

TypeScript

JavaScript that scales.

TypeScript is a typed superset of JavaScript that compiles to plain JavaScript.

Any browser. Any host. Any OS. Open source.



Thierry Chatel

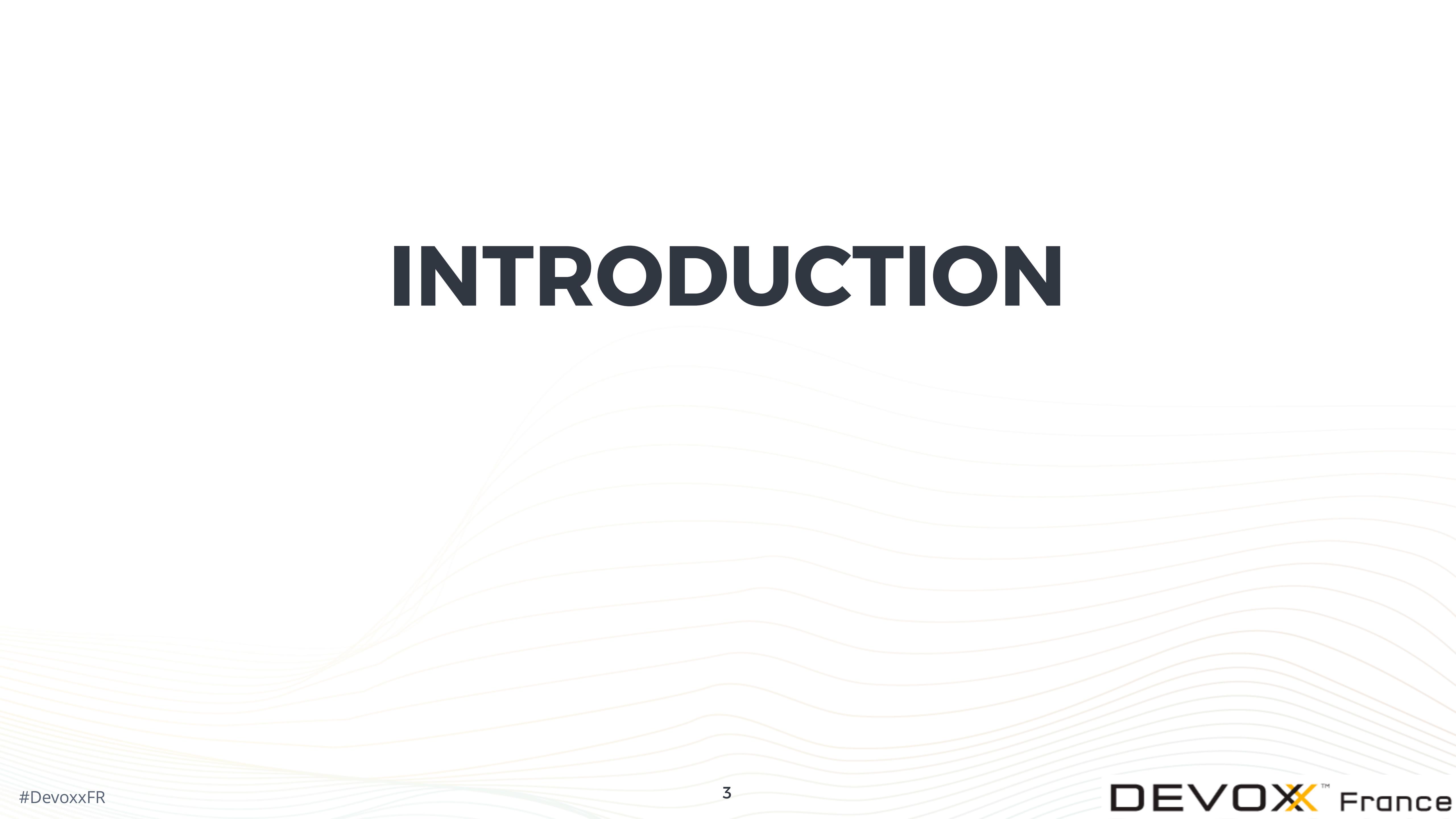
Metho**TIC** Conseil

-  tchatel@methotic.com
-  [@ThierryChatel](https://twitter.com/@ThierryChatel)

- consultant
- formateur
- GDE Angular



INTRODUCTION



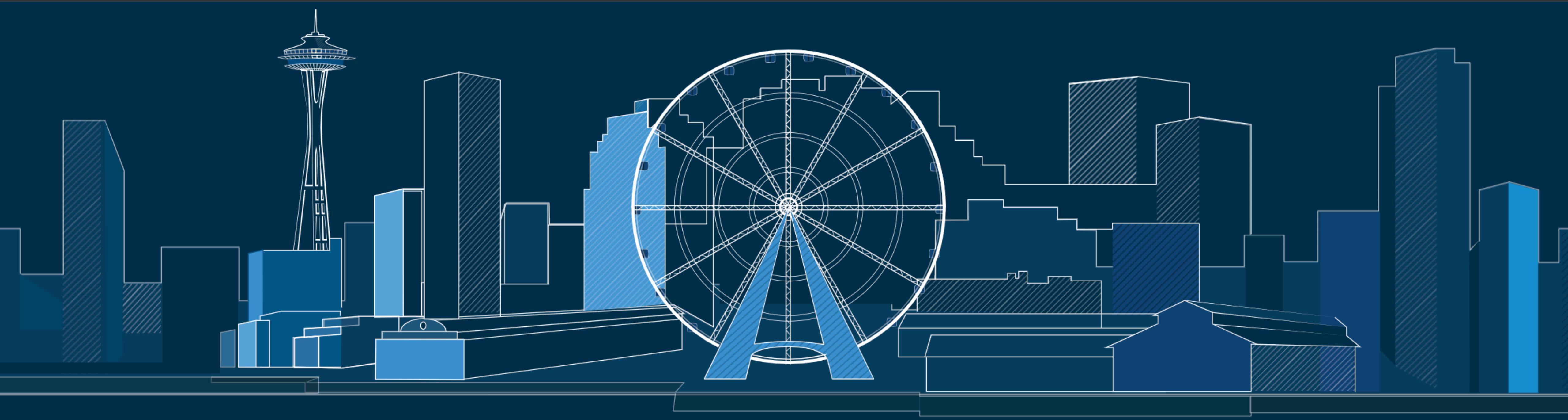


DEVOXX™
France 2015



DEVOXX
FRANCE

Comprendre enfin JavaScript



TypeScript

JavaScript that scales.

TypeScript is a typed superset of JavaScript that compiles to plain JavaScript.

