

TypeScript



TypeScript

Outils



Accès à Internet (La DOC + package NPM)



Navigateur web (à jour) - La console + ES6+



Éditeur de code - Visual Studio Code (Intellisense)



On aura besoin de NPM pour installer TypeScript

Installation du compilateur TypeScript via NPM

Le compilateur vient avec un package permettant de comprendre le TypeScript



Vérifier que Node est installé: `node -v` ou `node --version`

```
sudo npm install -g typescript
```



C'est quoi TypeScript ?

TypeScript est un langage open source développé par Microsoft et devenu très populaire suite à son intégration depuis Angular 2.



C'est un sur-ensemble permettant d'enrichir le JavaScript avec des fonctionnalités supplémentaires. Il s'appuie sur le langage JavaScript (et ses mises à jours) afin de lui apporter d'avantage de capacités.

JavaScript a ses propres limites et qui peuvent causer des problèmes!




Exemple: Envoyer quelqu'un pour nous acheter une pizza



Limites du JavaScript

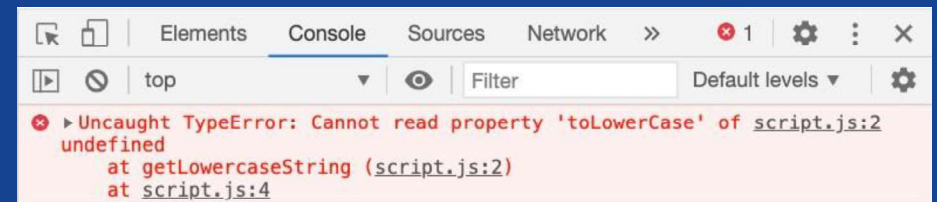
JavaScript n'exige pas de passer un argument au moment d'invoquer une fonction.



```
function getLowercaseString(arg) {  
    return arg.toLowerCase( );  
}
```

```
getLowercaseString( );
```

Cela va nous causer un problème!
On constatera l'erreur une fois
le code lancé !!!!



Solution TypeScript

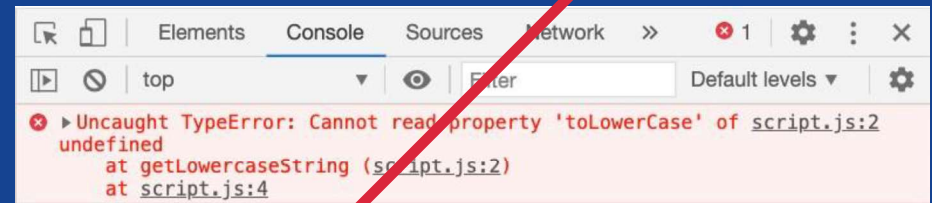


```
function getLowercaseString(arg: string) {  
    return arg.toLowerCase( );  
}
```

```
getLowercaseString( "Hello" );
```



```
getLowercaseString( 200 );
```

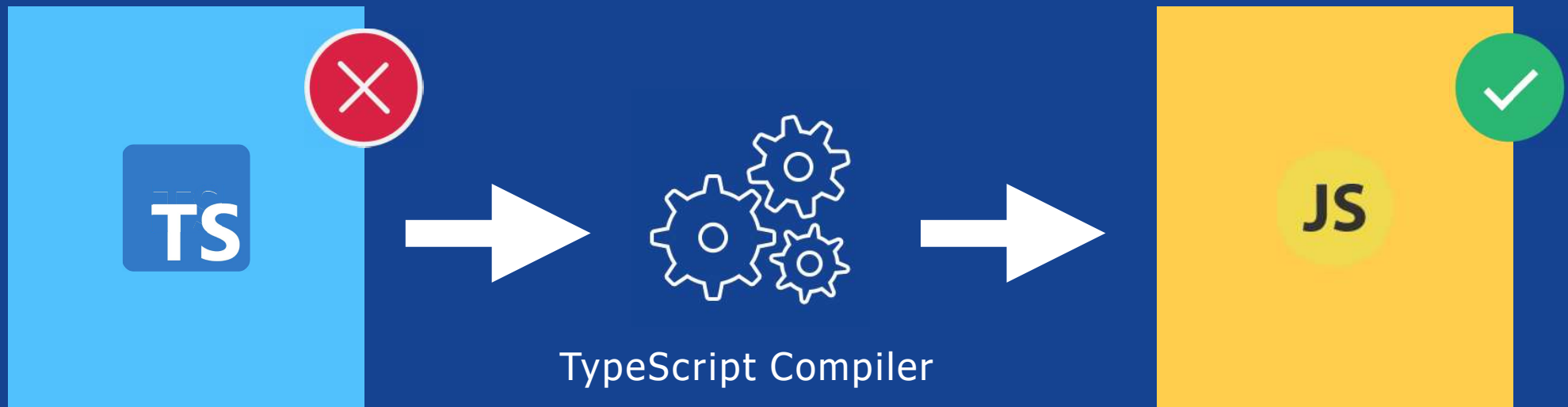


TypeScript exige de préciser le TYPE d'argument à passer dans la fonction.

- TypeScript exige de passer un argument
- correct au moment d'invoquer la fonction.

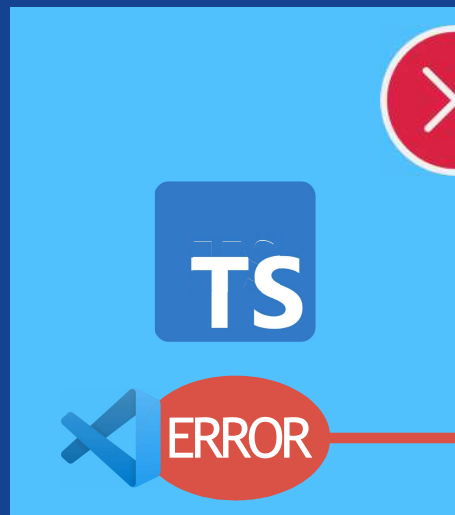
TypeScript et les navigateurs ?

Le code TypeScript n'est pas reconnu par les navigateurs web, il faut donc le compiler en langage JavaScript



Pourquoi TS pour obtenir du JS ?

Les fonctionnalités TS seront utiles ici et permettront d'obtenir un meilleur code JavaScript



TypeScript

Compiler

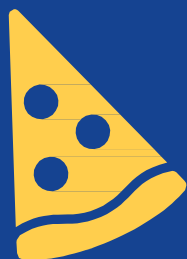
Et nous alerter

Des Erreurs



ERROR

JS



PIZZA type: 4 fromages



Compilateur TypeScript



Dois-je utiliser TypeScript dans tous mes projets ?

Il est conseillé d'utiliser TypeScript sur les grands projets.

- Permet de définir les types static de nos variables, fonctions ..etc (ce qui n'est pas le cas avec JavaScript)
- Permet une meilleure documentation du code (facile à comprendre)
- Vérifie que notre code fonctionne bien (constater les erreurs durant le dev)
- Permet d'écrire un code plus sûr et plus clean. (Cependant, TS n'est pas une raison pour ne pas tester votre code. Tout code qui se respecte doit être testé!)

