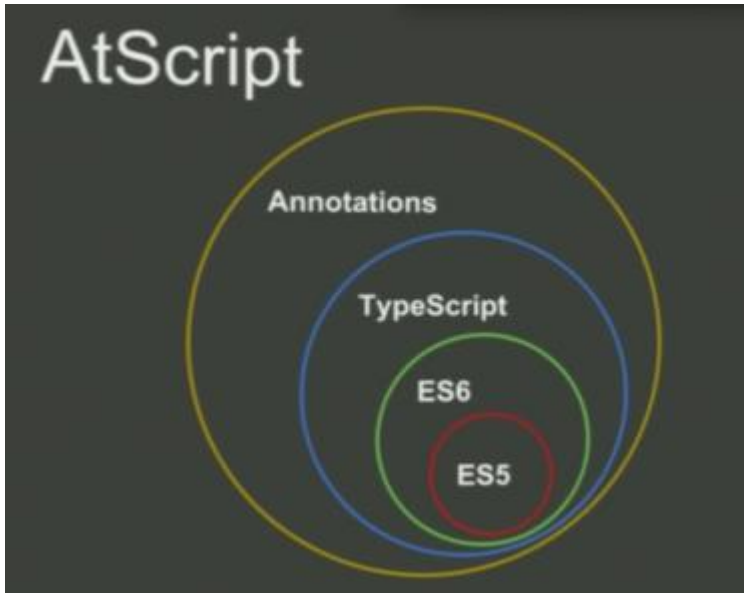
TypeScript

Animé par Mazen Gharbi

C'est quoi?

▹ TypeScript est un langage gratuit et open-source développé et maintenu par Microsoft depuis Octobre 2012.
▹ TypeScript est une surcouche d'ECMAScript permettant l'ajout optionnel de typage statique.

Contrairement à l'ECMAScript, TypeScript n'est pas un standard et aucun support n'est prévu sur les navigateurs, d'autant plus que contrairement à l'ECMAScript, TypeScript est typé statiquement et nécessitera donc toujours une transpilation.



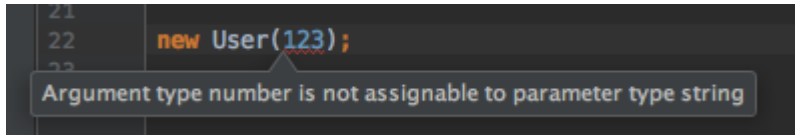
Exemple

```
class User {

    constructor(firstName: string) {
        this._firstName = firstName;
    }

}

// error TS2339: Property '_firstName' does not exist on type 'User'.

class User {

    _firstName: string;

    constructor(firstName: string) {
        this._firstName = firstName;
    }

}

new User(123);

// error TS2345: Argument of type 'number' is not assignable to parameter of type 'string'.
```

Les avantages

▷ Le typage statique impose plus de rigueur et force le respect des conventions.

▷ Le typage statique fournit une aide précieuse à l'IDE.



Typings

Tel un fichier ".h" en C/C++, les fichiers de déclaration TypeScript ".d.ts" permettent de définir des interfaces TypeScript pour les librairies ES.

App.ts

```ts
import {Utils} from './utils';

let utils = new Utils();

utils.hello(123);
```

App.js

```js
exports.Utils = class Utils {
    hello(message) {
        console.log(message);
    }
};
```

```ts
export declare class Utils {
    hello(message: string): void;
}
```

App.d.ts

Déclaration de variables

```
let bool: boolean;

bool = 'test'; // error TS2322: Type 'string' is not assignable to type 'boolean'.

let num: number = 0;

let list: string[] = ['hello'];
list.push(123); // error TS2345: Argument of type 'number' is not assignable to parameter of type 'string'.

enum Status {OK, NotOK}

let appStatus:Status = Status.OK;
appStatus = 'OK'; // error TS2322: Type 'string' is not assignable to type 'Status'.

let iDontKnow: any = 'test';

let logUserName = function logUserName(args: {user: {firstName: string, lastName: string}}) {
    console.log(`${args.user.firstName} ${args.user.lastName}`);
};

let user = {firstName: 'Foo', lastName: 'BAR'};

logUserName({user: user}); // Foo BAR
```

Paramètres optionnels

```
class User {
    firstName;
    lastName;

    constructor({firstName, lastName='default'}: {firstName: string, lastName?: string})
    {
        this.firstName = firstName;
        this.lastName = lastName;
    }

}

let user = new User({firstName: 'Foo'});
console.log(user.firstName); // ???      => Foo
console.log(user.lastName); // ???       => default
```

Classes – Custom types

```
class User {
    firstName: string = 'John’;
    lastName: string = 'Doe';
}

class UserSerializer {
    serialize({user}: {user: User}) {
        return `${user.firstName} ${user.lastName}`;
    }
}

let serializer = new UserSerializer();
console.log(serializer.serialize({user: new User()})); // John Doe
```

Interfaces

```
interface IUser {
    name(): string
}

class User implements IUser {
    name(): string {
        return 'Foo BAR';
    }
}

let printUser = ({user}: {user: IUser}) => console.log(user.name());

printUser({user: new User()});   //  Foo  BAR
```

Une interface peut également héritée d'une autre interface, voir ici

Generics

```
class User {
    firstName: string = 'Toto';
    lastName: string = 'Tata';
}

interface ISerializer<T> {
    serialize(T): string;
}

class UserSerializer implements ISerializer<User> {
    serialize({user}: {user: User}) {
        return `${user.firstName} ${user.lastName}`;
    }
}

let serializer:ISerializer<User> = new UserSerializer();
console.log(serializer.serialize({user: new User()}));      //  Toto  Tata
```

Annotations en AtScript

```
function f(param1) {
    return function (target, propertyKey, descriptor) {
        console.log("f(): called with param " + param1);
    }
}

class C {
    @f(1)
    method() {
        console.log("Contenu de ma méthode");
    }
}
```

Modules

▷ Depuis ES6, il est possible d'importer des fonctionnalités d'un autre fichiers javascript à partir d'un fichier javascript :

personne.class.ts

```
export class Personne {
    firstname: string;
}
```

---

main.ts

```
import { Personne } from './personne.class';

let pers = new Personne();
```

Questions