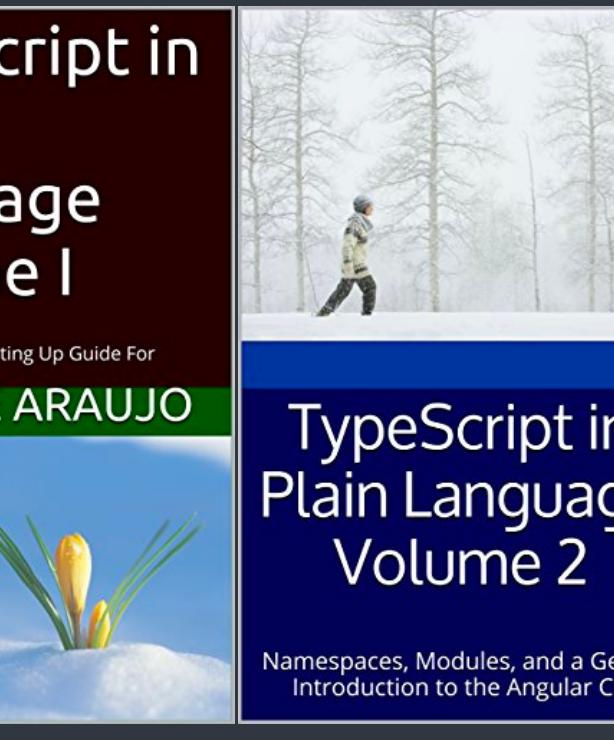
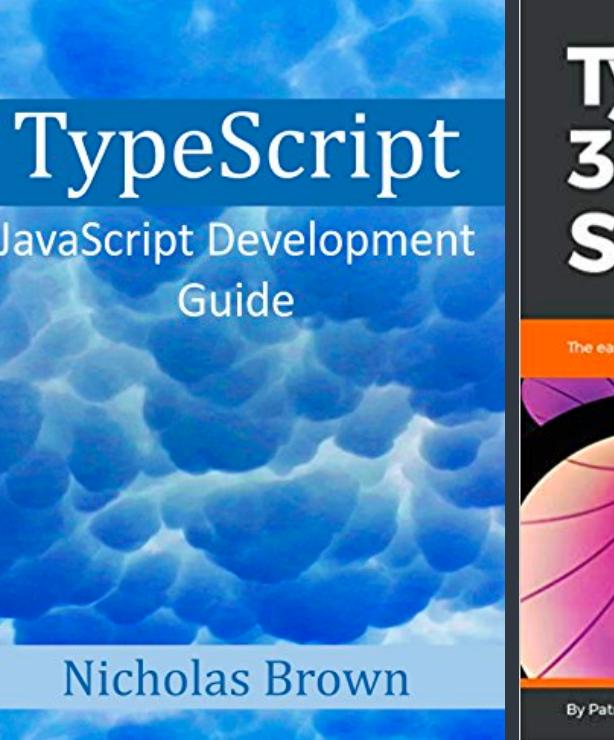
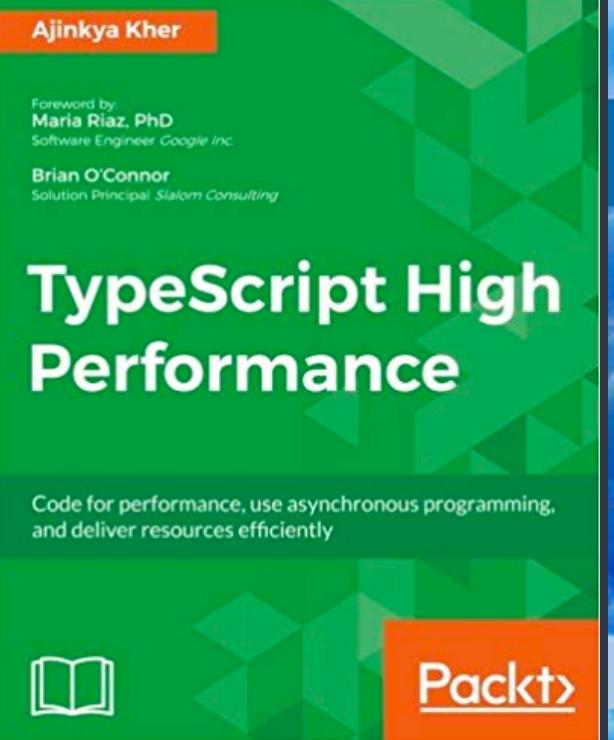
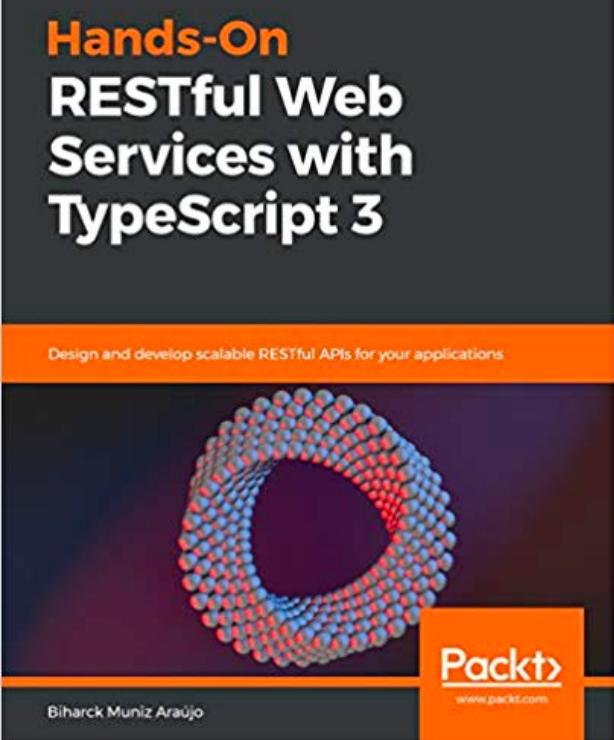
Any browser. Any host. Any OS. Open source.

