TYPESCRIPT 201

FAITES DU COMPILATEUR VOTRE MEILLEUR AMI

https://blemoine.github.io/ts-201-prez/

QUI SUIS JE?

- Benoît Lemoine
- Développeur Fullstack
  (TS, Scala, Rust, ...)
  aimant la programmation
  fonctionnelle
  chez Decathlon à Montréal
- https://activities.decathlon.ca
- @benoit_lemoine

CE DONT ON NE VAS PAS PARLER

CE DONT ON NE VAS PAS PARLER

MEILLEURE PRATIQUES EN JAVASCRIPT

OBJECTIFS

Le compilateur et les types doivent nous aider à

OBJECTIFS

Le compilateur et les types doivent nous aider à

- Découvrir les erreurs à la compilation, et pas au runtime

OBJECTIFS

Le compilateur et les types doivent nous aider à

- Découvrir les erreurs à la compilation, et pas au runtime
- maintenir et écrire le code

OBJECTIFS

Le compilateur et les types doivent nous aider à

- Découvrir les erreurs à la compilation, et pas au runtime
- maintenir et écrire le code
- augmenter notre confiance dans le code

OBJECTIF PRINCIPAL

OBJECTIF PRINCIPAL

SI CA COMPILE, C'EST QUE CA MARCHE!

RÈGLE DE BASE

RÈGLE DE BASE

LE COMPILATEUR A RAISON,

RÈGLE DE BASE

LE COMPILATEUR A RAISON,

VOUS AVEZ TORT !

AIDEZ TYPESCRIPT À VOUS AIDER

NULL/UNDEFINED SAFETY

```
function trim(s: string): string {
  return s.trim();
}
trim(null);
```

NULL/UNDEFINED SAFETY

```
function trim(s: string): string {
  return s.trim();
}
trim(null);
```

```
// au runtime
Cannot read properties of null
```

NULL/UNDEFINED SAFETY

```
// in tsconfig.json
"strictNullChecks": "true"
```

```
function trim(s: string): string {
  return s.trim();
}
trim(null);
```

NULL/UNDEFINED SAFETY

```
// in tsconfig.json
"strictNullChecks": "true"
```

```
function trim(s: string): string {
  return s.trim();
}
trim(null);
```

```
Argument of type 'null' is not assignable
              to parameter of type 'string'.

trim(null);
     ~~~~
```

FUNCTION/METHOD SUPPORT

```
function trimStart(s: string | null): string {
  return s?.trimStart() || '';
}
trimStart(null);
```

FUNCTION/METHOD SUPPORT

```
function trimStart(s: string | null): string {
  return s?.trimStart() || '';
}
trimStart(null);
```

```
Property 'trimStart' does not exist on type 'string'.
Do you need to change your target library?
Try changing the 'lib' compiler option to 'es2019' or later.

return s?.trimStart() || '';
           ~~~~~~~~~
```

FUNCTION/METHOD SUPPORT

```
// in tsconfig.json
"lib": ["es2019"]
```

```
function trimStart(s: string | null): string {
  return s?.trimStart() || '';
}
trimStart(null);
```

FUNCTION/METHOD SUPPORT

```
// in tsconfig.json
"lib": ["es2019"]
```

```
function trimStart(s: string | null): string {
  return s?.trimStart() || '';
}
trimStart(null);
```

```
// Compilation result
function trimStart(s) {
    return (s === null || s === void 0 ?
            void 0 :
            s.trimStart()) || '';
}
trimStart(null);
```

SYNTAX SUPPORT

```
// in tsconfig.json
"target": "es2020",
"lib": ["es2019"]
```

```
function trimStart(s: string | null): string {
  return s?.trimStart() || '';
}
trimStart(null);
```

SYNTAX SUPPORT

```
// in tsconfig.json
"target": "es2020",
"lib": ["es2019"]
```

```
function trimStart(s: string | null): string {
  return s?.trimStart() || '';
}
trimStart(null);
```

```
// Compilation result
function trimStart(s) {
    return s?.trimStart() || '';
}
trimStart(null);
```

IMPLICIT any

```
function trimStart(s) {
  return s.trimStart();
}
trimStart(1);
```