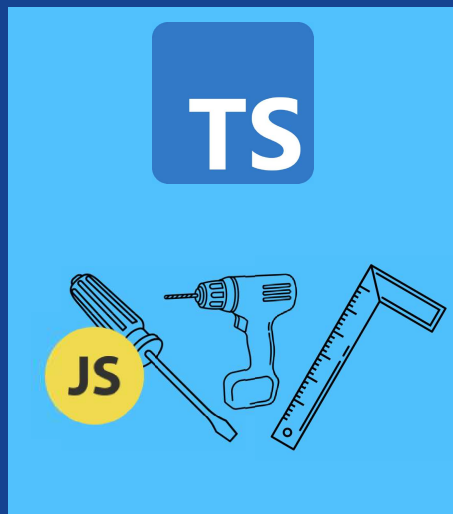
Types dynamiques acceptés en JavaScript mais peut causer des erreurs ou des incohérences



Types stricts en TypeScript



Compiler du TypeScript en JavaScript



TypeScript Compiler
TSC



TypeScript Compiler

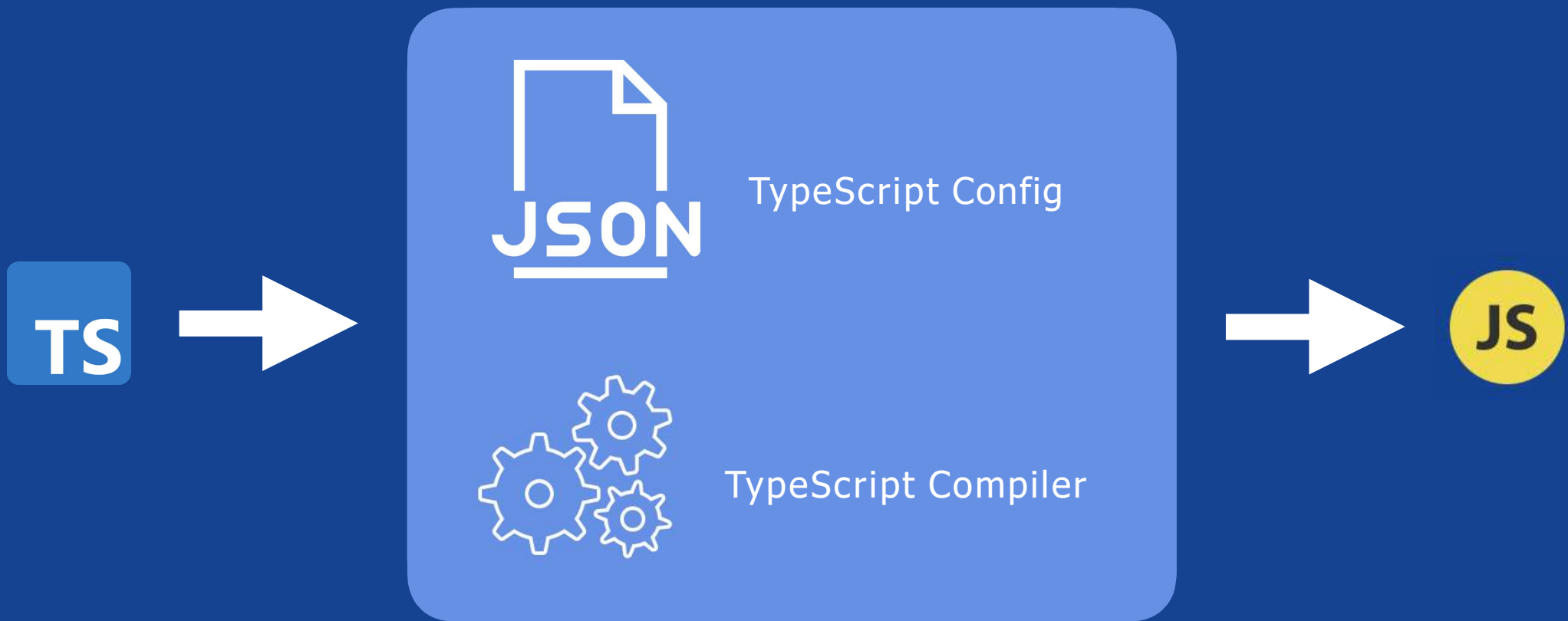


Installer le TypeScript Compiler

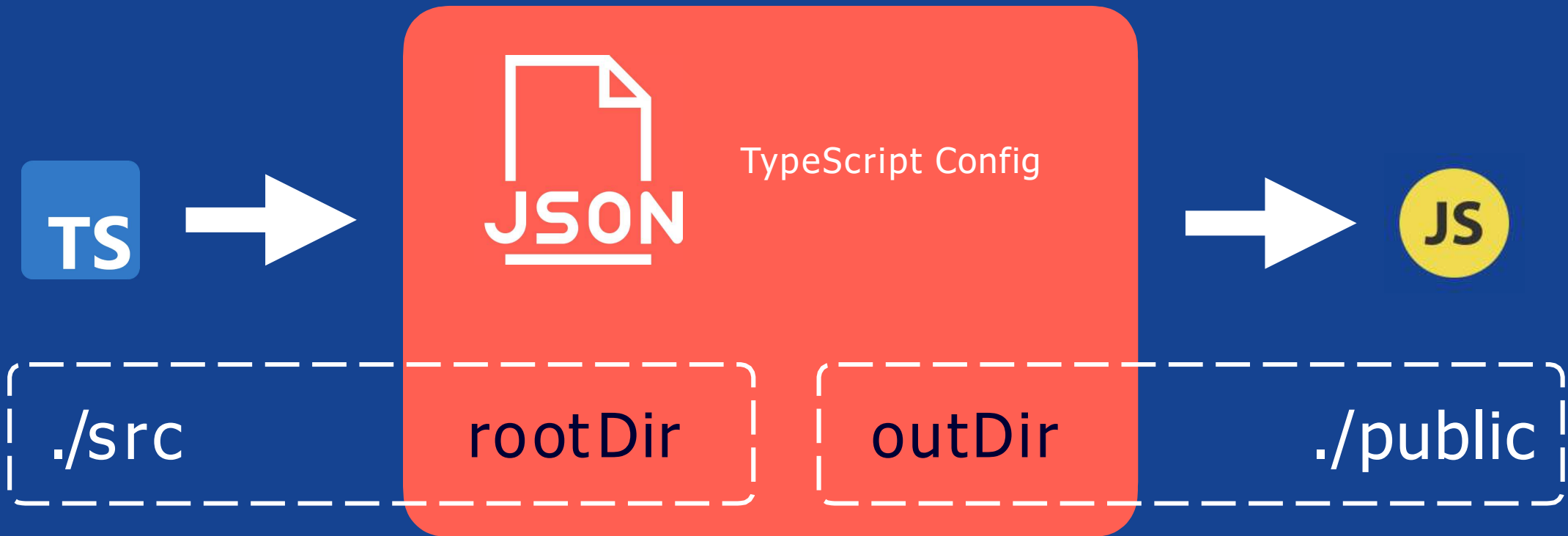
- Compiler les fichiers : **tsc file.ts**
- Compiler les fichiers avec des arguments : **tsc file.ts --out /js**
- Compiler les fichiers automatiquement après validation : **tsc --watch (ou -w)**

Au lieu de préciser les paramètres lors de la compilation, on peut les définir dans un fichier séparé qui va gérer tout cela automatiquement pour nous.