TypeScript

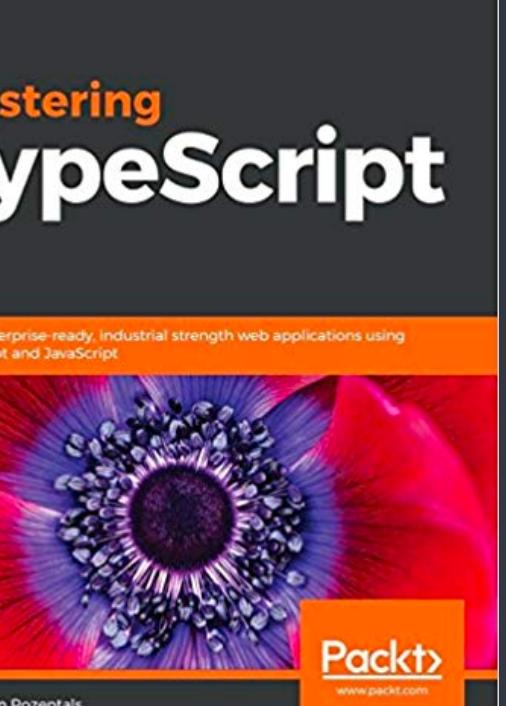
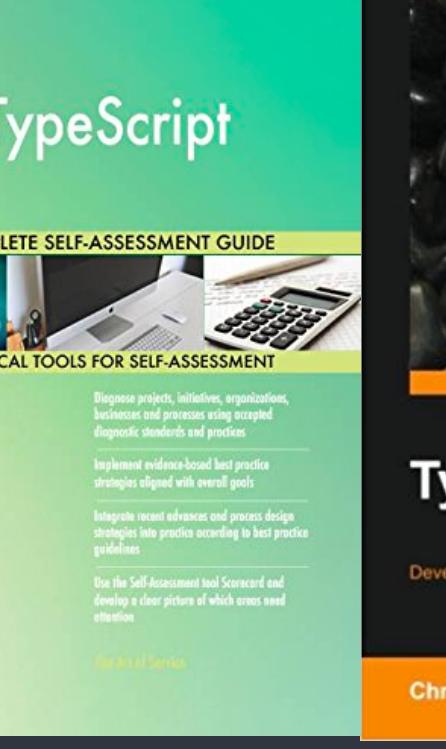
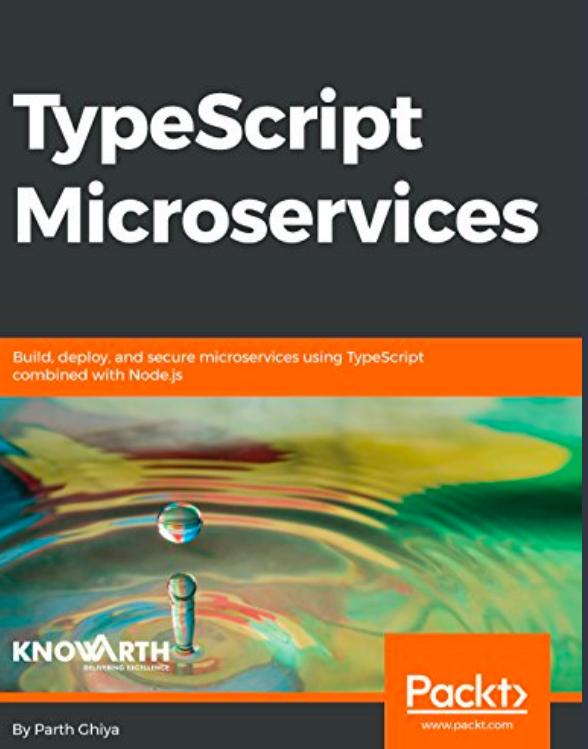
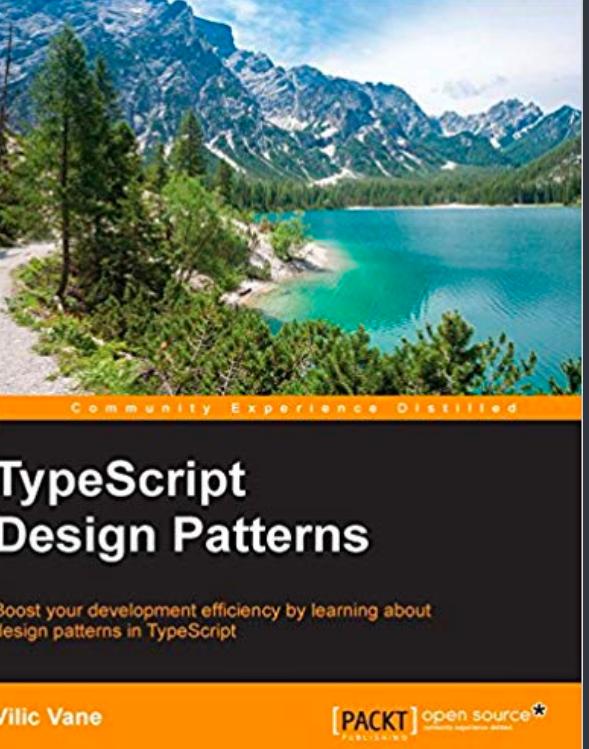
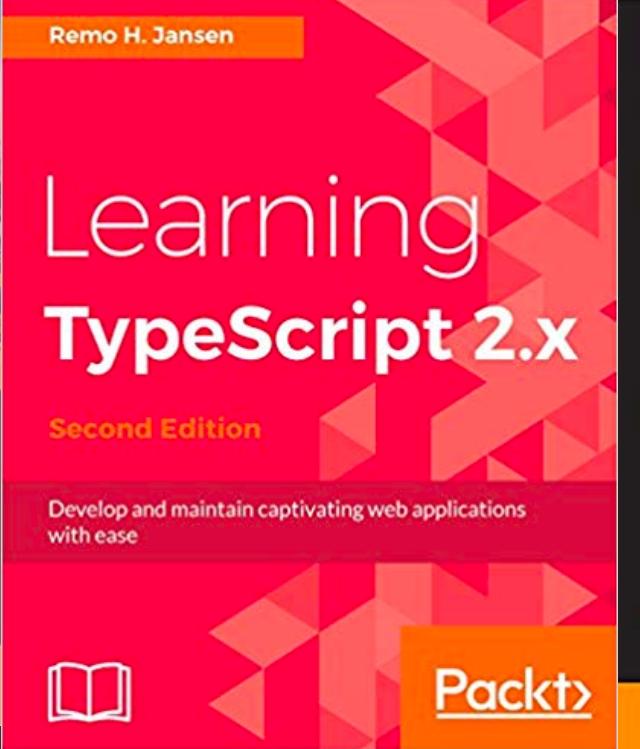
Deep Dive

