

# Algorithmique & Poo avec



TypeScript

# Plan

.C'est quoi l'algorithmique ?

.C'est quoi la POO ?

.Modélisation Uml

.Pourquoi TypeScript ?

.Dev Tools

.Hello World

.Les Variables

.Les types de base

.Liste/Tableau & Collection

.Les autres type

.Les Fonctions

.Les Boucles

.Classe

.Héritage

.Visibilité & Accesseur

.Méthode & attribut static

.Particularités des accesseurs (ES5)

.Interface (classique)

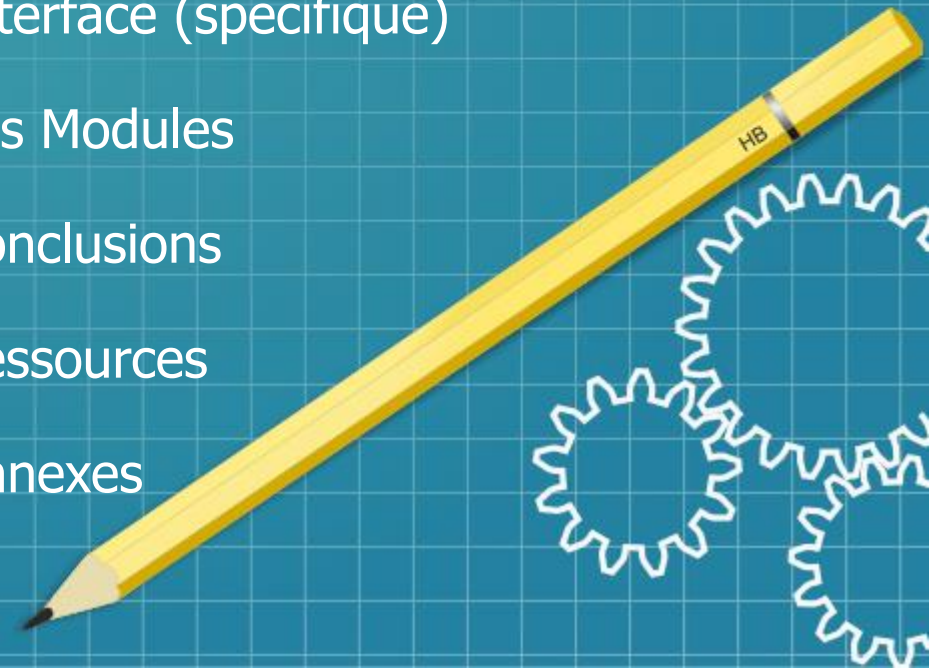
.Interface (spécifique)

.Les Modules

.Conclusions

.Ressources

.Annexes







# C'est quoi l'algorithmique ?

• L'algorithmique est l'étude et la production de règles et techniques qui sont impliquées dans la définition et la conception d'algorithmes.

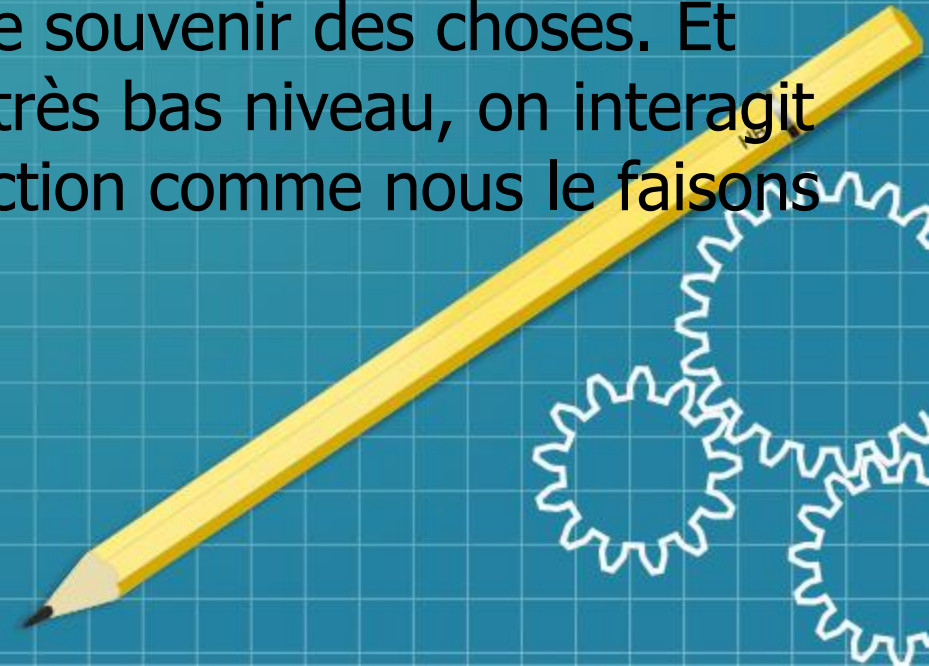
• Un algorithme est une suite finie et non ambiguë d'opérations ou d'instructions permettant de résoudre une classe de problèmes





