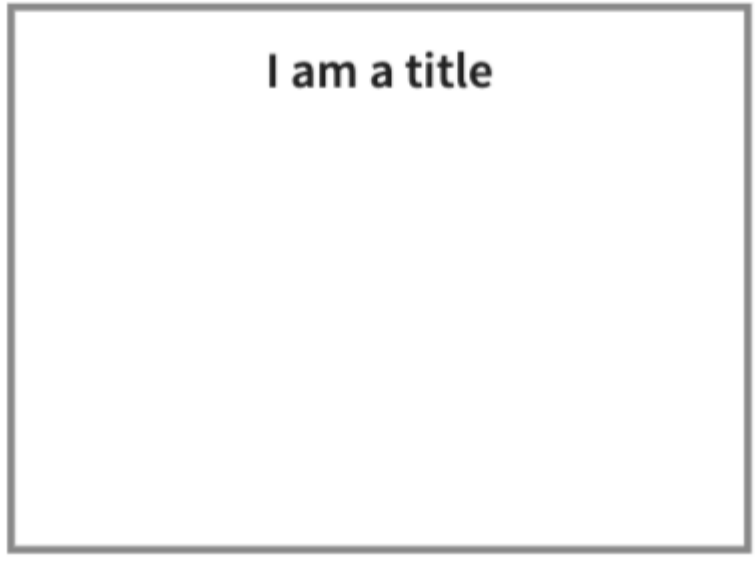

TS
The 20 most painful
profitable typescript
minutes of your life!

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

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



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

ER 2.

Guards in place

CHAPTER 3.

With union we stand

CHAPTER 4.

Never say never

CHAPTER 5.

A dash of flavoring

CHAPTER

Don't overload problems

D TYPE GUARDS

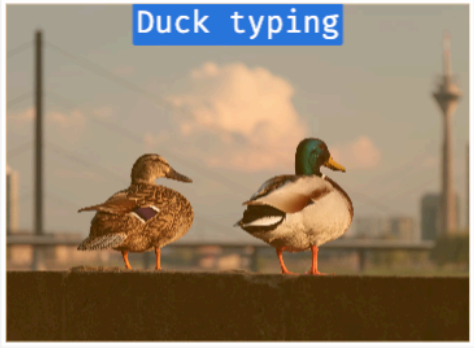
```
 value is number {
umber';


ber


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

Discriminated Union Type

👮Improved Type Safety
🤖Enhanced Autocompletion

404 Documentation not found

Duck typing



Function Ov

✍️Allow to write type-sa
multiple signatures

GNATURES

```
wn): asserts v is string {
) {
ust be a string!`);


input: unknown) {


string' here
Case()); // ✅
```

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

SLIDE GIF À FIXER 🙏

FLAVORING VS BRANDING

```
type UserId = {
  __type?: "UserId";
} & string;

const userId: UserId = "42"; ✅

type ValidUserId = {
  __type: "ValidUserId";
} & string;

const validUser: ValidUserId = "42"; ❌
```

Function Ov

✍️Allow to write type-sa
multiple signatures
🚀Powerful feature in Ty

AREFULLY...

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

BRANDING

```
function isIdValid(id: Id | ValidId): id is ValidId {}

async function saveId(id: ValidId): Promise<void> {}

const id: Id = "test";