TS

Basarat Ali Syed

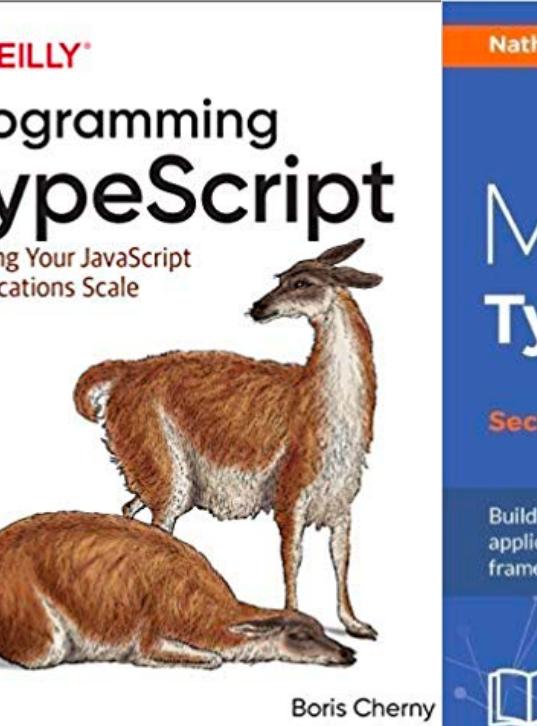
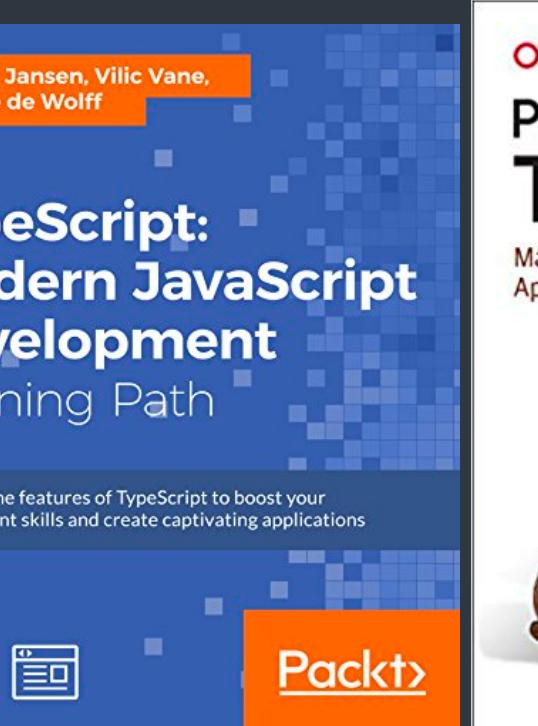
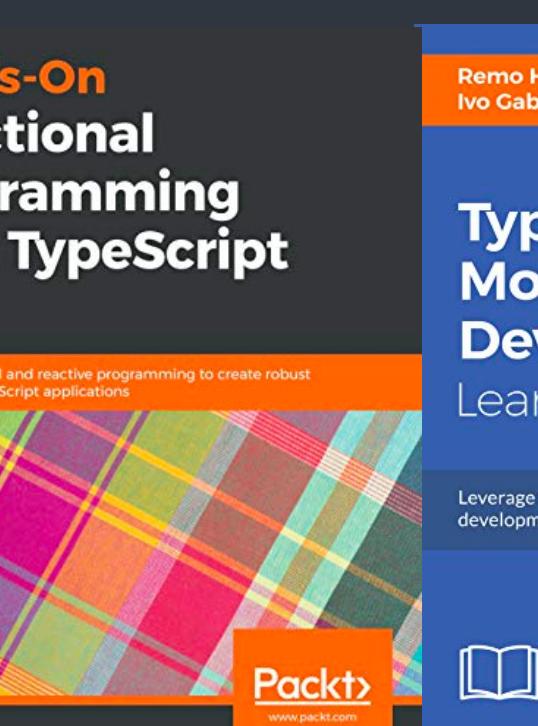
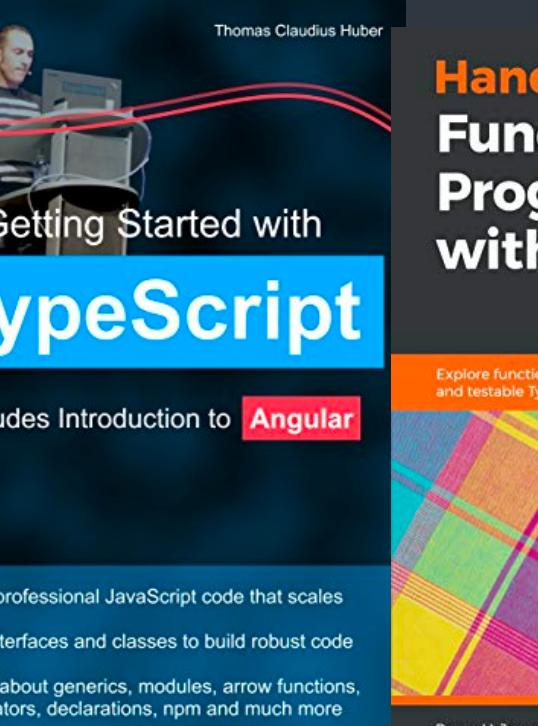