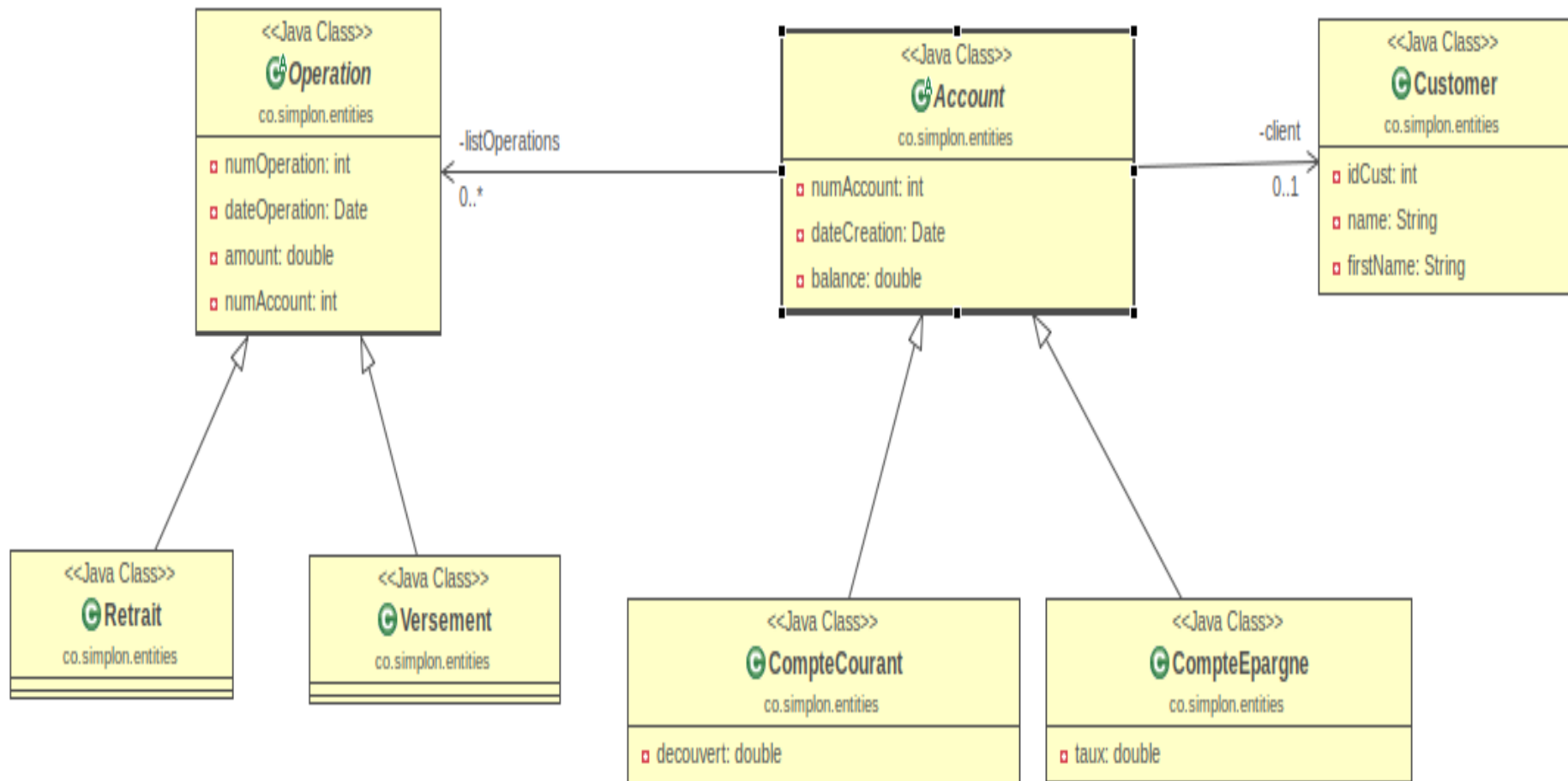
# C'est quoi la POO ?

