

The 20 most painful  
profitable typescript  
minutes of your life!

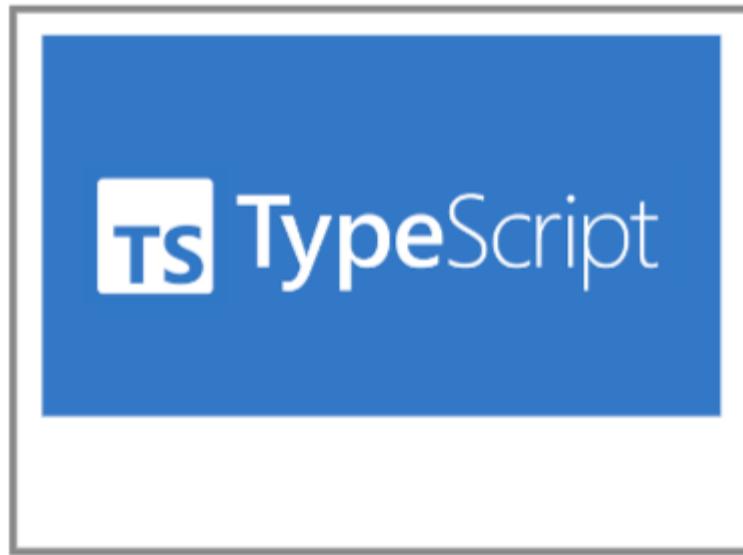
```
new TitleSlide("I am a title")
```

```
new TitleSlide("I am a title")
```



I am a title

```
new ImageSlide("/ts-logo.png")
```



ER 2.

## Guards in place

CHAPTER 3.

With **union** we stand

TYPE GUARDS

```
value is number {
  number';

  ...
}

[] = ['a', 1, 'b', 2];
es.filter(isNumber);
```

GNATURES

```
wn): asserts v is string {
  ...
  must be a string!');
}

input: unknown) {
  ...
  Case()); // ✓
```

CAREFULLY...

```
v is string {
  ...
  must be a string!');
```

CHAPTER 4.

Never say **never**

CHAPTER 5.

A dash of **flavoring**

CHAPTER

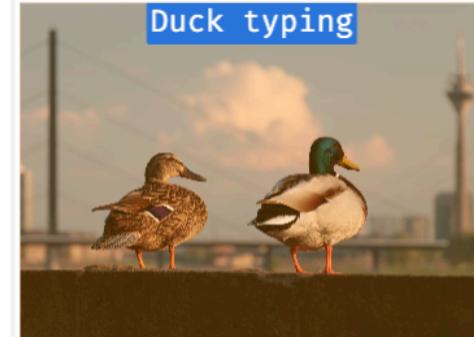
Don't **overload**  
problems

## Discriminated Union Type

- Improved Type Safety
- Enhanced Autocompletion

## 404 Documentation not found

## Duck typing



## Function Overloading

- Allow to write **type-safe** multiple signatures

// TODO: AUREL, VU  
QUE TU NE LIS PAS TES  
MAILS, JE TE METS UN  
RAPPEL ICI:

SLIDE GIF À FIXER 🙏

## FLAVORING VS BRANDING

```
1 type UserId = {
2   type: "UserId";
3 } & string;
4 const userId: UserId = "42"; ✓
5 type ValidUserId = {
6   type: "ValidUserId";
7 } & string;
8 const validUser: ValidUserId = "42"; ✗
```

## Function Overloading

- Allow to write **type-safe** multiple signatures
- Powerful feature in TypeScript

## BRANDING

```
1 function isValidId(id: Id | ValidId): id is ValidId {}
2 async function saveId(id: ValidId): Promise<void> {}
3 const id: Id = "test";
4 saveId(id); ✗
5 if (isValidId(id)) {
6   saveId(id); ✓
7 }
```

## Function Overloading

- Allow to write **type-safe** multiple signatures
- Powerful feature in TypeScript
- BUT Don't use it too much