TypeScript Compiler - `tsconfig.json`



TypeScript Compiler - `tsconfig.json`



TypeScript Compiler - `tsconfig.json`

`target: "es5"`

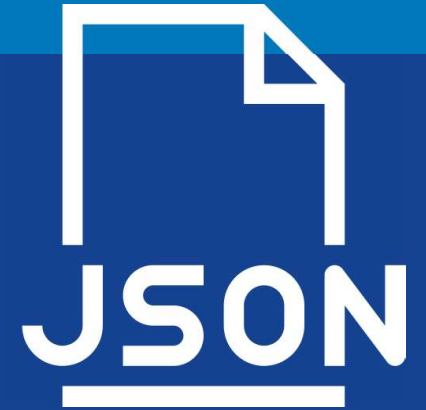
`module`

lib: Si désactivé, donc TS accède, par défaut, aux notions globales valides dans la version JS définie dans le « target ». Exemple: le DOM API comme l'objet document et ses méthodes, l'interface Math et ses méthodes, ..etc

Si activé, on aura plus les paramètres par défaut et le compiler ne détectera plus rien. Cela peut être utile si vous travaillez côté serveur et que vous n'avez pas forcément besoin des DOM types.

Nous allons donc devoir lui définir quelques bibliothèques et les définitions de types spécifiques sous forme de chaînes de caractères dans un array. (Voir la doc)

TypeScript Compiler - `tsconfig.json`

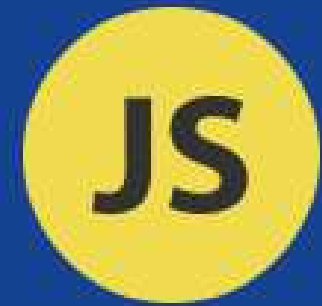


Quelques options tsc de base

- ✓ `allowJS`
- ✓ `checkJS`
- ✓ `jsx`
- ✓ `declaration`
- ✓ `declarationMap`
- ✓ `sourceMap`
- ✓ `removeComments`
- ✓ `noEmit`
- ✓ `downLevelIteration`

Le reste des options

Déclarations de Variables



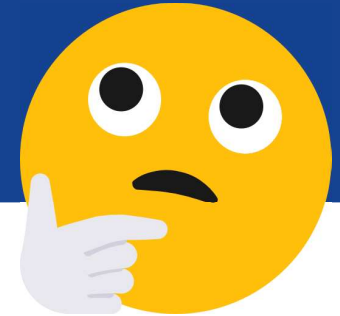
- TypeScript comprend JavaScript

The ES6 logo, consisting of the letters 'ES6' in black on a yellow rectangular background.

ES6

- Il nous encourage à utiliser les déclarations `let` et `const` (éviter quelques problèmes)

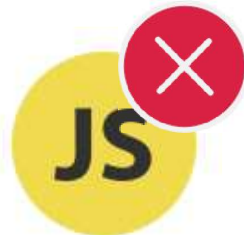
TypeScript types



TypeScript utilise les types **stricts**
ce qui n'est pas le cas du JavaScript



Types stricts
Durant le Dev



Types dynamiques
Lors de l'exécution

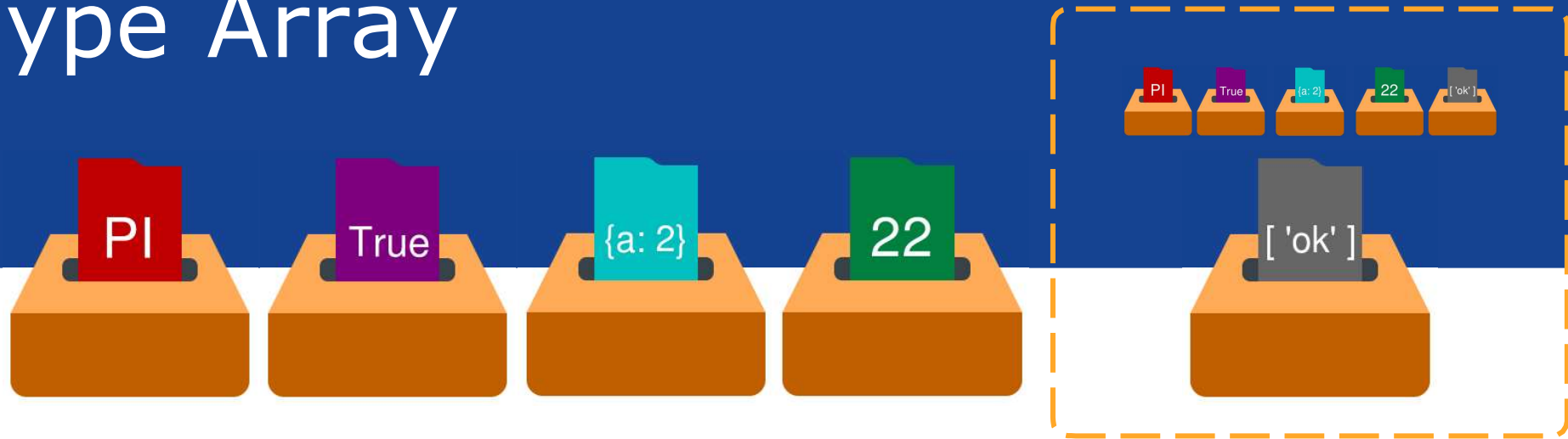


IntelliSense

- Quels sont ces types ?
- Sont-ils obligatoires en TypeScript?

Types par **inférence** Vs types par **Attribution**

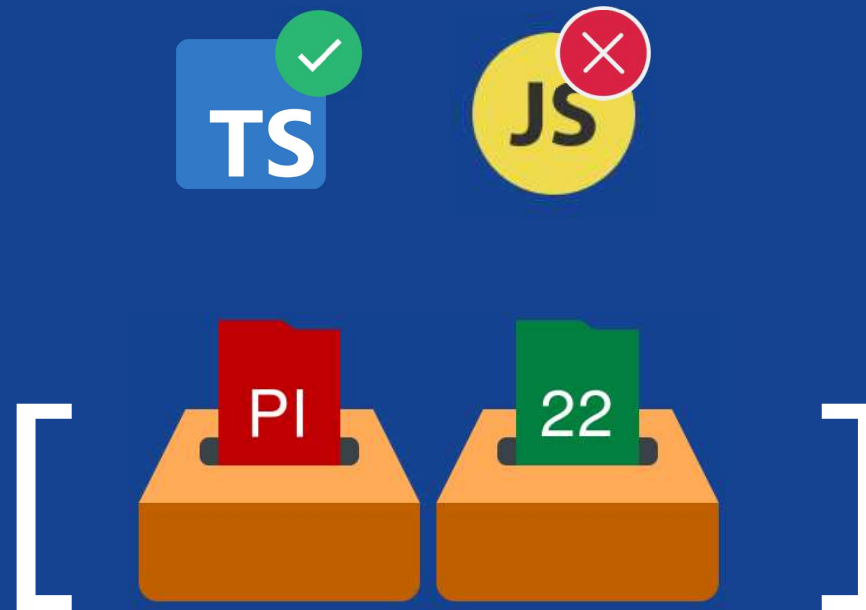
Type Array



JavaScript et TypeScript permettent de stocker ce qu'on veut dans un Array.