.Steve Jobs explique que « les objets sont comme les gens. Ce sont des choses qui vivent, respirent, ont des connaissances intrinsèques sur comment faire les choses et ont une mémoire à l'intérieur d'eux qui leur permet de se souvenir des choses. Et plutôt que d'interagir avec eux à un très bas niveau, on interagit avec eux à très haut niveau d'abstraction comme nous le faisons présentement »





# Modélisation Uml





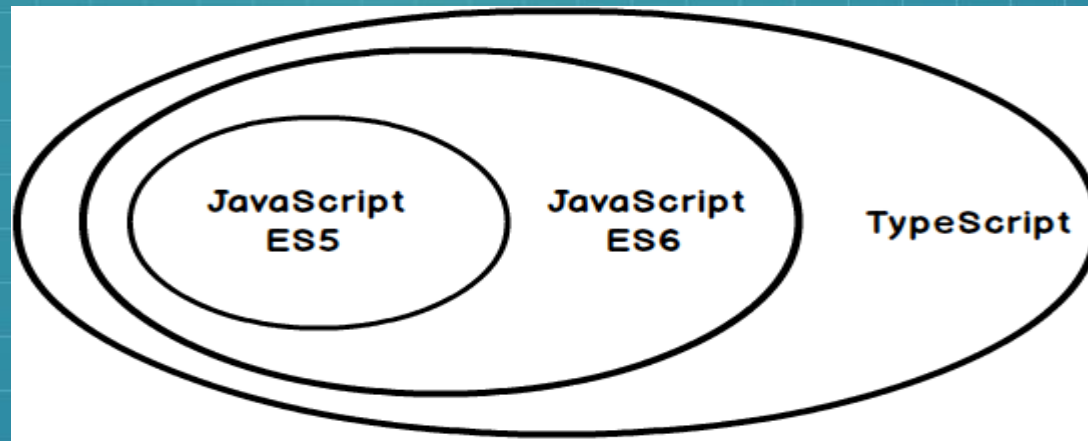
# Pourquoi TypeScript ?

- JavaScript c'est bien mais n'étant pas structuré, pour les grandes applis, cela devient fastidieux !