Aurélien Loyer  
@AurelienLoyer  
Software Engineering Leader @QDNA  
Delphin Aubin  
@DelphinAubin  
JavaScript expert (@Zenika)

# Slide

CHAPTER 1.  
**Strictness** won't save you  
when the bullets fly

## Summary

**Strictness** WON'T SAVE YOU WHEN THE BULLETS FLY  
GET YOUR **Type Guards** IN PLACE  
WITH **union** WE STAND  
NEVER SAY **never**  
A DASH OF **flavoring**  
DON'T **overload** ME WITH YOUR PROBLEMS, MATE.  
A QUICK ZOOM ON **Template literal** TYPE  
THE **enum** CONUNDRUM  
**Infer** THIS MATE!

CHAPTER 2.  
Get your **Type Guards** in place

CHAPTER 3.  
With **union**

## typescript-strict-plugin

```
1 // tsconfig.json
2 {
3   "compilerOptions": {
4     ...
5     "typeScript": "typescript-strict-plugin"
6   }
7 }
```

## USER-DEFINED TYPE GUARDS

```
1 function isNumber(value: any): value is number {
2   return typeof value === 'number';
3 }
4
5 const arr = [1, 'two', true];
6 arr.filter(isNumber);
```

```
interface Square {
  kind: 'square';
  sides: number;
}
interface Rectangle {
  kind: 'rectangle';
  width: number;
  height: number;
}
type Shape = Square | Rectangle;
```

## asserts SIGNATURES

```
1 function assertString(v: unknown): asserts v is string {
2   if (!v) throw new Error(`Value ${v} must be a string!`);
3 }
4
5 const arr = [1, 'two', true];
6 arr.filter((v) => assertString(v));
```

```
function arrow(s: Shape) {
  // Error: Property 'kind' is missing in type 'string'.
}
```

# Slide

## JUST A SCRIPT AWAY!

```
{ "scripts": [
  ...
  "typescript": "tscc && tsc-strict", // Can be added to CI
]
}
```

# Slide





Aurélien Loyer  
@AurelienLoyer  
Software Engineering Leader @QMAA



Delphin Aubin  
@DelphinAubin  
Javascript expert @Zenika



# Chapitre

CHAPTER 1.  
**Strictness** won't save you  
when the bullets fly

[typescript-strict-plugin](#)

```
1 // tsconfig.json
2 {
3   "compilerOptions": {
4     ...
5     "strict": true,
6     "noImplicitAny": true,
7     "noImplicitThis": true,
8     "noUnusedLocals": true,
9     "noUnusedParameters": true,
10    "noFallthroughCasesInSwitch": true
11  }
12 }
```

JUST A SCRIPT AWAY!

```
{
  "scripts": [
    ...
    "tsc && tsc-strict", // Can be added to CI
  ]
}
```

## Summary

**Strictness** WON'T SAVE YOU WHEN THE BULLETS FLY  
GET YOUR **Type Guards** IN PLACE  
WITH **union** WE STAND  
NEVER SAY **never**  
A DASH OF **flavoring**  
DON'T **overload** ME WITH YOUR PROBLEMS, MATE.  
A QUICK ZOOM ON **Template literal** TYPE  
THE **enum** CONUNDRUM  
**Infer** THIS MATE!

# Chapitre

CHAPTER 2.  
Get your **Type Guards** in place

With **union**

[USER-DEFINED TYPE GUARDS](#)

```
1 function isNumber(value: any): value is number {
2   return typeof value === 'number';
3 }
4
5 type Square = {
6   kind: 'square';
7   sides: number;
8 }
9
10 interface Rectangle {
11   kind: 'rectangle';
12   width: number;
13   height: number;
14 }
15
16 type Shape = Square | Rectangle;
```