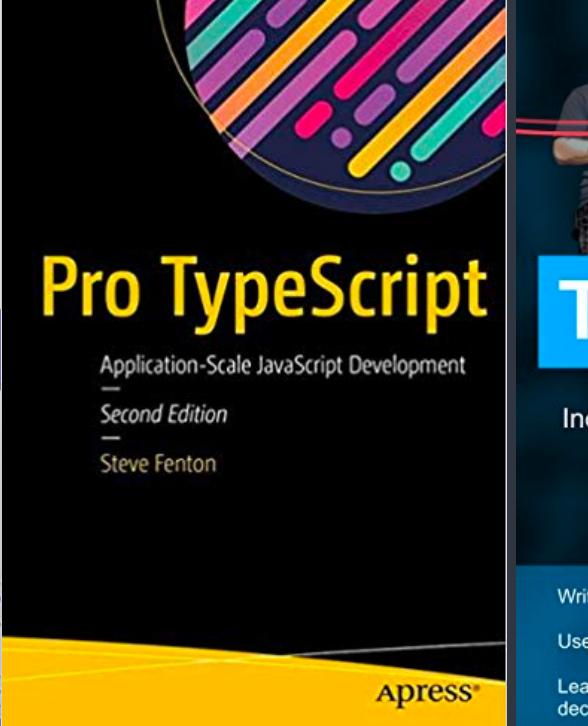
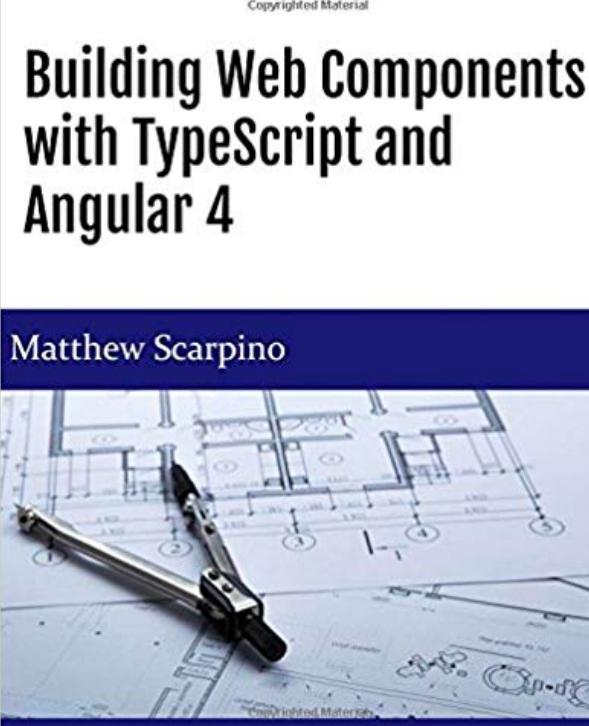
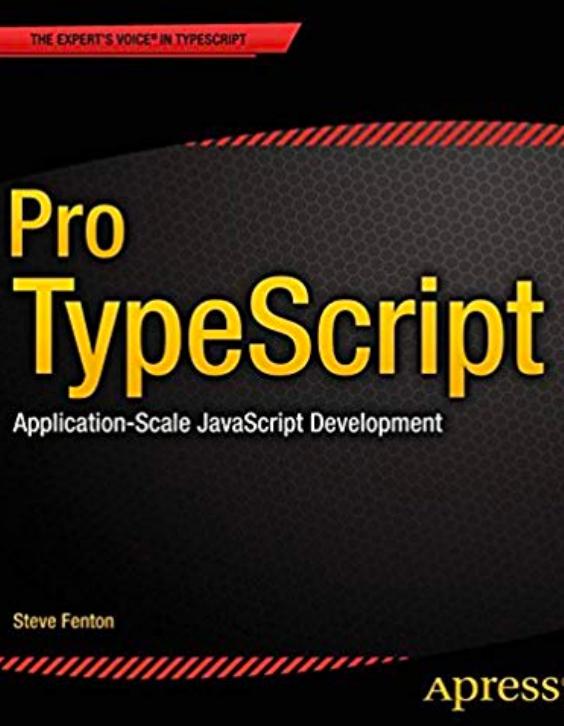
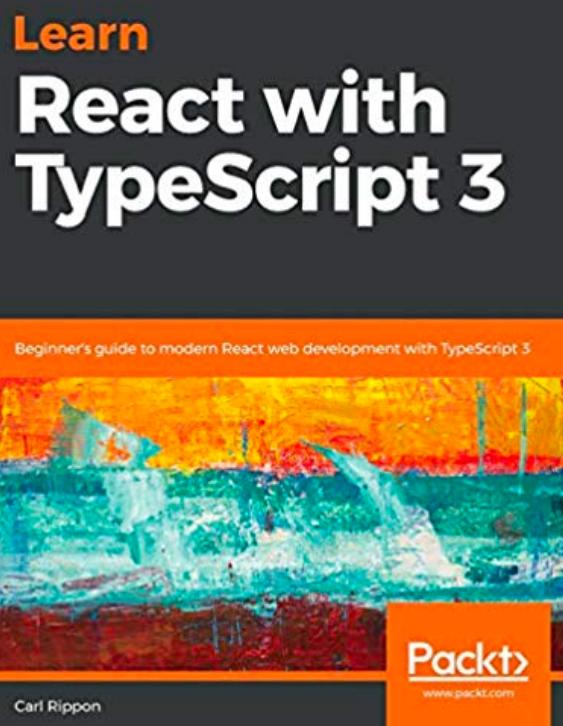