IMPLICIT any

```
function trimStart(s) {
  return s.trimStart();
}
trimStart(1);
```

```
// runtime
s?.trimStart is not a function
```

IMPLICIT any

```
// in tsconfig.json
"noImplicitAny": "true"
```

```
function trimStart(s) {
  return s.trimStart();
}
trimStart(1);
```

IMPLICIT any

```
// in tsconfig.json
"noImplicitAny": "true"
```

```
function trimStart(s) {
  return s.trimStart();
}
trimStart(1);
```

```
Parameter 's' implicitly has an 'any' type.

function trimStart(s): string {
                       ~
```

EXPLICIT any

```
function trimStart(s: any) {
  return s.trimStart();
}
trimStart(1);
```

EXPLICIT any

```
function trimStart(s: any) {
  return s.trimStart();
}
trimStart(1);
```

```
// runtime
s?.trimStart is not a function
```

N'UTILISEZ PAS any

```
function trimStart(s: string): string {
  return s.trimStart();
}
trimStart(1);
```

```
Argument of type '1' is not assignable
  to parameter of type 'string'.
```

UTILISEZ unknown

```
function trimStart(s: unknown): string {
  if (typeof s === 'string') {
    return s.trimStart();
  } else {
    return ''
  }
}
trimStart(1);
```

STRICT

```
"strict": "true"
```

CHANGER LA CONFIGURATION D'UN PROJET EXISTANT ?

- Projet utilisant un compilateur sans typechecking (babel, swc, etc.)
- Projet existant avec de nombreux fichiers possiblement pleins d'erreurs

CHANGER LA CONFIGURATION D'UN PROJET EXISTANT !

- Créer un fichier `tsconfig.json` avec les règles stricts
- Utiliser cette conf comme linter `tsc --noEmit`
- Ajouter les fichiers un par un jusqu'a avoir couvert tout le projet

TYPEZ EXPLICITEMENT

CE QUI EST PUBLIQUE/EXPORTÉ

TYPEZ EXPLICITEMENT

CE QUI EST PUBLIQUE/EXPORTÉ

UTILISEZ TYPESCRIPT-ESLINT

https://github.com/typescript-eslint/typescript-eslint

OPTIONAL IN FUNCTIONS

```
function test(a: string, b?: string): string {
    return b ?? a;
}

test("one"); // return one;
test("one", "two"); // return two;
```

OPTIONAL IN FUNCTIONS

```
function test(a: string, b?: string): string {
   return b ?? a;
}

test("one"); // return one;
test("one", "two"); // return two;
```

```
// I need to do a change
function test(a: string, c:string, b?: string): string {
   return b ?? (a + c);
}

test("one"); // An argument for 'c' was not provided.
test("one", "two"); // compile fines, but returns "one-two"
```

OPTIONAL IN FUNCTIONS

```
const myfn = (now?:Date) => console.log('it is now', now);
function TopComponent() {
  return <BottomComponent onclick={myfn} />
}

function BottomComponent({onclick}: {onclick: () => void}) {
    return <button onClick={onclick}>TEST</button>
}
```

```
it is now SyntheticBaseEvent {
  _reactName: 'onClick', _targetInst: null, type: 'click',
  nativeEvent: PointerEvent, target: button, …}
```

OPTIONAL IN MOCKS

```
type User = { id:string, name:string, age:number };
function mockUser(u?: Partial<User>): User {
  return { id: '1', name: 'JJ', age: 3 , ...u }
}

const fakeUser1 = mockUser();
const fakeUser2 = mockUser({id: '12'});
```

MODÉLISER AVEC LES TYPES LE PLUS PRÉCIS POSSIBLE

MODÉLISER AVEC LES TYPES LE PLUS PRÉCIS POSSIBLE

LES ÉTATS INVALIDES DEVRAIENT ÊTRE IMPOSSIBLE À REPRÉSENTER

LES TYPES LITERAUX

```
interface User {
    id: string;
    role: 'admin' | 'superadmin'
}

const user: User = { id:'1', role: 'ADMIN' };
```

```
Type '"ADMIN"' is not assignable to type '"admin" | "superadmin"'.
```