**asserts** SIGNATURES

```
1 function assertString(v: unknown): asserts v is string {
2   if (!typeof v === 'string') {
3     throw new Error(`Expected ${v} to be a string!`);
4   }
5 }
6
7 function validateEmail(email: unknown): asserts email is string {
8   if (!email.includes('@')) {
9     throw new Error(`Email ${email} is not valid!`);
10   }
11 }
```

```
interface Square {
  kind: "square";
  sides: number;
}
interface Rectangle {
  kind: "rectangle";
  width: number;
  height: number;
}
type Shape = Square | Rectangle;
```

```
function areEqual(s: Shape) {
  // Error: Property 'kind' is missing.
}
```



# TitleSlide

# TitleSlide



Aurélien Loyer  
@AurelienLoyer  
Software Engineering Leader @QMAA  
Delphin Aubin  
@DelphinAubin  
JavaScript expert @Zenika



CHAPTER 1.

## Strictness won't save you when the bullets fly

**Summary**

Strictness WON'T SAVE YOU WHEN THE BULLETS FLY  
GET YOUR **Type Guards** IN PLACE  
WITH **union** WE STAND  
NEVER SAY **never**  
A DASH OF **flavoring**  
DON'T **overload** ME WITH YOUR PROBLEMS, MATE.  
A QUICK ZOOM ON **Template literal** TYPE  
THE **enum** CONUNDRUM  
**Infer** THIS MATE!

CHAPTER 2.

## Get your **Type Guards** in place

With **union**

typescript-strict-plugin

```
1 // tsconfig.json
2 {
3   "compilerOptions": {
4     ...
5     "typecheck": "tscc && tsc-strict"
6   }
7 }
```

JUST A SCRIPT AWAY!

```
{
  "scripts": [
    ...
    "typecheck": "tsc && tscc-strict" // Can be added to CI
  ]
}
```

USER-DEFINED TYPE GUARDS

```
1 function isNumber(value: any): value is number {
2   return typeof value === "number";
3 }
```

```
1 type Square = {
2   kind: "square";
3   sides: number;
4 };
5
6 interface Rectangle {
7   kind: "rectangle";
8   width: number;
9   height: number;
10 }
```

asserts SIGNATURES

```
1 function assertStr(v: unknown): asserts v is string {
2   if (!v || !v.toString) {
3     throw new Error(`String ${v} must be a string!`);
4   }
5 }
```

```
1 function foo(a: string): asserts a is string {
2   assertString(a);
3 }
```

```
interface Square {
  kind: "square";
  sides: number;
}

interface Rectangle {
  kind: "rectangle";
  width: number;
  height: number;
}

type Shape = Square | Rectangle;
```



# Summary



Aurélien Loyer  
@AurelienLoyer  
Software Engineering Leader @QMAA  
Delphin Aubin  
@DelphinAubin  
JavaScript expert @Zenika



CHAPTER 1.  
**Strictness** won't save you when the bullets fly

**Summary**

**Strictness** WON'T SAVE YOU WHEN THE BULLETS FLY  
GET YOUR **Type Guards** IN PLACE  
WITH **union** WE STAND  
NEVER SAY **never**  
A DASH OF **flavoring**  
DON'T **overload** ME WITH YOUR PROBLEMS, MATE.  
A QUICK ZOOM ON **Template literal** TYPE  
THE **Enum** CONUNDRUM  
Infer THIS MATE!

CHAPTER 2.  
Get your **Type Guards** in place

CHAPTER 3.  
With **union**

typescript-strict-plugin

```
1 // tsconfig.json
2 {
3   "compilerOptions": {
4     ...
5     "typecheck": "tscc && tsc-strict"
6   }
7 }
```

USER-DEFINED TYPE GUARDS

```
1 function isNumber(value: any): value is number {
2   return typeof value === "number";
3 }
4
5 type Square = {
6   kind: "square";
7   sides: number;
8 };
9
10 type Rectangle = {
11   kind: "rectangle";
12   width: number;
13   height: number;
14 };
15
16 type Shape = Square | Rectangle;
```

```
interface Square {
  kind: "square";
  sides: number;
}
interface Rectangle {
  kind: "rectangle";
  width: number;
  height: number;
}
type Shape = Square | Rectangle;
```