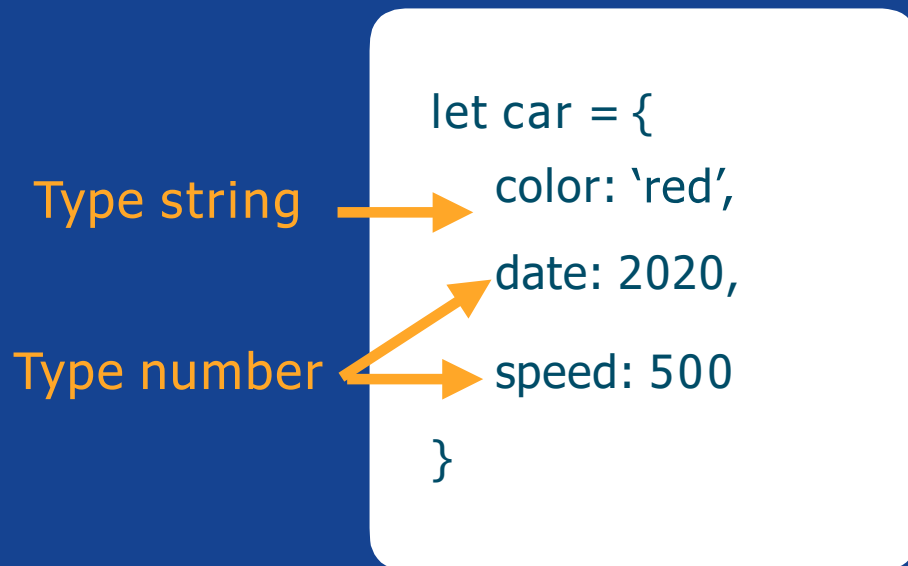
- Number, Boolean, Object, String, Array ...
- Mélanger toutes ces contenus ensemble dans un même Array

Tuple Type



Définir le nombre d'éléments dans un Array ainsi que leurs types

Type Object



Un type objet représente un type non primitif, c'est-à-dire tout ce qui n'est pas (number, string, boolean, symbol, null, or undefined).

Via l'inférence, TypeScript attribut un type aux propriétés d'un objet en fonction des valeurs indiquées.

Exercices - Typage



Révisions

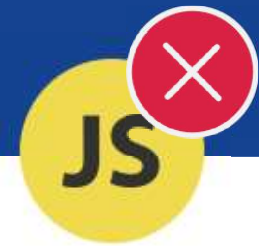


Note:

Un Array est également un objet. `{ } = []`

Donc on peut parfaitement assigner une valeur `[]` à un object ayant été défini en TypeScript en tant que tel.

Enums



Enum n'existe pas en JavaScript.

C'est un moyen utilisé dans TypeScript pour nommer des ensembles de valeurs d'une façon numérique.

C'est aussi un moyen qui nous facilite la lecture du code !

```
If ( user.level === 1 )
```

```
If ( user.level === Level.ADMIN )
```

Any Type

Via «any», le TypeScript n'impose aucun type particulier!

On peut y stocker tous les types



A utiliser avec modération! Vous n'avez aucun contrôle des types et le compilateur TypeScript ne va pas vérifier votre code!

Peut être utile dans certains cas. Exemple: On ne sait pas quel type de data à récupérer d'une API, d'un formulaire, on souhaite modifier le type etc.. (+ vérification, typeof)



Unknown Type

Via «Unknown », le TypeScript définit un type comme étant inconnu!

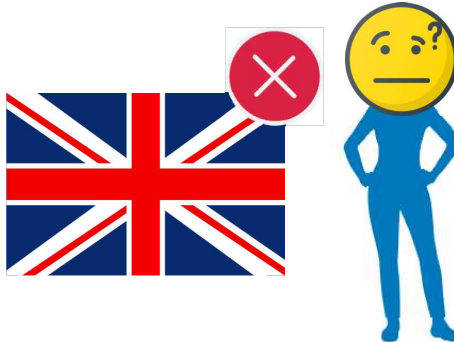
Comme le type «Any », Unknown accepte tous les types



Pour pouvoir utiliser le type Unknown, vous devez d'abord vérifier le type

Exemple

Variable



Datas types Objets



Pomme



Livre



Pomme



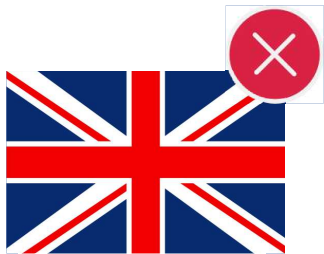
Livre

TYPES: COMESTIBLES



Exemple avec vérification

Variable



Datas types Objets



Pomme



Livre



Pomme



Livre

TYPES: COMESTIBLES





Any

Vs



Unknown

+ Vérification

Function & Void Type



Comme les Enum et les tuple, «void » aussi n'existe pas en JavaScript

De ce fait:

- Une fonction qui retourne une valeur définit le type de la valeur retournée via (via inférence ou Attribution.
- Une fonction qui ne retourne rien définit également un type «void » pour Undefined

Function Types



Type Function générique: (Une seule contrainte: le type doit être une fonction)

Types bien spécifiques: Non seulement c'est une fonction mais celle-ci doit être très explicite : `(param: type) => return type`

Function Types



- Paramètres facultatifs
- Paramètres par défaut

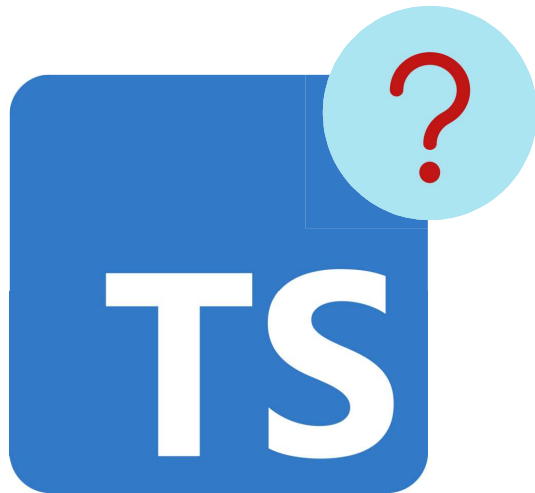
Null



Undefined



Type Assertions (Affirmatif)



C'est un Number !!!!

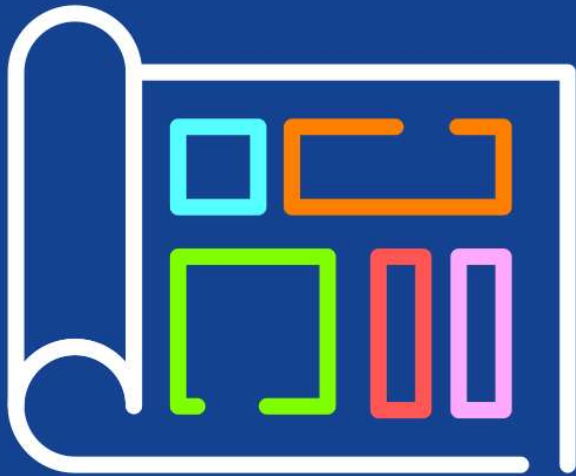
Fais moi confiance, je
sais ce que je fais :-)

Type Assertions (Affirmatif)

An illustration of a dark blue rectangular form. At the top center is a green circle with a white checkmark. Below it are two horizontal white input fields. At the bottom is a red rectangular button with the word 'VALIDER' in white capital letters.