# C'est quoi TypeScript ?

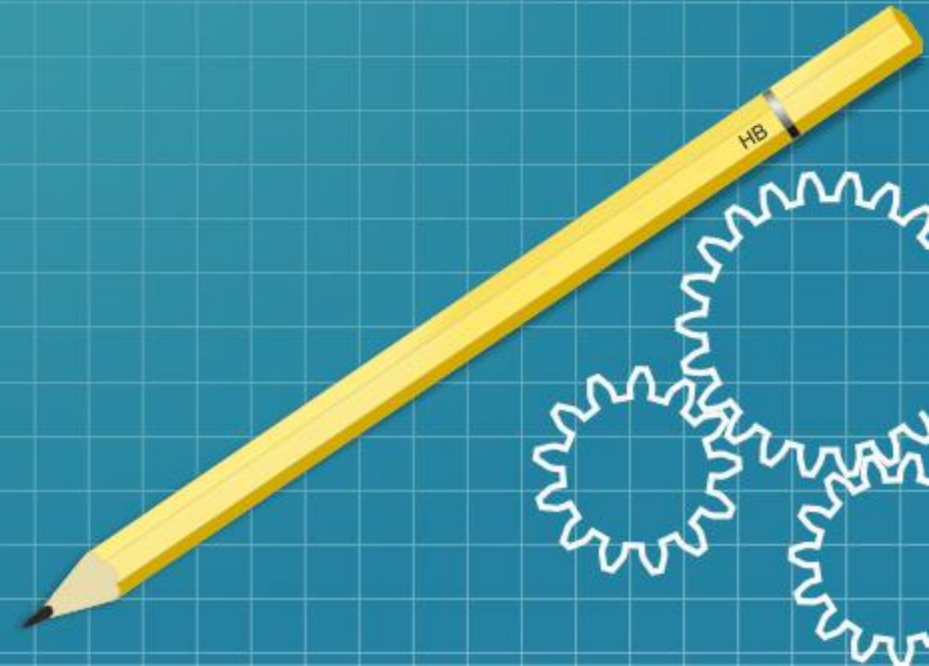
- TypeScript est un langage de prog open source développé par un dev de Microsoft en 2012
- étant un sur ensemble de JS, tout code JS valide l'est pour TS



EcmaScript :

- Ensemble de normes concernant les langages de programmation de type script standardisés par Ecma International.
- Standard donc dont les spécifications sont mises en œuvre dans différents langages de script.

TS offre de nouveaux concepts  
permettant de gérer de grand code







classe

héritage

interface

attribut & encapsulation

→ indispensable pour faire de la POO

typage statique avec ou sans verif

correction d'erreur à la compilation

→ code mieux structuré et plus sécurisé

code TS facile à comprendre

communauté de soutien importante à cause de microsoft

bonne doc & flux constant de dev

→ utilisé sur plusieurs Framework front dont ANGULAR



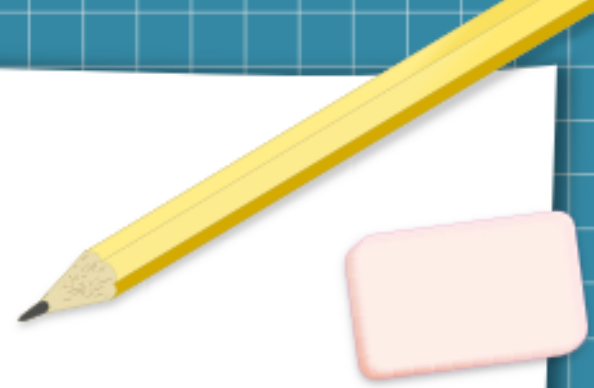
# Dev Tools

.Utilisation du Transpiler ou compilateur a pour rôle de générer un fichier index.js à partir d'index.ts

.Donc il faut l'installer (pré-requis node & npm) : `npm install -g typescript`

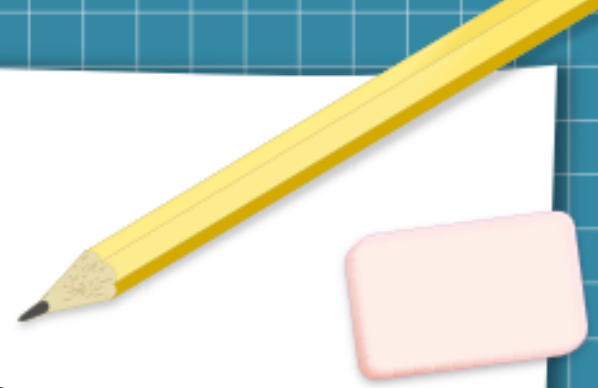


# Hello World !



- en ligne de commande, créer un rep TS dans IdeaProjects
- ouvrir IntelliJ sur ce rep
- à partir d'intelliJ, ajouter à la racine du rep un fichier hello.ts
- ajouter un petit message : `console.log('Hello World');` et(ou) `alert('Hello World');`
- compiler via le terminal de l'ide : `tsc hello.ts` => est-ce qu'un fichier hello.js a été généré ?
- ajouter le script dans une page html pour vérif

# Les Variables



.const|let|var nomVariable: type = valeur;