JUST A SCRIPT AWAY!

```
{
  "scripts": [
    ...
    "typecheck": "tscc && tsc-strict" // Can be added to CI
  ]
}
```

asserts SIGNATURES

```
1 function assertStr(v: unknown): asserts v is string {
2   if (!v || !v || typeof v !== "string") {
3     throw new Error(`Value ${v} must be a string!`);
4   }
5 }
6
7 function foo(): asserts typeof this === "object" {
8   assertStr(this);
9 }
10
11 function bar(): asserts typeof this === "function" {
12   assertStr(this);
13 }
```

```
function arrow(s: Shape) {
  // Error: Property 's' is missing.
}
```





# Delphin Aubin

**@DelphinAubin**

Javascript expert @Zenika



# Aurélien Loyer

**@AurelienLoyer**

Software Engineering Leader @QIMA

## CHAPTER 1.

Strictness won't save you  
when the bullets fly

# typescript-strict-plugin

```
1 /* tsconfig.json */
2 {
3   "compilerOptions": {
4     ...
5     "strict": false,
6     "plugins": [
7       {
8         "name": "typescript-strict-plugin"
9       }
10    ]
11  }
12 }
```

# typescript-strict-plugin

```
1 /* tsconfig.json */
2 {
3   "compilerOptions": {
4     ...
5     "strict": false,
6     "plugins": [
7       {
8         "name": "typescript-strict-plugin"
9       }
10    ]
11  }
12 }
```

# typescript-strict-plugin

```
1 /* tsconfig.json */
2 {
3   "compilerOptions": {
4     ...
5     "strict": false,
6     "plugins": [
7       {
8         "name": "typescript-strict-plugin"
9       }
10    ]
11  }
12 }
```

## JUST A SCRIPT AWAY!

```
{  
  "scripts": {  
    ...,  
    "typecheck": "tsc && tsc-strict", // Can be added to CI  
  },  
}
```

# TypeScript Strict Plugin



Progressive strictness



No more excuses not to be strict



Easy detection of files with strict error

# Summary

Strictness WON'T SAVE YOU WHEN THE BULLETS FLY

GET YOUR Type Guards IN PLACE

WITH union WE STAND

NEVER SAY never

A DASH OF flavoring

DON'T overload ME WITH YOUR PROBLEMS, MATE.

A QUICK ZOOM ON template literal TYPE

## CHAPTER 2.

Get your Type Guards in place

## is USER-DEFINED TYPE GUARDS

```
function isNumber(value: any): value is number {
    return typeof value === 'number';
}

// ✨ Useful for if condition
if (isNumber(value)) {
    // TS knows value is a number
}

// ✨ Or for filter condition
const values: (string | number)[] = ['a', 1, 'b', 2];
// 🤔 number[]
const numbers: number[] = values.filter(isNumber);
```

## asserts SIGNATURES

```
1 function assertStr(v: unknown): asserts v is string {  
2     if (typeof v !== "string") {  
3         throw new Error(`#${v} must be a string!`);  
4     }  
5 }  
6  
7 function logUpperCaseValue(input: unknown) {  
8     assertStr(input); // ✅  
9  
10    // input's type is just 'string' here  
11    console.log(input.toUpperCase()); // ✅  
12 }
```

## asserts SIGNATURES

```
1 function assertStr(v: unknown): asserts v is string {
2   if (typeof v !== "string") {
3     throw new Error(`#${v} must be a string!`);
4   }
5 }
6
7 function logUpperCaseValue(input: unknown) {
8   assertStr(input); // 🙏
9
10 // input's type is just 'string' here
11 console.log(input.toUpperCase()); // ✅
12 }
```

## BUT TO USE CAREFULLY...

```
1 function assertStr(v): asserts v is string {  
2     if (typeof v !== 'string') {  
3         throw new Error('Input must be a string!');  
4     }  
5 }
```

## BUT TO USE CAREFULLY...

```
1 function assertStr(v): asserts v is User[] { // 😰
2     if (typeof v !== 'string') {
3         throw new Error('Input must be a string!');
4     }
5 }
```

