CSS frameworks
BOOTSTRAP
FONDATION

CL & IHM

EcmaScript et TypeScript

Alexandre.Demeure@univ-grenoble-alpes.fr

2017

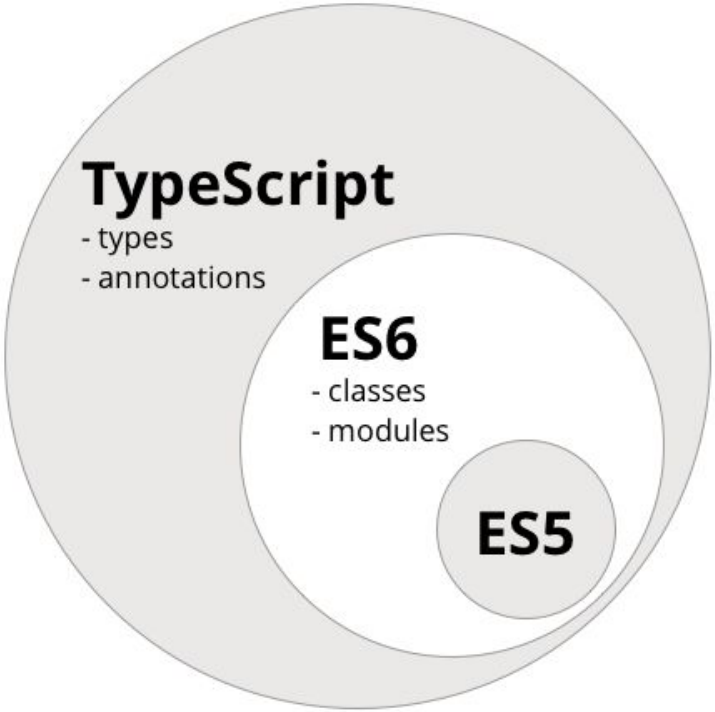
EcmaScript et TypeScript

Le langage EcmaScript

- LiveScript puis Javascript puis EcmaScript (**ES5**, **ES2015**, ES2017, ...)
- Principalemment implémenté par les **navigateur web** et **NodeJS**
- Langage interprété, faiblement typé
- Manipulation du HTML (API DOM)

Le langage TypeScript

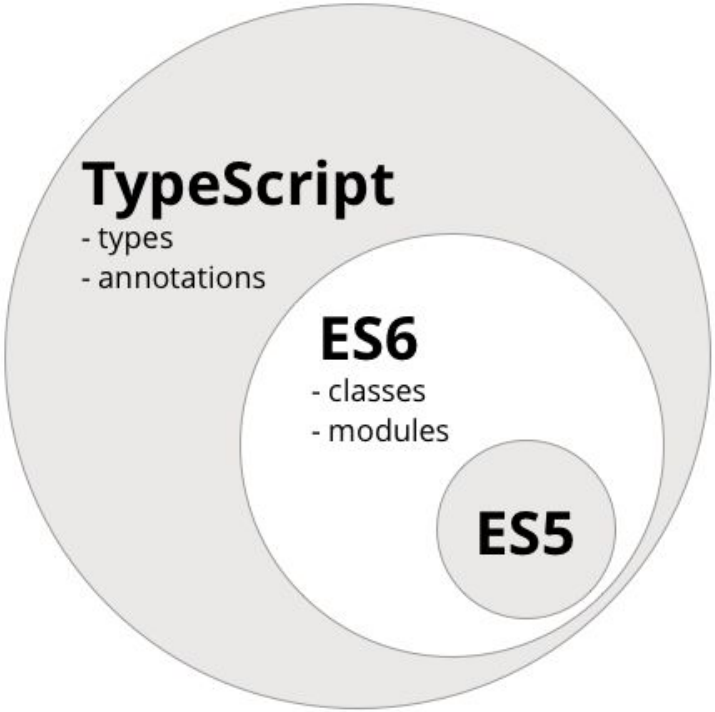
- Sur-ensemble de ES2015 (dît ES6)
- Typage => vérification statique
- Javascript qui passe à l'échelle
- Compilé vers du ES5 ou ES6
- Utilisé par plusieurs frameworks
  - Angular 2
  - IONIC
  - Aurelia, ...
- Typage des bibilothèques existantes: http://definitelytyped.org/