Essential TypeScript



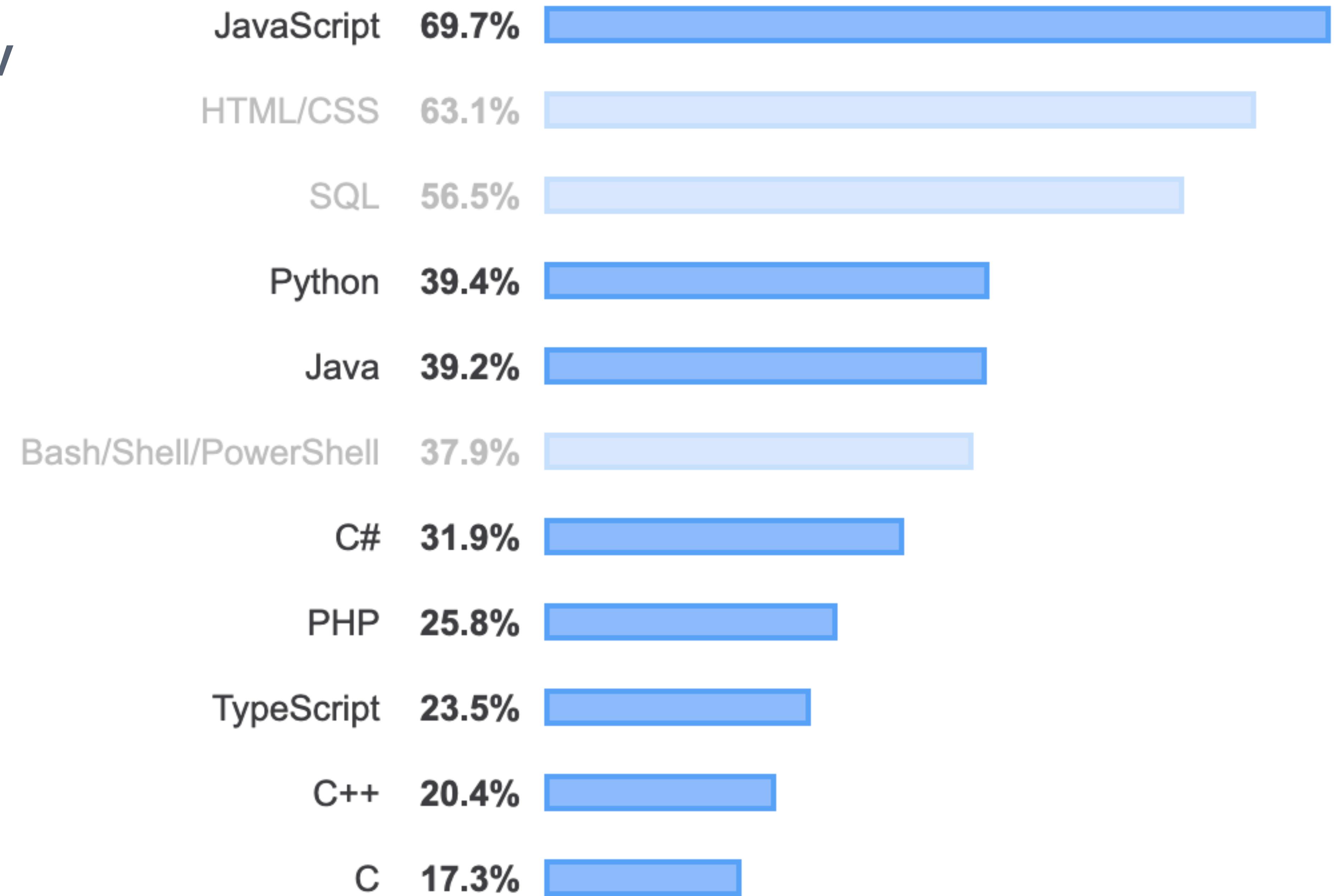
Learn

React with TypeScript 3



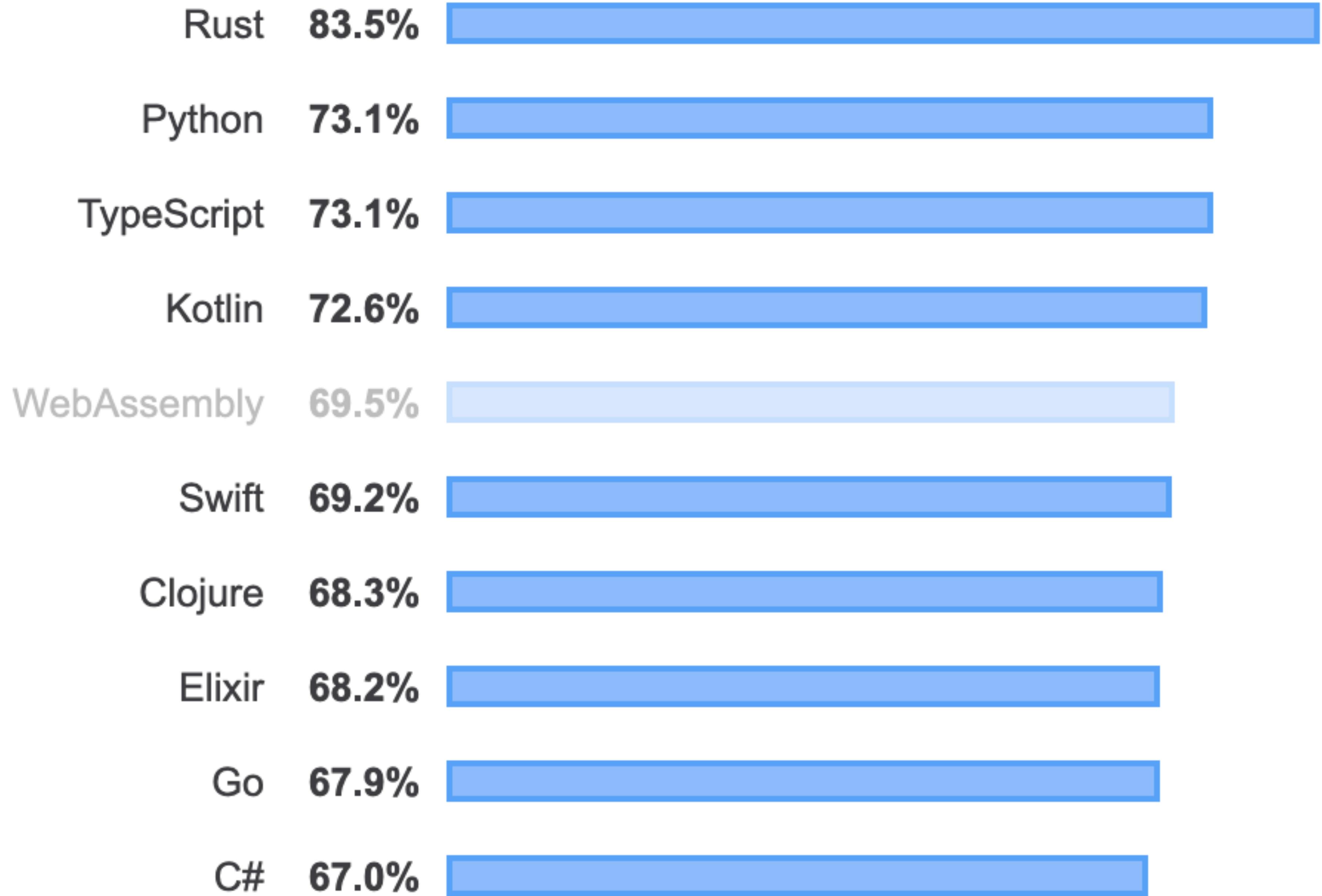
StackOverFlow

Most Popular Technologies



StackOverFlow

Most Loved Languages



TypeScript (or: How I learned to stop worrying and love web development)



Mark Jordan

[Follow](#)

Sep 17, 2018 · 7 min read

```
16 type Pet = Cat | Dog;  
17  
18 function getFood(dispenser: FoodDispenser, pet: Pet): F  
19     switch (pet.type) {  
20         case 'CAT':  
21             return [dispenser.provideWetCatFood(pet),  
22                     dispenser.provideDryCatFood(pet)];  
23         case '':  
24             retunCAT  
25     }  
26 }
```

*« 90% of C#'s typing power
with 50% of the effort
and a bunch of bonuses on top »*

For most of my programming career so far I've been using Java or C# on desktop tools, and only dipping into more esoteric languages like C++ or python for fun hobby projects. That all changed when I started working on a new tool with a web UI (albeit hosted in electron for the moment) and a whole new bag of technologies. Some felt kinda familiar—

DÉMO

code TS
JS généré
outillage, debug

SYNTAXE



Classes

```
class Person {  
    firstName = '';  
    lastName = '';  
    fullName(): string {  
        return `${this.firstName} ${this.lastName}`;  
    }  
}  
  
const me = new Person();  
me.firstName = 'Thierry';  
me.lastName = 'Chatel';  
console.log(me.fullName());
```

Classes

```
class Person {  
    constructor (public firstName: string,  
                public lastName: string) {  
    }  
    getFullName(): string {  
        return `${this.firstName} ${this.lastName}`;  
    }  
}  
  
const me = new Person('Thierry', 'Chatel');  
console.log(me.getFullName());
```

Classes

```
// Angular Component
export class PhotoPageComponent implements OnInit {
    subscription: Subscription;

    constructor(private photoApi: Photo ApiService,
               private result: ResultService,
               private router: Router) { }

    ngOnInit() {
    }
}
```