## BUT TO USE CAREFULLY...

```
function assertStr(v): asserts v is User[] { // 😰
    if (typeof v !== 'string') {
        throw new Error('Input must be a string!');
    }
}
```



# Type Guards

- 👮 Ensure type safety in your code
- 🤖 Enhance autocompletion
- ⚠️ Prevent runtime errors
- 🔥 To use with care

## CHAPTER 3.

With union we stand

# Discriminated Union Type



Improved Type Safety



Enhanced Autocompletion



## CHAPTER 4.

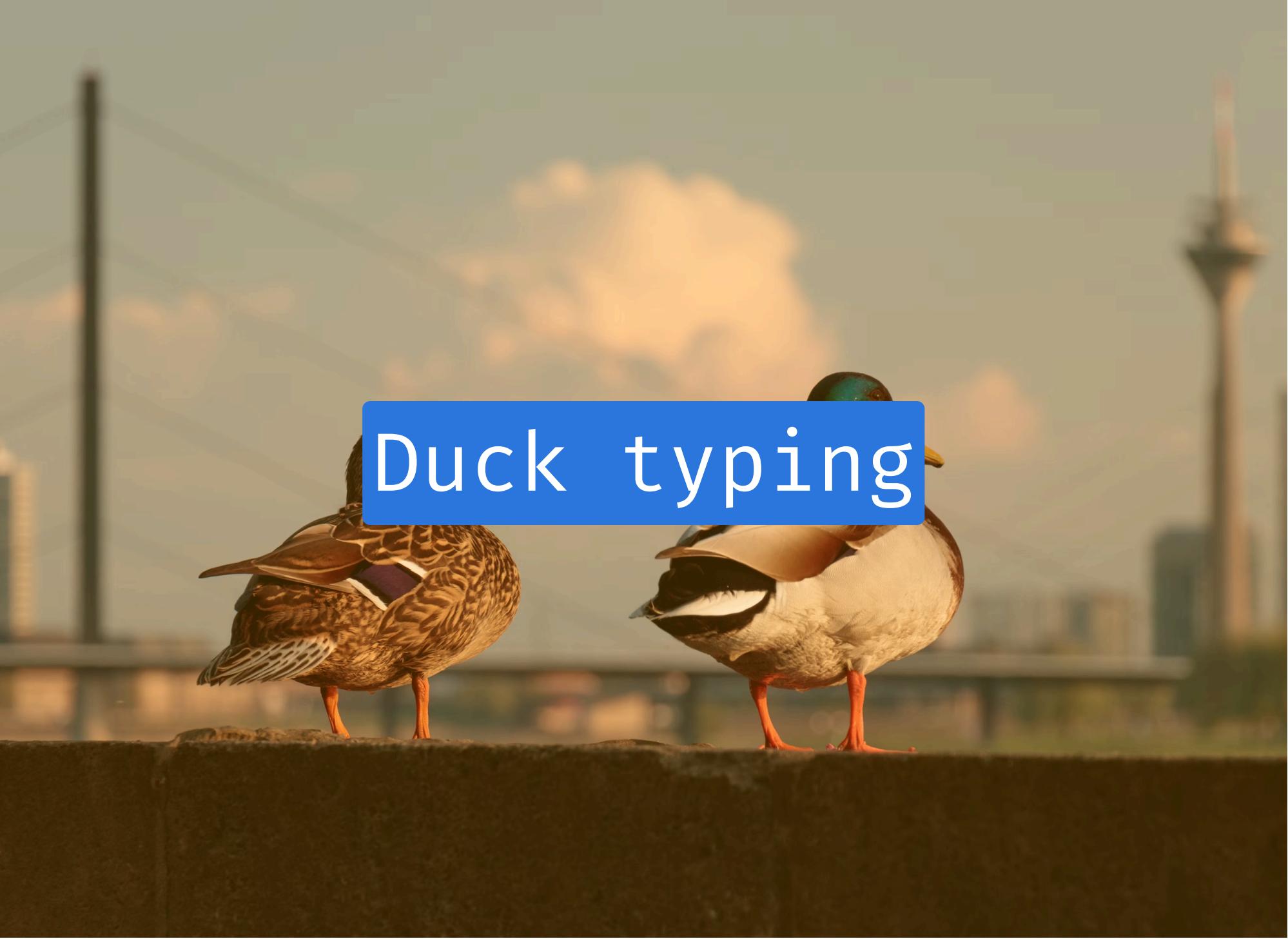
Never say never

# Never

Go to documentation 

## CHAPTER 5.

A dash of flavoring

A photograph of two ducks standing on a dark, textured ledge. The duck on the left is facing away from the camera, showing its brown and white patterned back. The duck on the right is facing towards the camera, showing its white breast and brown back. In the center of the image, there is a solid blue rectangular overlay containing the text "Duck typing" in a white, sans-serif font.

Duck typing

# FLAVORING VS BRANDING

```
1 type UserId = {  
2   __type?: "UserId";  
3 } & string;  
4  
5 const userId: UserId = "42"; // ✓  
6  
7 type ValidUserId = {  
8   __type: "ValidUserId";  
9 } & string;  
10  
11 const validUser: ValidUserId = "42"; // ✗
```

# FLAVORING VS BRANDING

```
1 type UserId = {  
2   __type?: "UserId";  
3 } & string;  
4  
5 const userId: UserId = "42"; // ✓  
6  
7 type ValidUserId = {  
8   __type: "ValidUserId";  
9 } & string;  
10  
11 const validUser: ValidUserId = "42"; // ✗
```

# FLAVORING VS BRANDING

```
1 type UserId = {  
2   __type?: "UserId";  
3 } & string;  
4  
5 const userId: UserId = "42"; // ✓  
6  