.let a:string = 'bonjour';

.a = 4; erreur

.a = "au revoir";

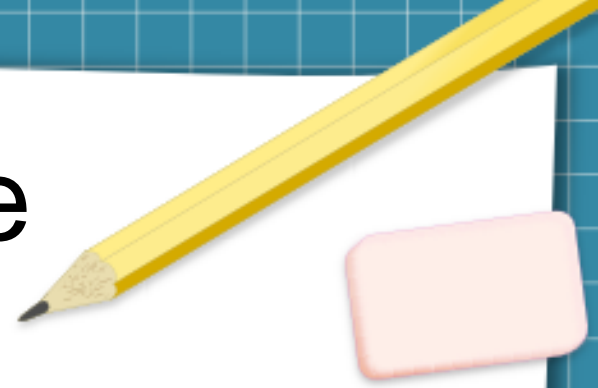
.Question :

.let b = 'hello';

.b = 3; erreur ou pas ?



# Les Types de base



- let entier: number = 1;
- let pi: number = 3.14;
- let bool: boolean = true;
- let nom: string = "je suis une chaine de caractère";
- let objet: Object = {"name": "topPhone"};

# Liste/tableau et collection



```
•let list: number[] = [1, 2, 3];
```

```
•let list2: Array<number> = [1, 2, 3];
```

```
•let values: (string | number)[] = ['Apple', 2, 'Orange', 3, 4, 'Banana'];
```

```
•let values2: Array<string | number> = ['Apple', 2, 'Orange', 3, 4, 'Banana'];
```

```
•let collection1: Object [] = [{"name":"topPhone"}, {"name":"bigPhone"}];
```

```
•let collection2: Array<Object> =  
[{"name":"topPhone"}, {"name":"bigPhone"}];
```



# les autres type : enum, any, void



```
.enum feux { rouge,    //0  
.  
.  
.  
orange}; //2  
  
.let feu = feux.rouge;
```

```
.if( feu == 0 )    console.log('arrêt immédiat');  
.else if(feux == 1) console.log('vous pouvez passer');  
.else              console.log("vous devez vous arrêter");
```

```
.let variable: any;  
  
.variable = 'any string';  
.console.log(variable);  
  
.variable = 450;  
.console.log(variable);
```

.void équivaut à rien mais il faut introduire la notion de fonction pour comprendre

# Fonctions & param optionnel



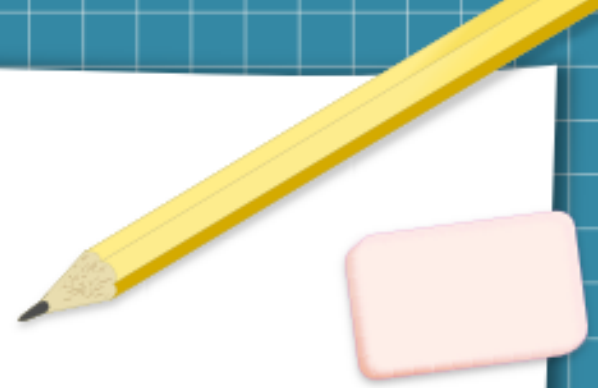
```
.function helloName(name:string): string {  
  return "hello " + name;  
.  
}
```

```
.function nameHello(name:string) : void {  
  console.log("hello "+ name);  
.  
}
```

```
.function helloSurname(name:string,surname?:string) :void {  
  let local:string="";  
  .if(surname)    local = " " + surname;  
  .console.log("hello " + name + local);  
.  
}
```

```
.-> console.log(helloName("trump"));  
  
nameHello('chirac');  
  
.helloSurname("sarko","nico");
```