saveId(id); ❌

if (isIdValid(id)) {
  saveId(id); ✅
}
```

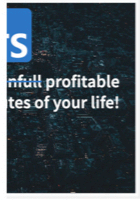
Function Ov

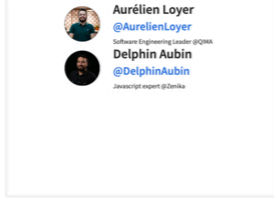
✍️Allow to write type-sa
multiple signatures
🚀Powerful feature in Ty
⚠️BUT Don't use it too m

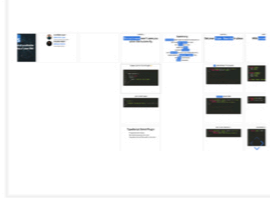

Slide
Slide
Slide
Aurélien Loyer
@AurelienLoyer
Software Engineering Leader @QIMA
Delphin Aubin
@DelphinAubin
Javascript expert @Zenika
CHAPTER 1.
Strictness won't save you when the bullets fly
Summary
Strictness WON'T SAVE YOU WHEN THE BULLETS FLY
GET YOUR Type Guards IN PLACE
WITH union WE STAND
NEVER 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
typescript-strict-plugin🙌
JUST A SCRIPT AWAY!
is USER-DEFINED TYPE GUARDS
asserts SIGNATURES

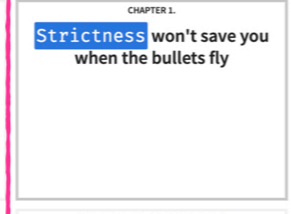
Chapitre

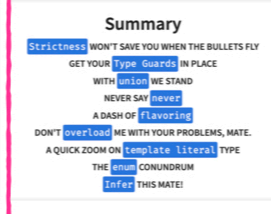
Chapitre

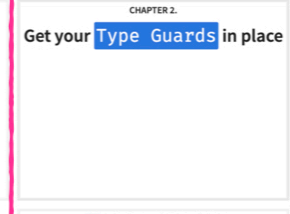


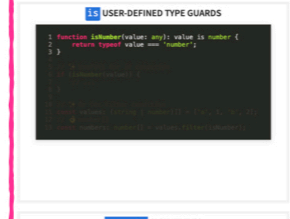


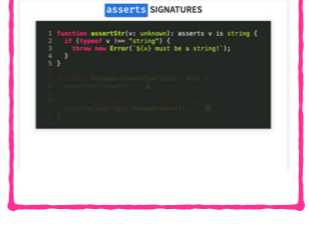


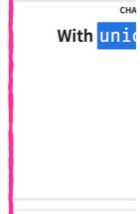


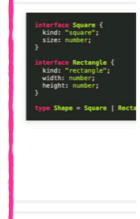








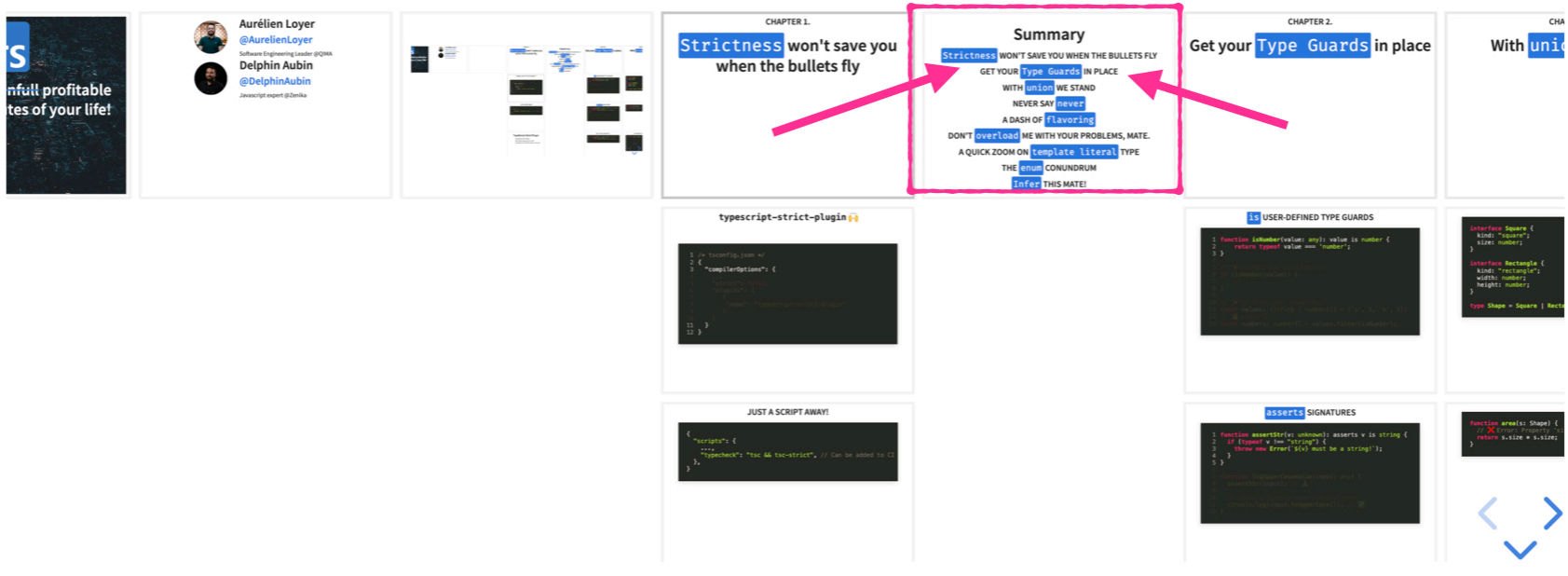




TitleSlide

TitleSlide

Summary


nfull profitable
tes of your life!
Aurélien Loyer
@AurelienLoyer
Software Engineering Leader @QIMA
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
CHA
With unio
typescript-strict-plugin
JUST A SCRIPT AWAY!
is USER-DEFINED TYPE GUARDS
asserts SIGNATURES

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