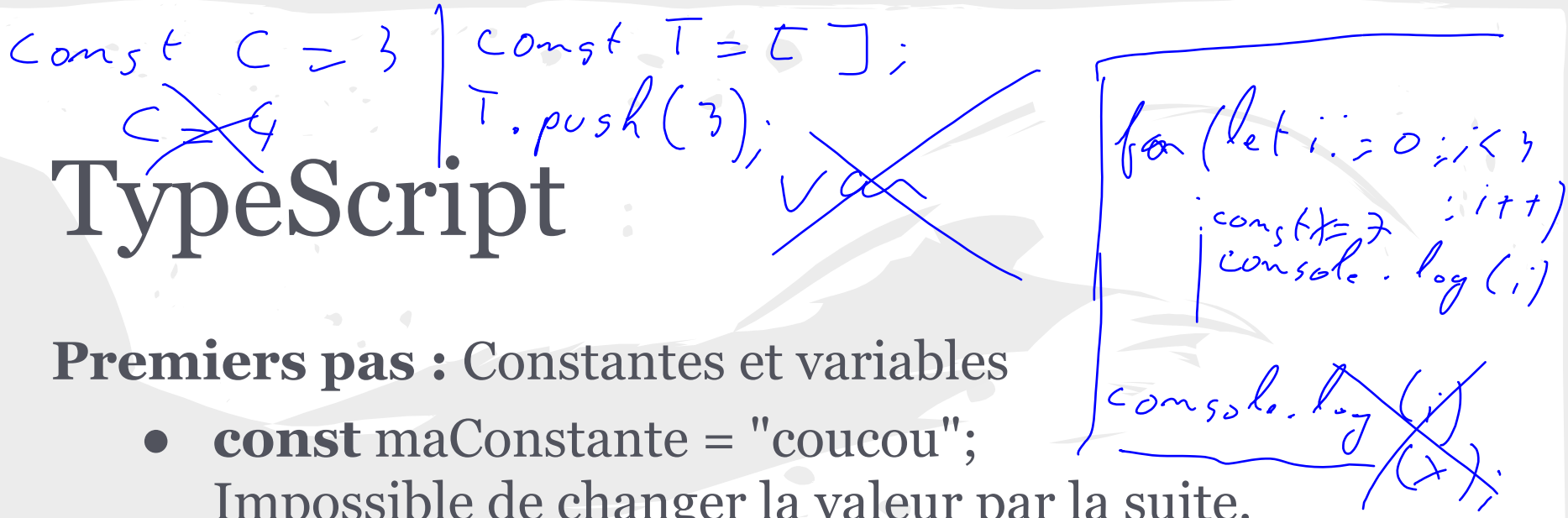
TypeScript

Pour ce cours: TypeScript

- Permet d'éviter un certain nombre d'erreurs "de bases"
- Permet de mieux aborder Angular2
- Le typage arrive dans ES2017
- Il y a des chances que TypeScript influence ES2017...
- Attention au code ES6 et ES5 trouvé sur le web !!!



Typescript
https://www.typescriptlang.org/
Mozilla Developer Network : MDN
Stack Overflow (forum)
IRC: #typescript



TypeScript

Premiers pas : Constantes et variables

- **const** maConstante = "coucou";
  Impossible de changer la valeur par la suite.
- **let** maVariable = "coucou";
  la variable peut être affectée à une nouvelle valeur du même type

Typage implicite ou explicite

- **let** monNombre = 15;
- **let** monNombre : number; monNombre = 15;
- **let** monNombre : number = 15;
- On ne peut pas changer le type d'une variable une fois qu'il est définit

TypeScript

Quelques opérateurs :