Les Classes

Programmation Orientée Objets en TS



Class - Plan



Instanciation



Objet

Les Classes

: Invoice

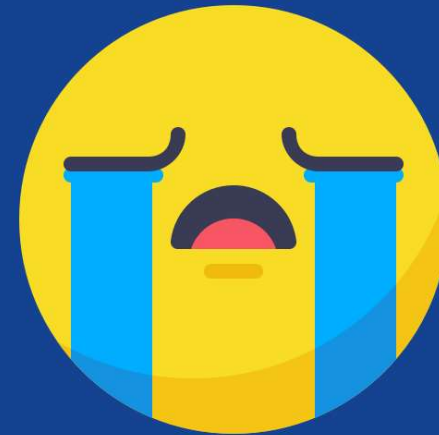
Respecter les caractéristiques d'un objet

: Invoice[]

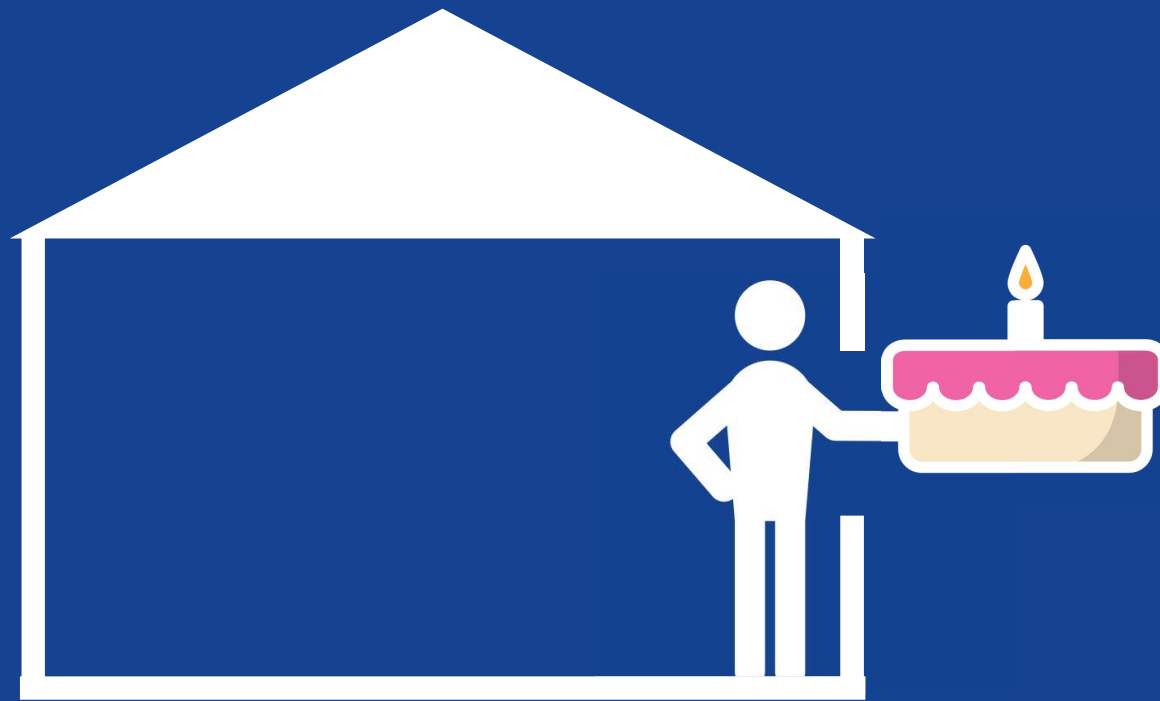
Un array d'objets « Invoice »



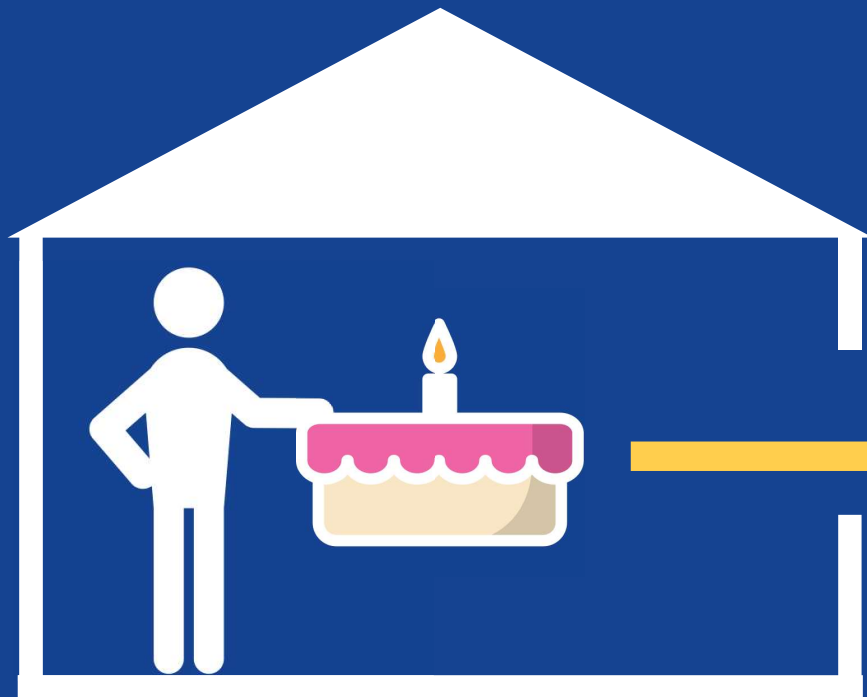
Public (accessible à l'extérieur)



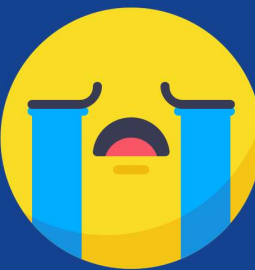
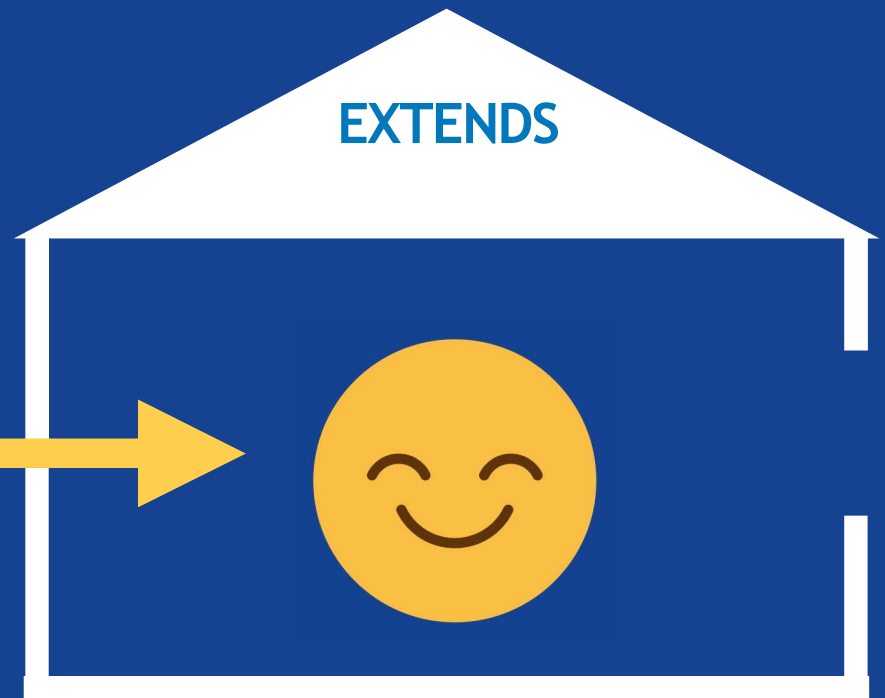
Private (Accessible seulement dans la classe)



