemble (superset) de Javascript.



- Ne peut pas être intégré directement à une page web.
- Permet de tirer partie des fonctionnalités modernes de ES6 comme les modules, les classes, les interfaces, fonctions fléchées, etc..
- Open source.

TYPESCRIPT, C'EST QUOI ?

- Doit être transpiler en Javascript.



- Un langage transcompilé est un langage pouvant être compilé en un autre langage.
- Autrement dit, on peut parler d'un langage en surcouche d'un langage existant.
- Amène une phase de compilation aux langages interprétés.

TYPES

- Le typage en TS ne sert qu'au compilateur pour prévenir les erreurs.
- Types proposés par Typescript :
  - Boolean
  - Number
  - String
  - Array
  - Enum
  - Any

TYPE ASSERTIONS

- Typescript peut avoir des confusions concernant les types de variables.
- Ceci peut poser des problèmes pour le développeur.

```
let msg = 'abc';

let test = msg.
```

endsWith (method) String.endsWith
fixed
fontcolor
fontsize
includes
indexOf
italics
lastIndexOf
length
link
localeCompare
match

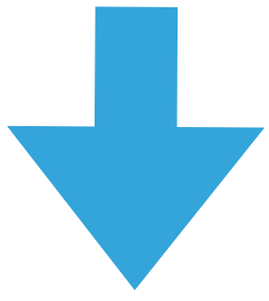
```
let msg;
msg = 'abc';

let test = msg.
```

IntelliSense ne propose rien car il ne reconnaît pas msg comme un String

ARROW FUNCTIONS

- Les fonctions fléchées (ES6, TypeScript) sont une syntaxe simplifiée pour les expressions de fonction. Elles éliminent le mot-clé **function** et ajoutent une flèche **=>** entre les arguments et le corps de la fonction.

```
let log = function(message) {
    console.log(message);
}
```



```
let doLog = (message) => {
    console.log(message);
}
```

CLASSES

- JavaScript s'appuie sur les fonctions pour la création de composant
- JavaScript s'appuie sur les prototypes pour spécialiser ces composants
- JavaScript ne propose pas de notion de classe.

- TypeScript propose d'utiliser les notions de programmation orientée objet tel que :
  - Le constructeur
  - L'encapsulation
  - Les accesseurs (es5)
  - L'héritage
  - Les classes et interfaces

```
class Personne{
    name : string;

    constructor(a:string){
        this.name = a;
    }

    direbonjour(){
        console.log("bonjour je m'appelle "+this.name);
    }

}

let a:Personne;
a= new Personne("Robert");
a.direbonjour();
```

CONSTRUCTOR

- Comme en Java, le constructeur est responsable de la création des objets à partir d'une classe.
- Contrairement au Java, une classe ne peut pas implémenter qu'un seul constructeur. D'où la possibilité de déclarer des arguments optionnels.

```
class Point {
    x: number;
    y: number;

    constructor(x?: number, y?: number) {
        this.x = x;
        this.y = y;
    }

    draw() {
        console.log('X: ' + this.x + ', Y: ' + this.y);
    }
}

let point = new Point();
point.draw();
```

ACCESSEURS

- Comme en Java ou en C#, le principe d'encapsulation adopté par Typescript nous propose 3 modes d'accès:
  - Public
  - Private
  - Protected
- Contrairement à Java, le mode par défaut est Public.

PROPRIÉTÉS

- Pour accéder en lecture ou en écriture à un attribut de classe, nous avons l'habitude d'utiliser des accesseurs (getters + setters).

```
class Point {
    constructor(private x?: number, private y?: number) {
    }

    draw() {
        console.log('X: ' + this.x + ', Y: ' + this.y);
    }

    getX() {
        return this.x;
    }

    setX(value) {

    }
}
```

```
class Point {
    constructor(private x?: number, private y?: number) {
    }

    draw() {
        console.log('X: ' + this.x + ', Y: ' + this.y);
    }

    get X() {
        return this.x;
    }

    set X(value) {
        if (value < 0)
            throw new Error('value cannot be less than 0.');

        this.x = value;
    }
```

Testons ces bouts de code...

INTERFACES

- Les Interfaces nous permettent d'établir des contraintes par rapport aux objets et aux classes qu'on implémente.
- Peuvent être utilisés pour définir des types bien définies sans passer par les classes.
- Il est important de noter que les interfaces ne sont pas compilés en Javascript.
- Ils servent juste à la validation de nos variables ou autre objet par le compilateur de Typescript.

GENERICS

- Generics en Typescript donne la possibilité de passer plusieurs types à un composant, en ajoutant une couche supplémentaire d'abstraction et de réutilisation à notre code.

- Les génériques peuvent être appliqués aux fonctions, interfaces et classes dans Typescript.

```
function identity(arg: number): number {
    return arg;
}
```

```
function identity<T>(arg: T): T {
    return arg;
}
```

MODULES

- Les modules servent à partager du code entre divers fichiers sources. Utilisation des mots-clés **Export** et **Import**.
- 2 types d'export : Export nommée et Export par défaut.
- On ne peut avoir qu'une seul Export par défaut par fichier.
- Possibilité d'utiliser des alias.

```
export default class Etudiant {
    id : number;
    nom : string;
    prenom : string;
```

```
export function testFct() {
    console.log("This is a text");
}
```

```
import add, { multiply as m } from './math';
import Etudiant, { testFct } from './component';
```