- & : ET logique
- | : OU logique
- && : ET ALORS
- || : OU ALORS
- = : Affectation
- == : égalité avec projection de type (5 == 5 && 5 == "5")
- != : inégalité avec projection de type
- === : égalité dîte stricte (de valeur et de type)
- !== : inégalité dîte stricte (5 === 5 && 5 !== "5")
- Opérateur ternaire : condition?valeurSiVrai:valeurSiFaux
  `5===x?"banco":"manqué"`
- Destructuration :
  - `[a, b] = [b, a]`
  - `[a, b, ...rest] = [0, 1, 2, 3, 4, 5, 6]`
  - `{nom:N, prénom:P} = {nom:"Bob", prénom:"Kelso", âge:56}`

TypeScript

Quelques instructions de contrôles:

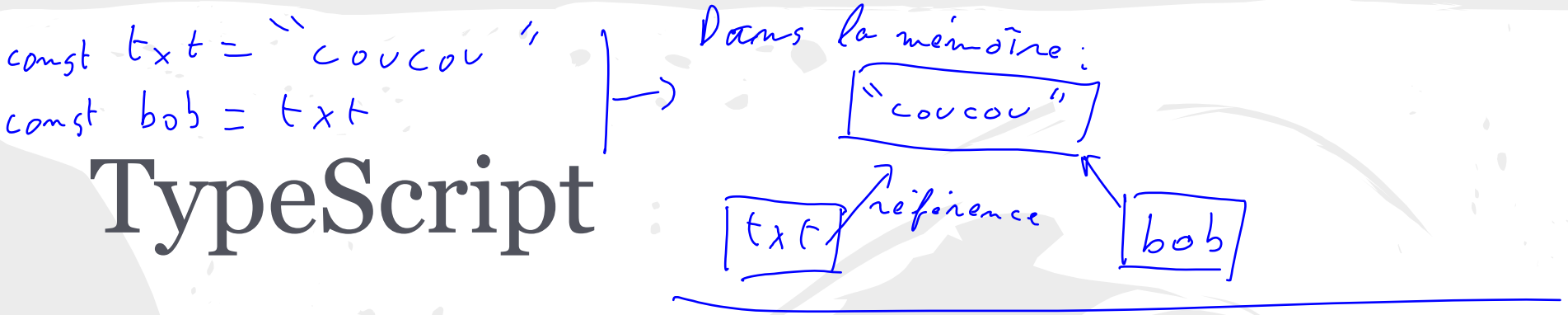
- **if** (CONDITION) {INSTRUCTIONS} **else** {INSTRUCTIONS}
- **while** (CONDITION) {INSTRUCTIONS}
- **do** {INSTRUCTIONS} **while**(CONDITION)
- **switch** (EXPRESSION) {
  case VALUE_1: INSTRUCTIONS; **break**;
  ...
  case VALUE_N: INSTRUCTIONS; **break**;
  default: INSTRUCTIONS
- **for**(INIT; CONDITION; INCR) {INSTRUCTIONS}
- **for**(**let** i **in** ITERABLE) {INSTRUCTIONS}
- **for**(**let** v **of** COLLECTION) {INSTRUCTIONS}
- **try** {INSTRUCTIONS} **catch**(ERROR) {INSTRUCTIONS}

TypeScript

Types de bases : **undefined**

Types de bases : **string** (voir sur MDN)

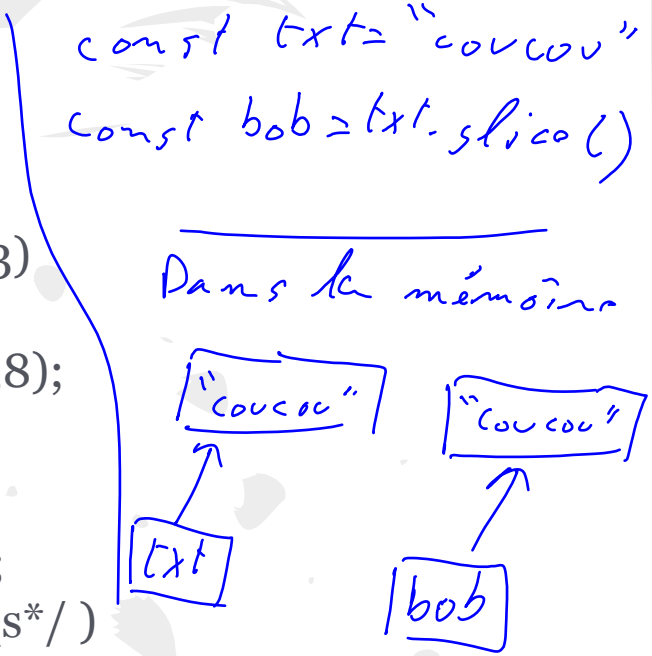
- **let** monTexte1 : **string** = "coucou on est là";
  let monTexte2 : **string** = "coucou on est là";
  let txt = "je dit" + monTexte1 + " et " + monTexte2;
- Texte sur plusieurs lignes avec accent grave
  let longTexte = `je suis un texte
  écrit sur
  plusieurs lignes et ${monTexte1} plus ${monTexte2}`;
- Comparaison avec === et !==