```jsonc
/* tsconfig.json */
{
  "compilerOptions": {
    ...
    "strict": false,
    "plugins": [
      {
        "name": "typescript-strict-plugin"
      }
    ]
  }
}
```

typescript-strict-plugin

```jsonc
/* tsconfig.json */
{
  "compilerOptions": {
    ...
    "strict": false,
    "plugins": [
      {
        "name": "typescript-strict-plugin"
      }
    ]
  }
}
```

typescript-strict-plugin

```
/* tsconfig.json */
{
  "compilerOptions": {
    ...
    "strict": false,
    "plugins": [
       {
        "name": "typescript-strict-plugin"
       }
    ]
  }
}
```

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

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

// ✨ Usefull 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

```ts
function assertStr(v: unknown): asserts v is string {
  if (typeof v !== "string") {
    throw new Error(`${v} must be a string!`);
  }
}

function logUpperCaseValue(input: unknown) {
  assertStr(input); // 🙏

  // input's type is just 'string' here
  console.log(input.toUpperCase()); // ✅
}
```

`asserts` SIGNATURES

```ts
function assertStr(v: unknown): asserts v is string {
  if (typeof v !== "string") {
    throw new Error(`${v} must be a string!`);
  }
}

function logUpperCaseValue(input: unknown) {
  assertStr(input); // 🙏

  // input's type is just 'string' here
  console.log(input.toUpperCase()); // ✅
}
```

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.

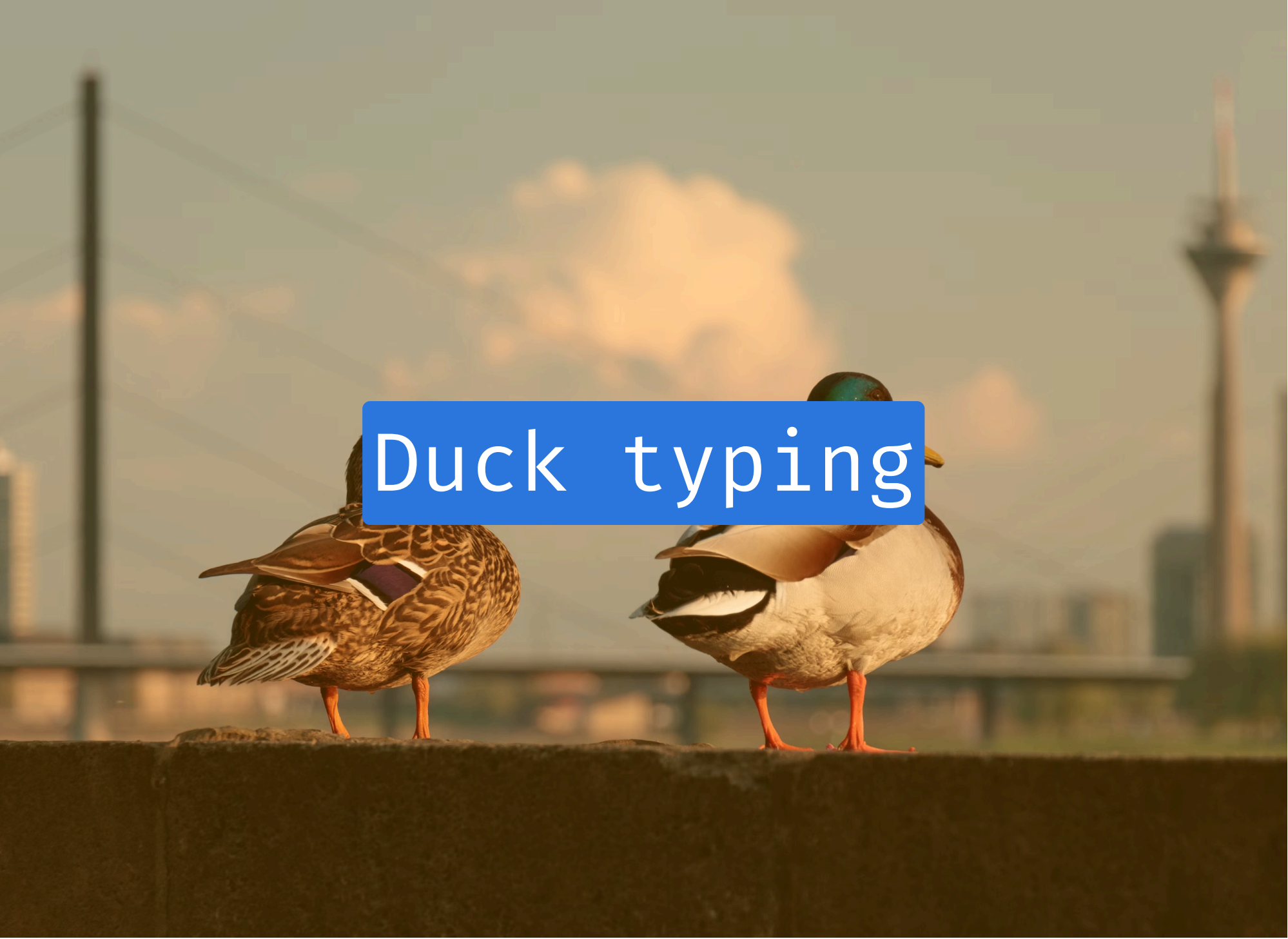
Never say `never`

Never

Go to documentation