7 type ValidUserId = {  
8   __type: "ValidUserId";  
9 } & string;  
10  
11 const validUser: ValidUserId = "42"; // ✗
```

# FLAVORING VS BRANDING

```
1 type UserId = {  
2   __type?: "UserId";  
3 } & string;  
4  
5 const userId: UserId = "42"; // ✓  
6  
7 type ValidUserId = {  
8   __type: "ValidUserId";  
9 } & string;  
10  
11 const validUser: ValidUserId = "42"; // ✗
```

# FLAVORING VS BRANDING

```
1 type UserId = {  
2   __type?: "UserId";  
3 } & string;  
4  
5 const userId: UserId = "42"; // ✓  
6  
7 type ValidUserId = {  
8   __type: "ValidUserId";  
9 } & string;  
10  
11 const validUser: ValidUserId = "42"; // ✗
```

# FLAVORING VS BRANDING

```
1 type UserId = {  
2   __type?: "UserId";  
3 } & string;  
4  
5 const userId: UserId = "42"; // ✓  
6  
7 type ValidUserId = {  
8   __type: "ValidUserId";  
9 } & string;  
10  
11 const validUser: ValidUserId = "42"; // ✗
```

# BRANDING

```
1 function isValid(id: Id | ValidId): id is ValidId {}  
2  
3 async function saveId(id: ValidId): Promise<void> {}  
4  
5 const id: Id = "test";  
6  
7 saveId(id); // ✗  
8  
9 if (isValid(id)) {  
10   saveId(id); // ✓  
11 }
```

# BRANDING

```
1 function isValidId(id: Id | ValidId): id is ValidId {}
2
3 async function saveId(id: ValidId): Promise<void> {}
4
5 const id: Id = "test";
6
7 saveId(id); // ✗
8
9 if (isValidId(id)) {
10   saveId(id); // ✓
11 }
```

# BRANDING

```
1 function isValidId(id: Id | ValidId): id is ValidId {}
2
3 async function saveId(id: ValidId): Promise<void> {}
4
5 const id: Id = "test";
6
7 saveId(id); // ✗
8
9 if (isValidId(id)) {
10   saveId(id); // ✓
11 }
```