# Les boucles



```
for (let i = 0; i < 3; i++) {  
  console.log ("top " + i);  
}
```

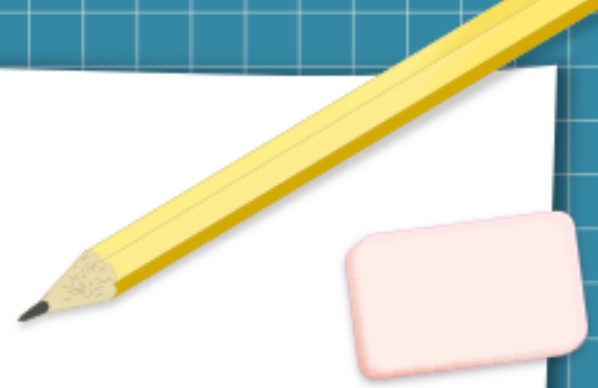
```
let num = [1, 2, 3];  
num.forEach(function (value) {  
  console.log("hello ---> " + value);  
});
```

```
let sun = ["un", "deux", "trois", "soleil"];  
for(let play of sun)  
  console.log(play);
```

```
let n:number = 5;  
let factorial:number = 1;  
while(n >= 1) {  
  factorial = factorial * n;  
  n--;  
}  
console.log("The factorial is "+factorial);
```



# Classe



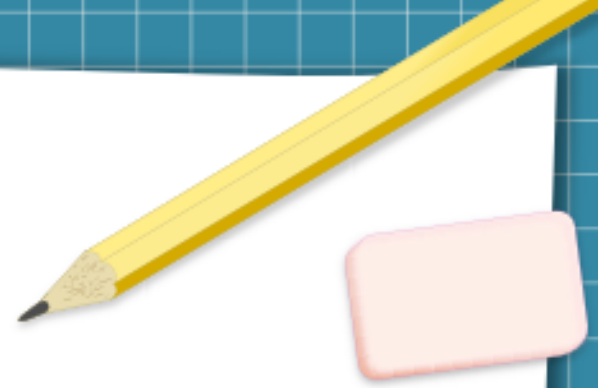
```
class Phone {  
    private name:string;  
    private brand:string;  
  
    constructor(name, brand){  
        this.name = name;  
        this.brand = brand;  
    }  
  
    public toString() : string{  
        return "name : " + this.name + " / brand : " + this.brand;  
    }  
}
```

---> Regarder le code javascript généré ici

```
let phone = new Phone("3310","nokia");
```

```
console.log(phone.toString());
```

# Héritage



```
class SmartPhone extends Phone {  
    private os:string;  
  
    constructor(name, brand, os){  
        super(name,brand);  
        this.os = os;  
    }  
    public toString():string{  
        return super.toString() + " / os : " + this.os;  
    }  
}  
  
let smartPhone = new SmartPhone("s9","samsung","android");  
console.log(smartPhone.toString());
```

# Visibilité et accesseurs

```
class Tablet {  
    private name:string;    private brand:string;  
    constructor(name, brand){    this.name = name;    this.brand = brand; }  
    public getName()          : string {    return this.name;    }  
    public getBrand()         : string {    return this.brand;    }  
    public setName(name:string) : void    {    this.name = name;    }  
    public setBrand(brand:string) : void    {    this.brand = brand;    }  
    public display() : void {  
        console.log('nom de ma tablette : ' + this.getName() + " / " + "marque : " + this.getBrand());  
        //est-ce que je peux changer qqchose ici ?  
    }  
}
```

```
let tablet = new Tablet("Ipad", "Apple");  
console.log('nom de ma tablette : ' + tablet.getName() + " / " + "marque : " + tablet.getBrand());  
  
tablet.setName("GalaxyTab");  
tablet.setBrand("Android");  
tablet.display();
```



# Méthodes & attributs static

.Par ex, on souhaite un compteur d'objets :

```
let ipad = new Tablet("Ipad","Apple");
```

```
let galaxyTab = new Tablet("GalaxyTab","Android");
```

```
let unknown = new Tablet();
```

```
ipad.display(); // 1 objet tablet
```

```
galaxyTab.display(); // 2 objets
```

```
unknown.display(); // 3 ? mais il y a un autre pb ici !
```