Getters & Setters



Protected





readonly (Peut être modifié que via le constructor)

Les Classes

Héritage et Polymorphisme en TypeScript

Personne



Propriétés

- Nom et prénom
- Cheveux châtain
- Yeux: 2
- Oreilles: 2 ... etc

Méthodes

- Parler - Rire ..etc

Héritage



Les Classes

Access Modifiers

Personne



Propriétés

- Nom et prénom
- Cheveux châtain
- Yeux: 2
- Oreilles: 2 ... etc

Méthodes

- Parler - Rire ..etc

Héritage



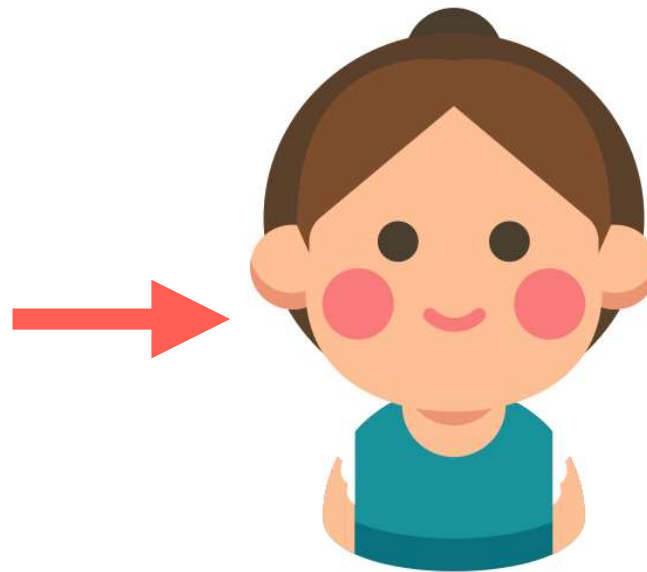
Les Classes (static)

class Person

- Nom et prénom
- Cheveux châtain
- Yeux: 2
- Oreilles: 2 ... etc

- Parler - Rire ..etc

Object



parler()

rire()

TP

Interfaces

class Mother

- Nom et prénom
- Cheveux châtain
- Yeux: 2
- Oreilles: 2 ... etc

- Parler - Rire ..etc

Personne: Mother



Interface type

Définir les contraintes d'un objet
propriétés méthodes

Interfaces avec les classes

class Mother implements «Interface»

- Nom et prénom
- Cheveux châtain
- Yeux: 2
- Oreilles: 2 ... etc

- Parler - Rire ..etc

Interface

Définir les contraintes d'un objet
propriétés méthodes



En implémentant une Interface à une classe, on s'assure que l'objet instancié est conforme aux spécificités définies dans l'interface.

Exemple: Si l'interface exige une méthode speak(), l'objet instancié d'une classe qui implémente cette interface doit pouvoir invoquer cette méthode. Donc il pourra parler..



Union Type

string | number | boolean



Exercise



Intersection Type

Person & number & boolean





Type Aliases

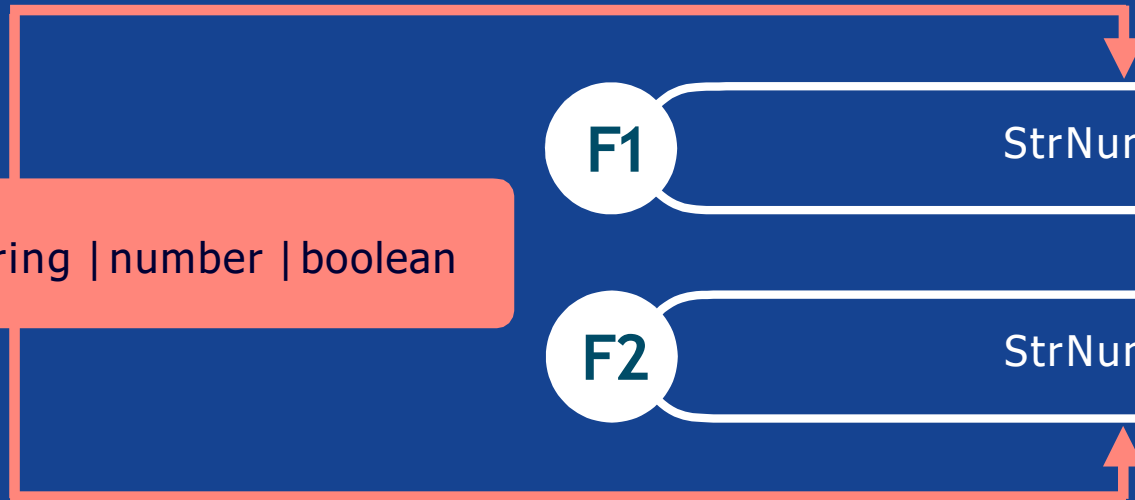
```
type StrNumBoo = string | number | boolean
```

F1

StrNumBoo

F2

StrNumBoo



Literal Type (Littéral - exact)



Union Type

F1

arg : string | number

'Hello'



200



true



Literal Type

F2

arg : 'toto' | 'number2'

'Hello'



200



true



'toto'



'number2'



Conditional Types

Un type conditionnel sélectionne, via un opérateur conditionnel, l'un des deux types possibles en fonction d'une condition exprimée sous la forme d'un test de relation de type

 extends  ? "Fruit" : "légume"



Mapped Types

Les mapped Types nous permettent de créer un nouveau type on nous basant sur des types déjà existants



Les Classes abstraites

(abstract)

abstract class Person

- Nom et prénom
- Cheveux châtain
- Yeux: 2
- Oreilles: 2 ... etc

Speak()

abstract walk() *signature*

class Mother extends Person

walk() est obligatoires

Object





Generics



Type Any

Accepte tous les types mais ne les détecte pas



Type *Generic*

- Détecter et gérer différents types
- Étant pas spécifique à un type, il peut donc être réutilisable

```
function getData( arg: any ) {  
  return arg  
}
```



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



- Generics avec Interfaces
- Generics avec les classes
- Generic Utility Types

TP

Omit<Type, Keys>

```
interface Todo {  
    title: string;  
    description: string  
    author: string  
}
```

```
Omit<Todo, "title" | "author" > = {  
    description: string;  
}
```

Le Generic Utility Type "Omit" est le contraire de Pick. Il permet de construire un nouveau Type en sélectionnant les propriétés Keys dans le Type utilisé (Interface) et en retirants certaines autres clés.

Exclude<Type, ExcludedUnion>

Le Generic Utility Type "Exclude" permet de construire un nouveau Union Type en excluant des types définis dans un autre Union Type.

```
type A = string | string[ ] | boolean
```

```
Exclude<A, boolean> = string | string[ ]
```


Extract<Type, Union>

Le Generic Utility Type "Extract" est l'opposé de «Exclude». Il permet de construire un nouveau type en extrayant des «Union membres» qui peuvent être affectés à «Union».