# BRANDING

```
1 function isValid(id: Id | ValidId): id is ValidId {}  
2  
3 async function saveId(id: ValidId): Promise<void> {}  
4  
5 const id: Id = "test";  
6  
7 saveId(id); // ✗  
8  
9 if (isValid(id)) {  
10   saveId(id); // ✓  
11 }
```

# BRANDING

```
1 function isValid(id: Id | ValidId): id is ValidId {}  
2  
3 async function saveId(id: ValidId): Promise<void> {}  
4  
5 const id: Id = "test";  
6  
7 saveId(id); // ✗  
8  
9 if (isValid(id)) {  
10   saveId(id); // ✓  
11 }
```

# Flavoring

[Go to documentation !\[\]\(0dc690309bcc577bffe1c73810af57c4\_img.jpg\)](#)

## CHAPTER 6.

Don't overload me with your  
problems, mate.

# Function Overloading

✍️ Allow to write *type-safe functions* with multiple signatures

# Function Overloading

- ✍️ Allow to write *type-safe functions* with multiple signatures
- 🚀 *Powerful* feature in TypeScript

# Function Overloading

- ✍️ Allow to write *type-safe functions* with multiple signatures
- 🚀 Powerful feature in TypeScript
- ⚠️ **BUT** Don't use it too much

# Function Overloading

- ✍️ Allow to write *type-safe functions* with multiple signatures
- 🚀 Powerful feature in TypeScript
- ⚠️ BUT Don't use it too much, it can be *HARD* to read

# Overload

[Go to documentation !\[\]\(cb4f0a20d825bd465eb05abcc7870399\_img.jpg\)](#)

## CHAPTER 7.

A quick zoom on template  
literal type

## LET'S VALIDATE EMAILS

```
1 type Ext = "com" | "fr" | "net";
2 type Email = `${string}@${string}.${Ext}`;
3
4
5 const valid: Email = 'john.doe@gmail.com' // ✓
6 const ko1: Email = 'john.doegmail.com' // ✗
7 const ko2: Email = 'john.doe@gmail.BAD' // ✗
```

## LET'S VALIDATE EMAILS

```
1 type Ext = "com" | "fr" | "net";
2 type Email = `${string}@${string}.${Ext}`;
3
4
5 const valid: Email = 'john.doe@gmail.com' // ✓
6 const ko1: Email = 'john.doegmail.com' // ✗
7 const ko2: Email = 'john.doe@gmail.BAD' // ✗
```

## LET'S VALIDATE EMAILS

```
1 type Ext = "com" | "fr" | "net";
2 type Email = `${string}@${string}.${Ext}`;
3
4
5 const valid: Email = 'john.doe@gmail.com' // ✓
6 const ko1: Email = 'john.doegmail.com' // ✗
7 const ko2: Email = 'john.doe@gmail.BAD' // ✗
```

## LET'S VALIDATE EMAILS

```
1 type Ext = "com" | "fr" | "net";
2 type Email = `${string}@${string}.${Ext}`;
3
4
5 const valid: Email = 'john.doe@gmail.com' // ✓
6 const ko1: Email = 'john.doegmail.com' // ✗
7 const ko2: Email = 'john.doe@gmail.BAD' // ✗
```

## CHAPTER 81.

Don't use enums

...

## CHAPTER 98.

Infer this mate!

# THANK YOU





@AurelienLoyer

@DelphinAubin

# Links

- [The Lies We Tell Ourselves Using TypeScript](#)
- [TypeScript Survival Guide: Life-Saving Tips and Techniques](#)
- [Don't use Enums in Typescript, they are very dangerous](#) 😞