# SOLUTION

```
class Tablet {
    private static compteur:number = 0;
    public static getCompteur() : number { return this.compteur };

    private name:string;    private brand:string;

    constructor(name, brand){    this.name = name;    this.brand = brand; Tablet.compteur++;}

    public getName()          : string {    return this.name;    }
    public getBrand()         : string {    return this.brand;    }
    public setName(name:string) : void    {    this.name = name;    }
    public setBrand(brand:string) : void    {    this.brand = brand;    }
    public display() : void {
        console.log('nom tablette : ' + this.getName() + " / " + " marque : " + this.getBrand());
    }
}

console.log("la valeur de compteur au tout début : " + Tablet.getCompteur());
let ipad = new Tablet("Ipad","Apple");
let galaxyTab = new Tablet("GalaxyTab","Android");
let unknown = new Tablet("?", "?");
ipad.display();
galaxyTab.display();
unknown.display();
console.log("la valeur de compteur après les 3 instances : " + Tablet.getCompteur());
```

# Particularités des accesseurs uniquement depuis ES5

```
class Tablet {  
  private _name:string;  
  private _brand:string;  
  constructor(name, brand){      this._name = name;          this._brand = brand;}  
  
  public get name()                : string {          return this._name;  }  
  public get brand()               : string {          return this._brand; }  
}  
  
public set name(name:string) {    this._name = name;    }  
public set brand(brand:string) {  this._brand = brand; }  
}
```

```
let ipad = new Tablet("Ipad","Apple");  
let galaxyTab = new Tablet("GalaxyTab","Android");  
let unknown = new Tablet("?", "?");
```

```
console.log(ipad.name + ' / ' + ipad.brand);  
console.log(galaxyTab.name + " / " + galaxyTab.brand);
```

```
unknown.name = "Pixel 3";  
unknown.brand = "Google";  
console.log(unknown.name + ' / ' + unknown.brand);
```

NB : par défaut on génère du JS ES3, si on veut spécifier de l'ES5 comme ici, il faut ajouter -target 'es5'



# Les Interfaces (classique)



```
interface EnFormation{  
    times() : void;  
    works() : void;  
}
```

```
interface EnEntreprise{  
    assignments() : void;  
}
```

```
class Stagiaire implements EnFormation, EnEntreprise{  
    private login : string; private password : string;  
    constructor(login,password){  
        this.login = login;      this.password = password;  
    }  
    times() : void {  
        console.log(this.login + ", pendant la formation, commence à 9h et fini à 17h avec une  
entre 12h30 et 13h30");  
    }  
    works() : void {  
        console.log(this.login + ", durant la formation, doit réaliser 2 projets pédagogiques, un  
professionnel...");  
    }  
    assignments() : void {  
        console.log(this.login + ", en entreprise, doit réaliser 1 application web complète front +  
back en 2 mois");  
    }  
}
```

```
let stagiaire = new Stagiaire("Neo","123");  
console.log(stagiaire.times());  
console.log(stagiaire.works());  
console.log(stagiaire.assignements());
```

# Les Interfaces (cas particulier)

```
interface Promo {  
    referentiel : string;  
    nbStagiaire : number;  
    group : (nb:number) => string; //renvoi le nombre de stagiaire par équipe projet  
}
```

```
class CentreFormation {  
    public name:string;  
    public listePromos : Array<Promo>;  
  
    constructor(name:string){  
        this.name = name;  
    }  
}
```

```
let simplon = new CentreFormation("Simplon");  
simplon.listePromos =  
[ { referentiel : "dev web" , nbStagiaire : 15 , group : function(nb)
```

```
return "nb stag/gp : " + this.nbStagiaire/nb; }},  
  { referentiel : "ref dig" , nbStagiaire : 21 , group : function(nb)  
    { return "Team : " + this.nbStagiaire/nb; } }  
];
```

```
simplon.listePromos.push({referentiel : "Telecom" , nbStagiaire : 12 , group : function(r  
return "pas de groupe ici"; }));
```

```
for(let promo of simplon.listePromos)  
    console.log(promo.referentiel + ' / ' + promo.nbStagiaire + ' / ' + promo.group(3));
```