TypeScript

Types de bases : **string** (voir sur MDN)

- Copie : **slice**([début, [fin]])
  - **let** copieTexte = monTexte1.**slice**();
  - **let** troisPremiers = monTexte1.**slice**(0, 3)
  - **let** troisDerniers = monTexte1.**slice**(-3)
  - **let** deQuatreàHuit = monTexte1.**slice**(4,8);
- Tronçonnage : **split**
  - **let** TableauChar = monTexte1.**split**();
  - **let** TableauMots = monTexte.**split**(" ");
  - **let** TableauMots2 = monTexte1.**split**(/\s*/)
- … et bien d'autres !



TypeScript

Types de bases : **number** (voir sur MDN)

- Pas de différences entre entier et flottants
- let x : number;
- x = 30/0; x.isNaN() === true
- x = 90; x.isFinite() && x.isInteger() === true
- Objet Math
  - Nombreuses fonctions : Math.cos, Math.abs, ...
  - Nombreuses constantes : Math.PI, ...
- Conversion vers les chaînes de caractères
  - parseInt(` 127.43 c'est bien`) === 127
  - parseFloat(` 127.43 c'est bien`) === 127.43

TypeScript

Types de bases **boolean** (voir sur MDN)

- **let** b : **boolean** = true; // ou false
- Conjonctions, disjonctions et autres opérateur booléens
- Tout ce qui n'est pas faux est vrai…
- Qu'est ce qui est faux à part false ?
  - 0 et -0
  - NaN
  - null
  - undefined
  - "" // chaîne de caractère vide

TypeScript

Types de bases **enum** (voir sur MDN)

- N'existe pas dans ES6
- ```
  enum typeEnum {v0, v1, vN}
  let v : typeEnum = typeEnum.v0;
  ```
  ```

TypeScript

Types de bases **function** (voir typescriptLang et MDN)

- Un type de premier plan en ES et TS !
- Une variable ou un paramètre peut être une function…
  …comme dans les autres langages
- Une fonction renvoie une valeur
- Une fonction prend des arguments en paramètres

```
function F(x: number, y: number) : number {return x+y}
let Somme = F;
F(3,4) === Somme(3, 4)
```

- Type d'une fonction, ex :

```
let Somme : (x: number, y: number) => number = F
```

TypeScript

Types de bases **function** (voir typescriptLang et MDN)

- Paramètres optionnels : suffixe ?
  function F(x : **number**, y**?** : **number**) {
  if(y === **undefined**) {...}
  }
- Valeur par défaut d'un paramètre
  function F(x : **number**, y : **number = 0**) .....
- Notation pour le reste des paramètres : **...NOM**
  function F(x : **number**, ...reste: **number[]**) .....
- Notation fléchée :
  let F = (x: **number**, y: **number**) : **number** => x+y

TypeScript

Types de bases **Array** (tableau,voir sur MDN)

- Notation avec les crochets ou le mot clef **Array**

```
let tableauNombres : number[];
let tableauNombres : Array<number>;
```

- Initialisation

```
let tableauNombres : number[] = [];
let tableauNombres : number[] = [76, 56];
```

- Longueur d'un tableau