## CHAPTER 81.

The `enum` conundrum

```
1 enum Countries {  
2     FR,  
3     ES  
4 }  
5  
6 function loadCountry(country: Countries) {  
7     // do something with country  
8     // because I use the country enum I'm sure it exists  
9     // Sure about that ? 🤷  
10 }  
11  
12 let magicNumber: number = 42  
13  
14 loadCountry(magicNumber) // 🤷 It's ok 🤷;
```

```
1 enum Countries {
2     FR,
3     ES
4 }
5
6 function loadCountry(country: Countries) {
7     // do something with country
8     // because I use the country enum I'm sure it exists
9     // Sure about that ? 🤔
10 }
11
12 let magicNumber: number = 42
13
14 loadCountry(magicNumber) // 🤔 It's ok 🤔;
```

```
1 enum Countries {  
2     FR,  
3     ES  
4 }  
5  
6 function loadCountry(country: Countries) {  
7     // do something with country  
8     // because I use the country enum I'm sure it exists  
9     // Sure about that ? 🤷  
10 }  
11  
12 let magicNumber: number = 42  
13  
14 loadCountry(magicNumber) // 🤷 It's ok 🤷;
```

```
1 enum Countries {
2     FR,
3     ES
4 }
5
6 function loadCountry(country: Countries) {
7     // do something with country
8     // because I use the country enum I'm sure it exists
9     // Sure about that ? 🤔
10 }
11
12 let magicNumber: number = 42
13
14 loadCountry(magicNumber) // 🤪 It's ok 🤪;
```

```
1 enum Countries {
2     FR,
3     ES
4 }
5
6 function loadCountry(country: Countries) {
7     // do something with country
8     // because I use the country enum I'm sure it exists
9     // Sure about that ? 🤔
10 }
11
12 let magicNumber: number = 42
13
14 loadCountry(magicNumber) // 🤪 It's ok 🤪;
```

```
1 enum Countries {
2     FR,
3     ES
4 }
5
6 function loadCountry(country: Countries) {
7     // do something with country
8     // because I use the country enum I'm sure it exists
9     // Sure about that ? 🤔
10 }
11
12 let magicNumber: number = 42
13
14 loadCountry(magicNumber) // 😱 It's ok 😱;
```

```
1 enum Countries {
2     FR = 0,
3     ES = 1
4 }
5
6 function loadCountry(country: Countries) {
7     // do something with country
8     // because I use the country enum I'm sure it exists
9     // Sure about that ? 🤔
10 }
11
12 let magicNumber: number = 42
13
14 loadCountry(magicNumber) // 😱 It's ok 😱;
```

enum

```
enum Country {  
    FR = "FR",  
    ES = "ES",  
}  
const myCountry = Country.ES;
```

# enum

```
enum Country {  
    FR = "FR",  
    ES = "ES",  
}  
const myCountry = Country.ES;
```

```
var Country;  
(function (Country) {  
    Country["FR"] = "FR";  
    Country["ES"] = "ES";  
})(Country || (Country = {}));  
const myCountry = Country.ES;
```

## const enum

```
const enum Country {  
    FR = "FR",  
    ES = "ES",  
}  
const myCountry = Country.ES;
```

## const enum

```
const enum Country {  
    FR = "FR",  
    ES = "ES",  
}  
const myCountry = Country.ES;
```

```
const myCountry = "ES";
```

OR JUST AN union type

```
type Country = "ES" | "FR";  
  
const myCountry: Country = "ES";
```

```
const myCountry = "ES";
```

## OR AN Object

```
const Countries = {  
  ES: "ES",  
  FR: "FR",  
} as const;  
  
const myCountry = Countries.ES;
```

## OR AN Object

```
const Countries = {  
  ES: "ES",  
  FR: "FR",  
} as const;  
  
const myCountry = Countries.ES;
```

```
const Countries = {  
  ES: "ES",  
  FR: "FR",  
};  
const myCountry = Countries.ES;
```

## WORK WITH Objects

```
const countries = {  
  ES: "ES",  
  FR: "FR",  
} as const;  
  
type Countries = typeof countries;  
type Country = Countries[keyof Countries];  
// "Spain" | "France"  
  
function loadCountry(country: Country) {}  
  
loadCountry("Timbuktu"); // ✗  
loadCountry("Spain"); // ✓  
loadCountry(countries.ES); // ✓
```