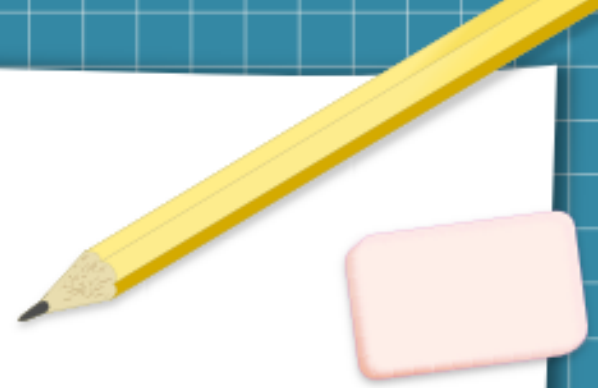
dev web / 15 / nb stag/gp : 5

ref dig / 21 / team : 7

Telecom / 12 / pas de groupe ici



# Les Modules



**<shape.ts>**

```
export default class Shape {  
  private x:number;  
  private y:number;  
  
  constructor(x,y){  
    this.x = x;  
    this.y = y;  
  }  
  
  display(){  
    console.log("Shape [" + this.x + "," + this.y + "]");  
  }  
}
```

**<index.ts>**

```
import Shape from './shape';
```

```
let shape = new Shape(2,3);  
shape.display();
```

NB : Afin que ce mécanisme fonctionne, il faudra utiliser un bundler (à installer) tel que FuseBox, WebPack...



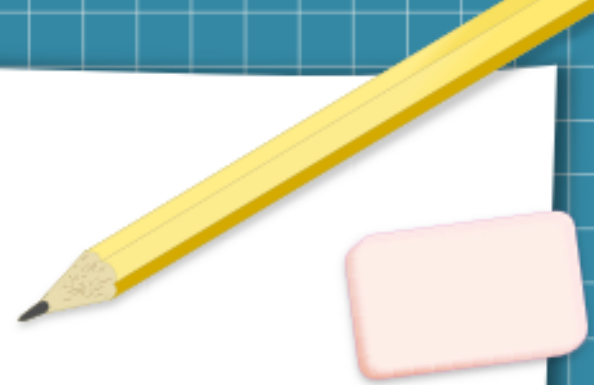
# Conclusions

.On a pas tout vu, reste namespace, generics, decorators... on y reviendra avec Angular !

.Retenons les principaux points fort :

- →Poo
- →Code mieux structuré, + secure
- →Utilisé par Angular, React

# Ressources



.Site de ref de typescript :

<https://www.typescriptlang.org/index.html>

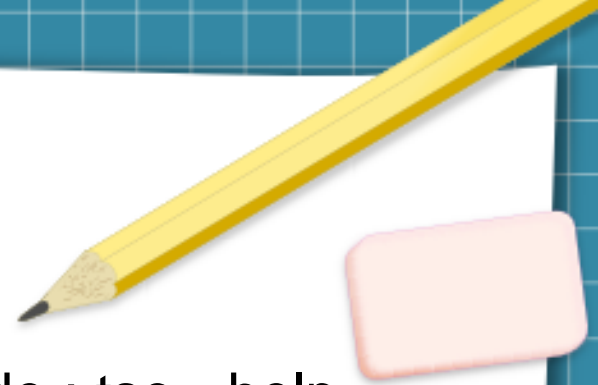
.Pour dev en ts, on peut utiliser le playground ici

<https://www.typescriptlang.org/play/index.html>

. ou <https://repl.it> ...

.Tutoriel : <https://www.grafikart.fr/tutoriels/typescript-781>

# Annexes



- Pour connaître l'ensemble des options de la commande : `tsc --help`
- `tsc -w` permet de watcher le fichier, dès qu'il y a une modification, recompile !
- `tsc -w --pretty` pour avoir des erreurs explicites sur la console
- `tsc -w --pretty hello.ts` pour continuer à compiler ce fichier
- `tsc -w --pretty hello.ts --target 'es5'` (génère de l'ES5)
- → crée un fichier `tsconfig.json` pour sauvegarder la config :

```
{  
  "compilerOptions": {  
    "target": "es5",  
    "pretty": true,  
    "outDir": "./build"  
  },  
  "files": ["hello.ts"],  
  "exclude": ["node_modules"]  
}
```

- `tsc --watch` pour exécuter ce fichier de config