LES TYPES LITERAUX

```
interface Beer {
    name: string;
    sizeInMl: 250 | 330 | 475
}

const beer: Beer = { name: 'westvleteren', sizeInMl: 500 };
```

```
Type '500' is not assignable to type '250 | 330 | 475'.
```

LES TYPES UNION

```
type BeerBottle = { sizeInMl: 250;  material: "glass"; };
type BeerCan = { sizeInMl: 330 | 475; material: "aluminium";};
type Beer = BeerBottle | BeerCan;

const beer: Beer = {
  material: "aluminium",
  sizeInMl: 250,
};
```

```
Type '{ material: "aluminium"; sizeInMl: 250; }'
              is not assignable to type 'Beer'.
  Type '{ material: "aluminium"; sizeInMl: 250; }'
              is not assignable to type 'BeerCan'.
    Types of property 'sizeInMl' are incompatible.
      Type '250' is not assignable to type '330 | 475'.

const beer: Beer = {
      ~~~~
```

TYPE GUARD

```
type BeerBottle = {material: "glass"};
type BeerCan = {material: "aluminium"; color: "blue" | "red"};
type Beer = BeerBottle | BeerCan;

function getColor(beer: Beer): "blue" | "red" | "transparent" {
  return beer.material === "aluminium" ?
           beer.color :
           "transparent";
}
```

TYPE GUARD

```
type BeerBottle = {material: "glass"};
type BeerCan = {material: "aluminium"; color: "blue" | "red"};
type Beer = BeerBottle | BeerCan;

function isCannedBeer(beer: Beer): beer is BeerCan {
  return beer.material === "aluminium";
}

function getColor(beer: Beer): "blue" | "red" | "transparent" {
  return isCannedBeer(beer) ? beer.color : "transparent";
}
```

HANDLING NEW CASE

```
type BeerBottle = {material: "glass"};
type BeerCan = {material: "aluminium"; color: "blue" | "red"};
type BeerKeg = {material: "wood"}

type Beer = BeerBottle | BeerCan | BeerKeg;

function isCannedBeer(beer: Beer): beer is BeerCan {
  return beer.material === "aluminium";
}

function getColor(beer: Beer): "blue" | "red" | "transparent" {
  return isCannedBeer(beer) ? beer.color : "transparent";
}

// returns 'transparent' !
getColor({  material: "wood" })
```

HANDLING NEW CASE

```
type BeerBottle = {material: "glass"};
type BeerCan = {material: "aluminium"; color: "blue" | "red"};
type BeerKeg = {material: "wood"}
type Beer = BeerBottle | BeerCan | BeerKeg;

function getColor(beer: Beer): "blue" | "red" | "transparent" {
  switch (beer.material) {
    case "aluminium":
      return beer.color;
    case "glass":
      return "transparent"
  }
}
```

```
Function lacks ending return statement and
             return type does not include 'undefined'.

function getColor(beer: Beer): "blue" | "red" | "transparent"
```

NEVER/CANNOTHAPPEN

```
type BeerBottle = {material: "glass"};
type BeerCan = {material: "aluminium"; color: "blue" | "red"};
type BeerKeg = {material: "wood"}
type Beer = BeerBottle | BeerCan | BeerKeg;

function cannotHappen(x: never): never {
  throw new Error(`${x} is not valid value`);
}
function getColor(beer: Beer): "blue" | "red" | "transparent" {
  switch (beer.material) {
    case "aluminium":
      return beer.color;
    case "glass":
      return "transparent"
    default:
      cannotHappen(beer);
  }
}
```

```
Argument of type 'BeerKeg' is not
              assignable to parameter of type 'never'.
```

AVOIR DES SIGNATURES PRÉCISES

```
interface User {  id: string;  name: string; }
interface Company {  id: string;  address: string; }

type UserWithEmail = User & { email: string };
type CompanyWithEmail = Company & { email: string };
function addEmail(
  userOrCompany: User | Company
): UserWithEmail | CompanyWithEmail {
  return { ...userOrCompany, email: "georges@abitbol.com" };
}

const user: User = {id:'1', name:'Georges'};
// result is UserWithEmail | CompanyWithEmail
const result = addEmail(user);
```