Getters & Setters

```
class Person {  
    get fullName(): string {  
        return `${this.firstName} ${this.lastName}`;  
    }  
    constructor (public firstName: string,  
                public lastName: string) {  
    }  
}  
  
const me = new Person('Thierry', 'Chatel');  
console.log(me.fullName);
```

Getters & Setters

```
class Person {  
    private _name: string = '';  
    get name(): string {  
        return this._name;  
    }  
    set name(name: string) {  
        this._name = name;  
    }  
}  
  
const me = new Person();  
me.name = 'Thierry';  
console.log(me.name);
```

public, protected, private

```
class Book {  
    public title = 'Abcd';  
    protected author = 'Xyz';  
    private isbn = '132-8263498638';  
    page = 352; // default is public  
}
```

readonly

```
class Book {  
    constructor(public readonly title: string,  
                public readonly author: string) {  
    }  
}  
  
const book = new Book(  
    'La Horde du Contrevent',  
    'Alain Damasio'  
);  
book.author = 'toto'; // Error: read-only property!
```

static

```
class Point {  
    private constructor(public readonly x: number,  
                        public readonly y: number) {  
    }  
    static fromArray(coord: [number, number]): Point {  
        const [x, y] = coord;  
        return new Point(x, y);  
    }  
}  
  
const p = Point.fromArray([12, 5]);
```

extends

```
class Pet {  
    eat(): void {  
        console.log('eat');  
    }  
}  
class Dog extends Pet {  
    run(): void {  
        console.log('run');  
    }  
}  
const dog = new Dog();  
dog.eat();
```

super

```
class Dog extends Pet {  
    constructor() {  
        super();  
    }  
    run(): void {  
        console.log('run');  
    }  
}
```

abstract

```
abstract class Pet {  
    abstract eat(): void;  
}  
class Dog extends Pet {  
    constructor() {  
        super();  
    }  
    eat(): void {  
        console.log('eat');  
    }  
    run(): void {  
        console.log('run');  
    }  
}
```

Interfaces

```
interface Entity {  
    id: number;  
    name: string;  
}  
  
class Person implements Entity {  
    public id: number;  
    constructor(public name: string) {  
        this.id = IdGenerator.get();  
    }  
}  
  
const p: Entity = new Person('Thierry');
```

Interfaces

```
interface Entity {  
    id: number;  
    name: string;  
}  
  
class Person {  
    public id: number;  
    constructor(public name: string) {  
        this.id = IdGenerator.get();  
    }  
}  
  
const p: Entity = new Person('Thierry'); // still ok
```

Interfaces

```
interface Entity {  
    id: number;  
    name: string;  
    timestamp?: number; // optional  
}  
  
const p: Entity = {  
    id: 1,  
    name: 'Thierry',  
};
```

Interfaces

```
interface Entity {  
    id: number;  
    name: string;  
    timestamp?: number; // optional  
    // methods  
    log(): string;  
    touch(timestamp: number): void;  
}
```

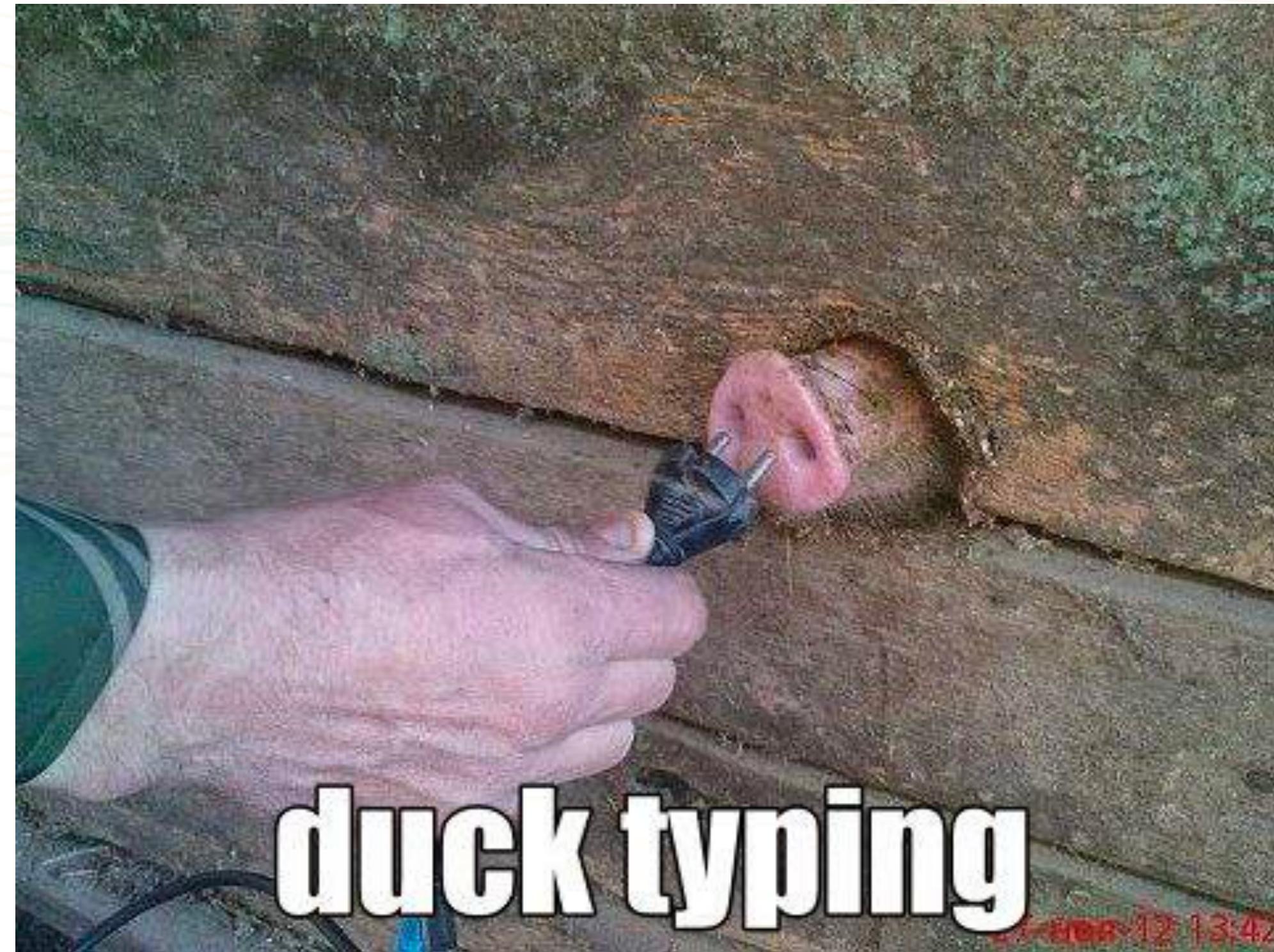
Interfaces

```
interface Entity {  
    id: number;  
    name: string;  
    // methods  
    log(): string;  
    touch(timestamp: number): void;  
}
```

```
interface TimedEntity extends Entity {  
    timestamp: number;  
}
```