```
tableauNombres.length
```

- Accès aux éléments

```
tableauNombres[2]
tableauNombres[1] = 65
```

TypeScript

Types de bases **Array** (tableau,voir sur MDN)

- Quelques fonctions de bases
  - **push** : ajout d'un élément à la fin
    `tableauNombres.push( 45 )`
  - **pop** : retrait du dernier élément
    `tableauNombres.pop()`
  - **splice**(deb, nbSuppr, ...) : supprime et insère
    `let T=[0,1,2]; T.splice(1, 1, 6) // T vaut [0,6,2]`
  - **slice**([deb, [fin]]) : renvoie une copie du tableau
  - **indexOf** : renvoi l'index de l'élément cherché ou -1

splice(index : number, nbSup : number, ...elementsAInsérer : any[])

TypeScript

Types de bases **Array** (tableau,voir sur MDN)

- Fonctions un peu plus poussées
  - **forEach** : Applique une fonction sur chaque élément
    T.forEach((v,i) => console.log(i, ":", v))
  - **map** : Prend une fonction en paramètre. Applique la fonction sur chaque élément et ajoute le résultat dans le tableau renvoyé en résultat.
    T = [6,-5,7,-9]
    let T2 = T.map(v => Math.abs(v))

```
function forEach(T: any[], F: (v?: any, i?: number,
                                T?: any[]) => void))
    for (let i=0; i<T.length; i++) {
        F(T[i], i, T);
```

```
function map(T: any[]
             , F: (v?: any, i?: number, T?: any[]) => any
             ): any[] // de même taille que T
    let Tres = [];
    T.forEach( (v,i,T) => Tres.push(F(v,i,T)))
    return Tres;
```

TypeScript

Types de bases **Array** (tableau,voir sur MDN)

- Fonctions un peu plus poussées
  - **reduce** : Réduit un tableau à une seule valeur

```
let T : number[] = [6,7,-4,98,-32]
let somme = T.reduce( (acc, v) => acc+v )
```

function reduce(T : any[ ]
, F : (acc : any
, v : any
, i : number
, T : any[ ])
, acc ? : any
) : any // La valeur de l'accumulateur à la fin.

function reduce (T , F , vInit)

```
function reduce(T, F, vInit) {
    if (T.length === 0 && vInit === undefined) throw "PAS BON !!!"
    let index = (vInit === undefined) ? 1 : 0;
    let acc   = (vInit === undefined) ? T[0] : vInit;
    for(let i=index; i<T.length; i++) {
        acc = F(acc, T[i], i, T)
    }
    return acc;
}
```

TypeScript

Types de bases **Array** (tableau,voir sur MDN)

- Fonctions un peu plus poussées
  - **filter** : Prend une fonction renvoyant un booléen en paramètre. Renvoi le tableau contenant les éléments pour lesquels la fonction à répondue vrai.

```
let T : number[] = [6,8,-5,8,-43]
let Positifs : number[] = T.filter( v => v>0 )
```

function filter(T : any[], f : (e : any) => boolean) : any[]

function filter<A>(T : A[], f : (e : A) => boolean) : A[]
ou
function filter<bob>(T : bob[], f(e : bob) => boolean) : bob[]

```
f filter(T, fct) {
    const result = [];
    for (let i=0; i<T.length; i++) {
        const e = T[i];
        if ( fct(e) ) {
            result.push(e);
        }
    }
    return result;
}
```

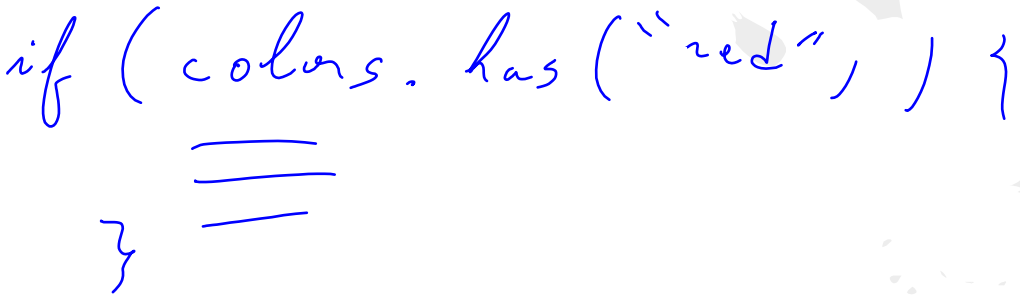
Map avec des types génériques

function map<T_1, T_2>(tab : T_1[ ], f : (e : T_1) => T_2) : T_2[ ]

TypeScript

Types de bases **Map** (voir sur MDN)

- Tableau associatif
- Associe des instances d'un type T1 à des instances d'un type T2 (avec T1 qui peut être égal à T2)
- Utilisation de la notion de générique…
- Exemple :
  - ```
    let colors = new Map<string, string>()
    colors.set("red", "FF0000")
    let red = colors.get("red")
    ```