CHAPTER 5.

A dash of `flavoring`


Duck typing

FLAVORING VS BRANDING

```ts
type UserId = {
  __type?: "UserId";
} & string;

const userId: UserId = "42"; // ✅

type ValidUserId = {
  __type: "ValidUserId";
} & string;

const validUser: ValidUserId = "42"; // ❌
```

FLAVORING VS BRANDING

```ts
type UserId = {
  __type?: "UserId";
} & string;

const userId: UserId = "42"; // ✅

type ValidUserId = {
  __type: "ValidUserId";
} & string;

const validUser: ValidUserId = "42"; // ❌
```

FLAVORING VS BRANDING

```ts
type UserId = {
  __type?: "UserId";
} & string;

const userId: UserId = "42"; // ✅

type ValidUserId = {
  __type: "ValidUserId";
} & string;

const validUser: ValidUserId = "42"; // ❌
```

FLAVORING VS BRANDING

```ts
type UserId = {
  __type?: "UserId";
} & string;

const userId: UserId = "42"; // ✅

type ValidUserId = {
  __type: "ValidUserId";
} & string;

const validUser: ValidUserId = "42"; // ❌
```

FLAVORING VS BRANDING

```ts
type UserId = {
  __type?: "UserId";
} & string;

const userId: UserId = "42"; // ✅

type ValidUserId = {
  __type: "ValidUserId";
} & string;

const validUser: ValidUserId = "42"; // ❌
```

FLAVORING VS BRANDING

```ts
type UserId = {
  __type?: "UserId";
} & string;

const userId: UserId = "42"; // ✅

type ValidUserId = {
  __type: "ValidUserId";
} & string;

const validUser: ValidUserId = "42"; // ❌
```

BRANDING

```ts
function isIdValid(id: Id | ValidId): id is ValidId {}

async function saveId(id: ValidId): Promise<void> {}

const id: Id = "test";