En gros, on va extraire ce qu'on souhaite utiliser dans notre nouveau type.

```
type A = string | string[ ] | boolean
```

```
Extract<A, boolean> = boolean
```

Decorators

Les decorators sont actuellement en phase expérimentale et peuvent subir des changements dans les prochaines versions JavaScript. Néanmoins, je les aborde dans cette formation car c'est un pattern assez important adopté largement dans les récents frameworks tel que Angular

C'est simplement un moyen de mettre un morceau de code dans une fonction dans le but d'étendre les fonctionnalités d'une classe par exemple.

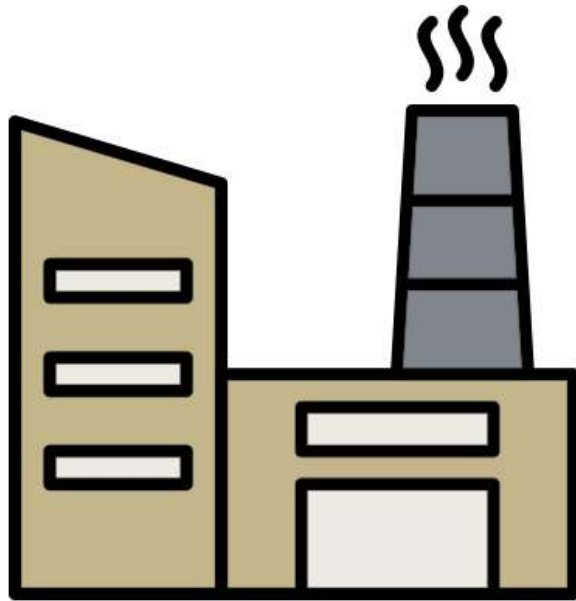
✓ *"experimentalDecorators": true*

✓ *"target": "es6"*



Decorator Factories

Fonction qui génèrent les decorators



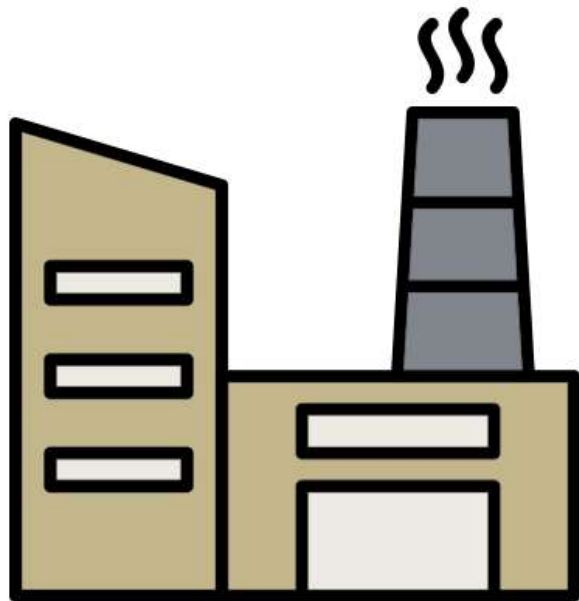
Function(param)



Anonymous Function

Decorator Factories

Injecter du contenu dans le DOM à la manière Angular via un decorator `@component`

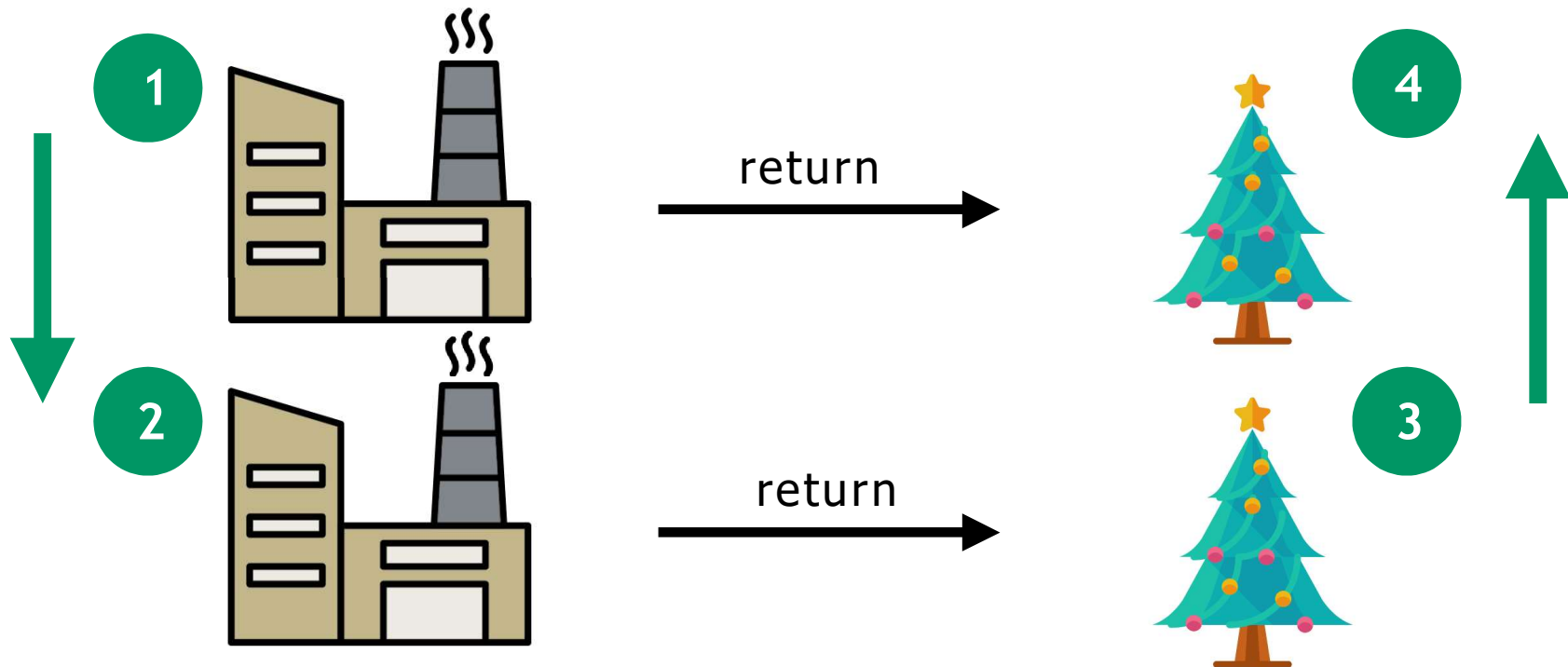


Function(param)



Anonymous Function

Multiple Decorator Factories



Class Decorator

✓ Propriétés

Accessors (getters - setters)