    ```

TypeScript

Exercices : Tableau de notes

- Calcul des notes moyenne, min, max
- Changer la notation de /20 à /10
- Sélectionner les notes entre min et max

TypeScript

Types de bases **Array** (tableau,voir sur MDN)

- Fonctions un peu plus poussées
  - Exercice : Ecrire map avec foreach

TypeScript

Types de bases **Object** (voir sur MDN)

- Un objet est une structure de données
- Un objet est constitué d'attributs
  - Un attribut est une variable liée à l'objet qui le contient
  - Un attribut peut avoir n'importe quel type
  - Les attributs de types fonction sont appelés méthodes
- Déclaration d'un objet

```
let médecinChef = {
    nom           : "Bob",
    prénom        : "Kelso",
    dateNaissance : new Date(1960, 8, 17),
    nomComplet    : function() {
        return `${this.nom} ${this.prénom}, né le ${this.dateNaissance}`
    }
}
```

TypeScript

Types de bases **Object** (voir sur MDN) : Les classes

- On peut définir des classes d'objets

```
class Personne {
     nom           : string;
     prénom        : string;
     dateNaissance : Date;
     nomComplet() {
          return `${this.nom} ${this.prénom}, né le ${this.dateNaissance}`
     }
     constructor(nom: string, prénom: string, dn: Date) {
          this.nom           = nom;
          this.prénom        = prénom;
          this.dateNaissance = dn;
     }
}
let médecinChef = new Personne("Bob", "Kelso", new Date(1960, 8, 17) );
```

TypeScript

Types de bases **Object** (voir sur MDN) : Les interfaces

- Une interface définit la signature que doit avoir un objet
- Une interface peut servir de type
- Une interface ne peut pas être instanciée

```
interface Surface2D {
    Aire      : () => number;
    Périmètre : () => number;
}
```

≡

```
interface Surface2D {
    Aire() : number;
    Périmètre() : number;
}
```

```
class Rectangle implements Surface2D {
    largeur       : number;
    Hauteur       : number;
    constructor   (L: number, H:number) {this.largeur=L; this.hauteur=H;}
    Aire          () {return this.largeur*this.hauteur}
    Périmètre     () {return 2*(this.largeur+this.hauteur)}
}

let surface : Surface2D = new Rectangle()
```

TypeScript

Types de bases **Object** (voir sur MDN)

```
interface Surface2D {
    Aire      : () => number;
    Périmètre : () => number;
}

class Rectangle implements Surface2D {
    largeur       : number;
    Hauteur       : number;
    constructor   (L: number, H:number) {this.largeur=L; this.hauteur=H;}
    Aire          () {return this.largeur*this.hauteur}
    Périmètre     () {return 2*(this.largeur+this.hauteur)}
}

class Carre extends Rectangle {
    constructor   (côté: number) {super(côté, côté)}
}
```

TypeScript

Types de bases **Object** (voir sur MDN)

```
interface maStructureDeDonnées {
    x         : number;
    y         : number;
    rayon?    : number;
}

let  donnée_1 : maStructureDeDonnées = {x: 32, y: 64},
     Donnée_2 : maStructureDeDonnées = {x: 16, y: 96, rayon: 128};
```