AVOIR DES SIGNATURES PRÉCISES

```ts
interface User {  id: string;  name: string; }
interface Company {  id: string;  address: string; }

function addEmail<T extends User | Company>(
  userOrCompany: T
): T & { email: string } {
  return { ...userOrCompany, email: "georges@abitbol.com" };
}

const user: User = { id: "1", name: "Georges" };
// result is User & {email:string}
const result = addEmail(user);
```

OVERLOADING

```
function getLength(str: string | null): number | null {
  return str === null ? null : str.length;
}

// result is of type number | null
const result = getLength(null)
```

OVERLOADING

```
function getLength(str: null): null;
function getLength(str: string): number;
function getLength(str: string | null): number | null {
  return str === null ? null : str.length;
}
const res = getLength(null); // res is of type null
const res1 = getLength('test'); // res1 is of type number
```

AS CONST

```
interface Beer {
  name: string;
  sizeInMl: 250 | 330 | 475
}

const beer = { name: 'pabs', sizeInMl: 250};
const beer2: Beer = beer;
```

```
 Type '{ name: string; sizeInMl: number; }'
              is not assignable to type 'Beer'.
  Types of property 'sizeInMl' are incompatible.
    Type 'number' is not assignable to type '250 | 330 | 475'.

const beer2: Beer = beer;
```

AS CONST

```
interface Beer {
  name: string;
  sizeInMl: 250 | 330 | 475
}

const beer = { name: 'pabs', sizeInMl: 250} as const;
const beer2: Beer = beer;
```

AS CONST

```
interface Beer {
  name: string;
  sizeInMl: 250 | 330 | 475
}

const beer = { name: 'pabs', sizeInMl: 250} as const;
const beer2: Beer = beer;
```

```
beer.name = 'molson';
```

AS CONST

```ts
interface Beer {
  name: string;
  sizeInMl: 250 | 330 | 475
}

const beer = { name: 'pabs', sizeInMl: 250} as const;
const beer2: Beer = beer;
```

```ts
beer.name = 'molson';
```

```
Cannot assign to 'name' because it is a read-only property.

beer.name = 'molson';
     ~~~~
```

TUPLE ANNOTÉ

```
function getCoordinates(): readonly [number, number] {
  return [1, 2] as const;
}
```

TUPLE ANNOTÉ

```
function getCoordinates(): readonly [number, number] {
  return [1, 2] as const;
}
```

```
function getCoordinates(
   ): readonly [latitude:number, longitude:number] {
  return [1, 2] as const;
}
```

GESTION D'ERREUR

TYPESCRIPT EST INCONSCIENT DES ERREURS

```
function divide(num:number, denom: number): number {
  if ( denom === 0 ) {
    throw new Error('Cannot divide by 0');
  }
  return num / denom;
}

divide(1,0)
```

```
Uncaught Error: Cannot divide by 0
```

GESTION D'ERREUR

TYPESCRIPT EST INCONSCIENT DES ERREURS

```
function divide(num:number, denom: number): number | Error {
  if ( denom === 0 ) {
    return new Error('Cannot divide by 0');
  }
  return num / denom;
}

const result = divide(1,0)
result + 1;
```

```
Operator '+' cannot be applied to
              types 'number | Error' and 'number'.
```

GESTION D'ERREUR

GESTION D'ERREUR

- `throw` Erreur si n'est pas récupérable

GESTION D'ERREUR

- `throw` Erreur si n'est pas récupérable
- `return` Erreur si on veut que l'appelant gère l'erreur

GESTION D'ERREUR AVANCÉ

REGARDER LE PATTERN `Either` (AKA `Result`)

eg. fp-ts https://gcanti.github.io/fp-ts/modules/Either.ts.html

CONCLUSION

CONCLUSION

- Ne mentez pas au compilateur ! (ie. pas de cast, pas de conversion sans validation, pas de `any`, etc.)

CONCLUSION

- Ne mentez pas au compilateur ! (ie. pas de cast, pas de conversion sans validation, pas de `any`, etc.)
- Plus vous serez précis dans vos types, plus le compilateur pourra vous aider