Méthodes

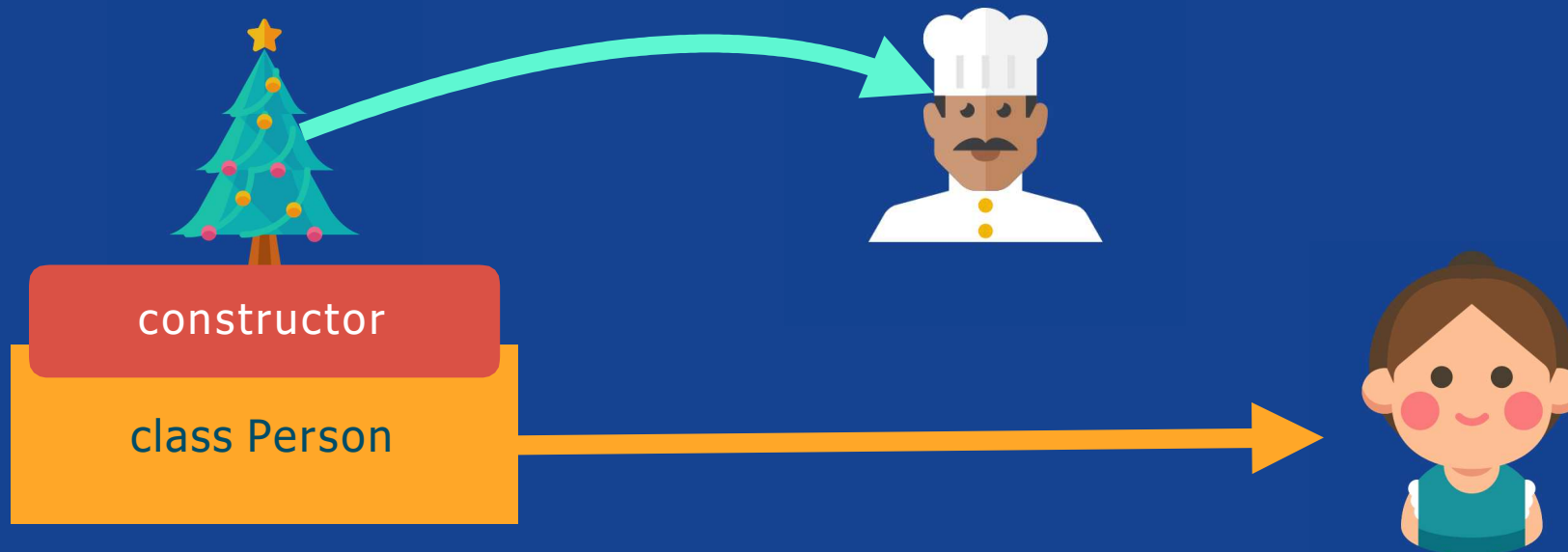
Paramètres

Class Decorator

- ✓ Propriétés
- ✓ Accessors (getters - setters)
- ✓ Méthodes
- ✓ Paramètres

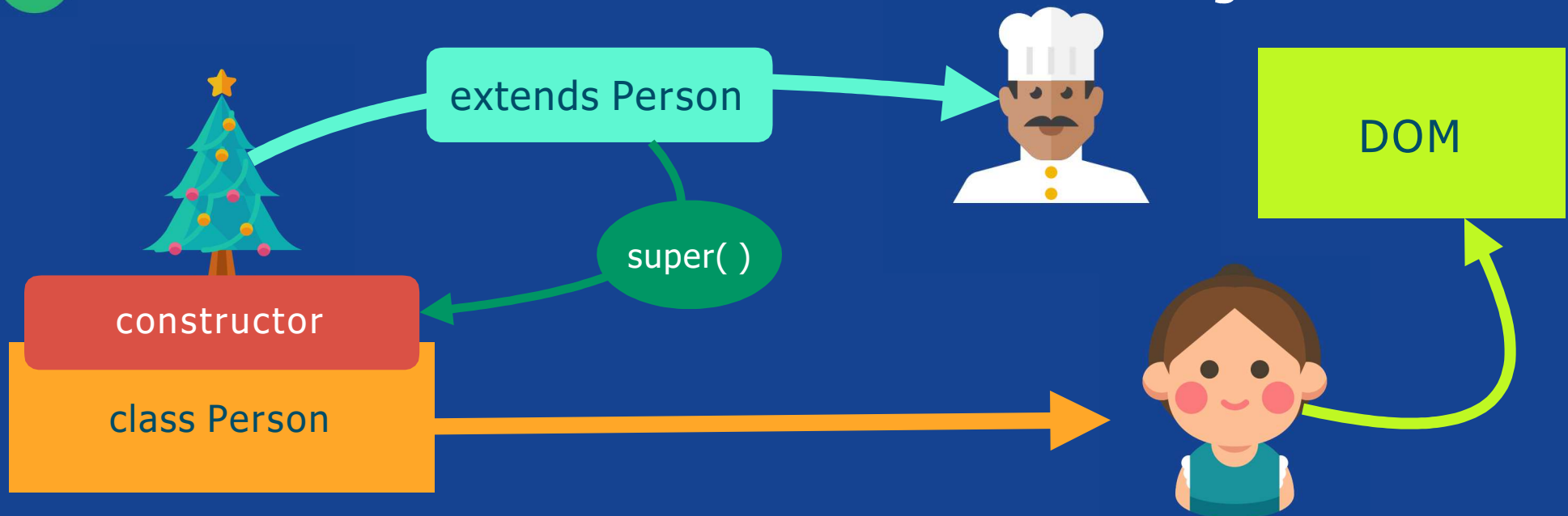
Class Decorator

- ✓ class décorateur peut réécrire la fonction constructor

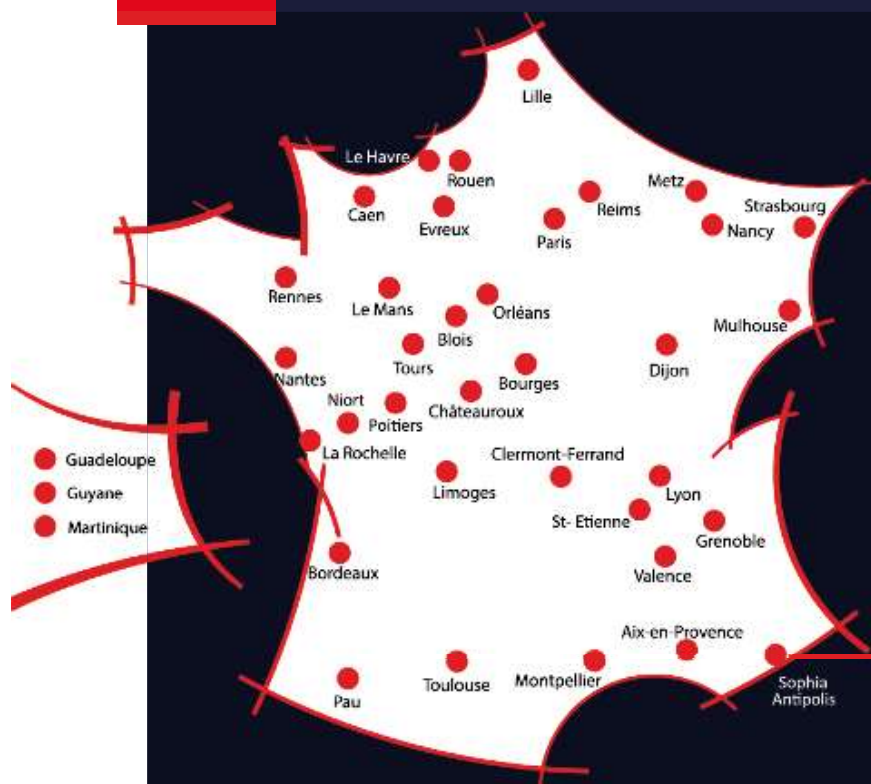


Class Decorator

- ✓ Réécrire la fonction constructor via l'héritage



TP



Découvrez également
l'ensemble des stages à votre disposition
sur notre site m2information.fr

m2information.fr