TypeScript

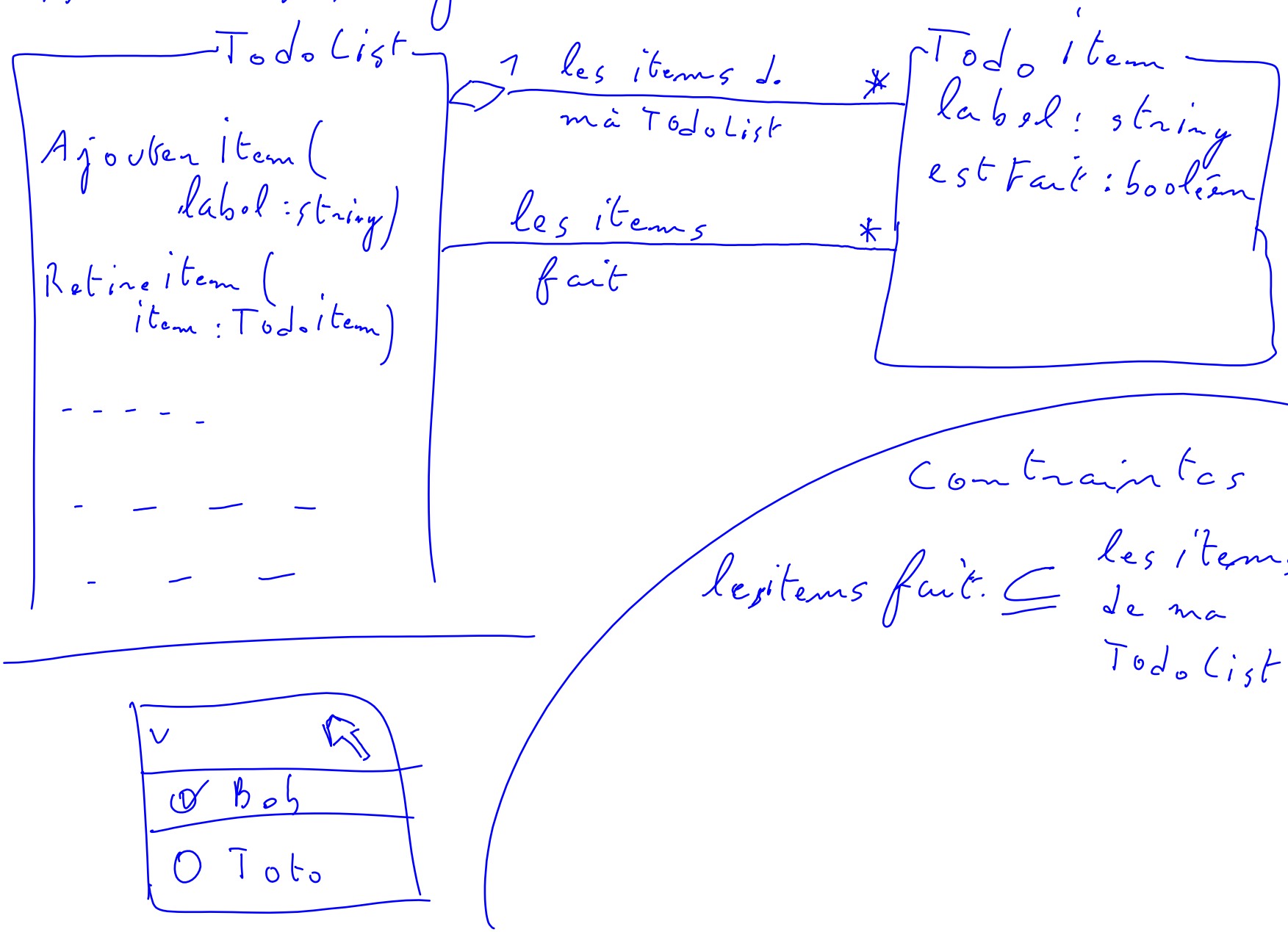
Mise en pratique: Liste de choses à faire

1 type d'objet -> les choses à faire

1 chose à faire <-> | 1 label (du texte)
| 1 indication pour savoir si la chose est faite ou pas (1 booléen)

1 autre type d'objet -> les listes de choses à faire

1 liste de choses à faire ?



TypeScript

Mise en pratique: Liste de choses à faire

TypeScript

Mise en pratique: Liste de choses à faire

TypeScript

Mise en pratique: Liste de choses à faire