saveId(id); // ❌

if (isIdValid(id)) {
  saveId(id); // ✅
}
```

BRANDING

```ts
function isIdValid(id: Id | ValidId): id is ValidId {}

async function saveId(id: ValidId): Promise<void> {}

const id: Id = "test";

saveId(id); // ❌

if (isIdValid(id)) {
  saveId(id); // ✅
}
```

BRANDING

```ts
function isIdValid(id: Id | ValidId): id is ValidId {}

async function saveId(id: ValidId): Promise<void> {}

const id: Id = "test";

saveId(id); // ❌

if (isIdValid(id)) {
  saveId(id); // ✅
}
```

BRANDING

```ts
function isIdValid(id: Id | ValidId): id is ValidId {}

async function saveId(id: ValidId): Promise<void> {}

const id: Id = "test";

saveId(id); // ❌

if (isIdValid(id)) {
  saveId(id); // ✅
}
```

BRANDING

```ts
function isIdValid(id: Id | ValidId): id is ValidId {}

async function saveId(id: ValidId): Promise<void> {}

const id: Id = "test";

saveId(id); // ❌

if (isIdValid(id)) {
  saveId(id); // ✅
}
```

Flavoring

Go to documentation

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

CHAPTER 7.

A quick zoom on `template literal` type

LET'S VALIDATE EMAILS

```ts
type Ext = "com" | "fr" | "net";
type Email = `${string}@${string}.${Ext}`;


const valid: Email = 'john.doe@gmail.com' // ✅
const ko1: Email = 'john.doegmail.com' // ❌
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

```ts
type Ext = "com" | "fr" | "net";
type Email = `${string}@${string}.${Ext}`;


const valid: Email = 'john.doe@gmail.com' // ✅
const ko1: Email = 'john.doegmail.com' // ❌
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


enum Countries {
FR,
ES
}
function loadCountry(country: Countries) {
// do something with country
// because I use the country enum I'm sure it exists
// Sure about that ? 🤔
}
let magicNumber: number = 42
loadCountry(magicNumber) // 😱 It's ok 😱;

```ts
enum Countries {
  FR,
  ES
}

function loadCountry(country: Countries) {
  // do something with country
  // because I use the country enum I'm sure it exists
  // Sure about that ? 🤔
}

let magicNumber: number = 42

loadCountry(magicNumber) // 😱 It's ok 😱;
```

```ts
enum Countries {
  FR,
  ES
}

function loadCountry(country: Countries) {
  // do something with country
  // because I use the country enum I'm sure it exists
  // Sure about that ? 🤔
}

let magicNumber: number = 42

loadCountry(magicNumber) // 😱 It's ok 😱;
```

```ts
enum Countries {
  FR,
  ES
}

function loadCountry(country: Countries) {
  // do something with country
  // because I use the country enum I'm sure it exists
  // Sure about that ? 🤔
}

let magicNumber: number = 42

loadCountry(magicNumber) // 😱 It's ok 😱;
```

```ts
enum Countries {
  FR,
  ES
}

function loadCountry(country: Countries) {
  // do something with country
  // because I use the country enum I'm sure it exists
  // Sure about that ? 🤔
}

let magicNumber: number = 42

loadCountry(magicNumber) // 😱 It's ok 😱;
```

```typescript
enum Countries {
  FR,
  ES
}

function loadCountry(country: Countries) {
  // do something with country
  // because I use the country enum I'm sure it exists
  // Sure about that ? 🤔
}

let magicNumber: number = 42

loadCountry(magicNumber) // 😱 It's ok 😱;
```

```typescript
enum Countries {
  FR = 0,
  ES = 1
}

function loadCountry(country: Countries) {
  // do something with country
  // because I use the country enum I'm sure it exists
  // Sure about that ? 🤔
}

let magicNumber: number = 42

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

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

const enum

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

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

OR JUST AN `union type`

```ts
type Country = "ES" | "FR";

const myCountry: Country = "ES";
```

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

OR AN `Object`

```ts
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`

```ts
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); // ✅
```