Interfaces

« Structural Typing » ≠ « Nominal Typing »



Overloads

```
class Point {  
    // overloads, from most to least specific  
    move(x: number, y: number): void;  
    move(delta: {x: number, y: number}): void;  
  
    // implementation, not a callable type definition  
    move(xOrDelta: number | {x: number, y: number},  
        y?: number): void {  
        // code  
    }  
}  
  
new Point().move(10, 20);  
new Point().move({x: 10, y: 20});
```

Paramètres

```
class Shape {  
  build(type: string,  
        id?: number,  
        size: number = 10,  
        ...otherArgs: string[]) {  
    // code  
  }  
}
```

Déstructuration

```
class Shape {  
  build(type: string,  
        id?: number,  
        size: number = 10,  
        ...otherArgs: string[]) {  
    // code  
  }  
}
```

rest & spread

```
const [x, y] = [10, 5];
```

```
const {x, y} = {x: 10, y: 5};
```

```
return {x, y};
```

```
const numbers = [10, 20, 30];
```

```
const [first, ...otherNumbers] = numbers;
```

```
return [first + 1, ...otherNumbers];
```

```
const obj = {firstName: 'Thierry', lastName: 'Chatel',  
            id: 1, age: 47};
```

```
const {id, firstName, lastName, ...otherProperties} = obj;
```

```
return {firstName, ...otherProperties};
```

async & await

```
function delay(ms: number): Promise<void> {
    return new Promise<void>( resolve =>
        setTimeout( resolve, ms ) );
}

async function dramaticWelcome() {
    console.log('Hello');
    for (let i = 0; i < 3; i++) {
        await delay(500);
        console.log(".");
    }
    console.log('World!');
}
```

namespace

```
export class CropPlugin extends DisplayPlugin {  
    // ...  
}  
  
export namespace CropPlugin {  
    export class CropEvent extends DisplayPlugin.PluginEvent {  
        // ...  
    }  
  
    export interface Api {  
        selection: ApiSelection;  
        removeSelected(): void;  
    }  
}
```

TYPAGE AVANCÉ



Type Aliases

```
type Name = string;  
type NameResolver = () => string;  
type Coord = [number, number];  
type Container<T> = { value: T };
```

Types de base

**boolean, number, string, null, undefined,
symbol**

object, Date, RegExp, Function...

**void, never
any, unknown
this**

DÉMO : unknown

```
function demo1(s: any) {
    console.log(s.substring(1, 5));           // ok
}

function demo2(s: unknown) {
    console.log(s.substring(1, 5));           // error
    if (typeof s !== 'string') {
        console.log(s.substring(1, 5)); // ok
    }
    console.log(s.substring(1, 5));           // error
}
```

DÉMO : this

```
class Collection {  
    add(item: any): this {  
        return this;  
    }  
}  
  
class SpecialCollection extends Collection {  
    remove(item: any): this {  
        return this;  
    }  
}  
  
const collec = new SpecialCollection();  
collec.add(2).remove(1);
```

Énumérations

```
enum Direction {  
    Up = 1,          // By default, first value is 0  
    Down,  
    Left,  
    Right,  
}  
  
enum PetType {  
    DOG = 'dog',  
    CAT = 'cat',  
}
```

Tableaux

```
let stats: number[];  
let stats: Array<number>;  
const stats: number[] = [7, 12, 8, 5];  
const stats = [7, 12, 8, 5];  
  
let points: Array<{x: number, y: number}>;  
let points: Array<Point>;  
  
let matrix: Array<Array<number>>;  
  
let point: [number, number, Color]; // tuple
```

Fonctions... et classes

```
let callback: (n: number, s: string) => boolean;
let callback: (n: number, ...args: string) => boolean;
type PersonFilter = (person: Person) => boolean;

// Class
let clazz: new (name: string) => Person;
let personFactory: (clazz: new (name: string) => Person,
                    name: string) => Person;
type PersonClass = new (person: Person) => boolean;
let pFactory: (clazz: PersonClass, name: string) => Person;
```

& : Intersection Types

```
interface Person {  
    name: string;  
}
```

```
interface WithId {  
    id: number;  
}
```

```
let savePerson: (modified: Person & WithId) => void;
```

```
type PersonWithId = Person & WithId;
```

...with enum (+ DÉMO)

```
enum TitleType { }
enum AuthorType { }

type Title = string & TitleType;
type Author = string & AuthorType;

interface Book { title: Title; author: Author; }

const book1: Book = {
    title: 'La Horde du Contrevent' as Title,
    author: 'Alain Damasio' as Author,
};
```

| : Union Types

```
// Angular Route Guard

canActivate(
  next: ActivatedRouteSnapshot,
  state: RouterStateSnapshot
): Observable<boolean> | Promise<boolean> | boolean {
  return true;
}
```

... with Literal Types

```
type Bit = 0 | 1;
```

```
let bit: Bit = 0;  
bit = 1; // ok  
bit = 2; // error
```

Type Guards

```
// Type Guard with typeof

class Formater {
    leftPad(value: string, padding: string | number):
        string {
        if (typeof padding === 'number') {
            padding = Array(padding + 1).join(' ');
        }
        return padding + value;
    }
}
```

Type Guards

```
// Type Guard with instanceof

class Demo {
    addPet(pet: Dog | Cat): void {
        if (pet instanceof Dog) {
            // pet: Dog
        } else {
            // pet: Cat
        }
    }
}
```

Type Guards

```
// Type Guard with in

interface Dog { name: string; }
interface Cat { name: string; }
interface Duck { }

class Demo {
    addPet(pet: Dog | Cat | Duck): void {
        console.log(pet.name); // error
        if ('name' in pet) {
            console.log(pet.name); // ok, pet: Dog | Cat
        }
    }
}
```

Type Guards

```
// Literal Type Guard

interface Dog { type: 'dog', name: string; }
interface Cat { type: 'cat', name: string; }
interface Duck { type: 'duck' }

class Demo {
    addPet(pet: Dog | Cat | Duck): void {
        if (pet.type === 'duck') {
            console.log('Duck has no name.');// pet: Duck
        } else {
            console.log(pet.name); // ok, pet: Dog | Cat
        }
    }
}
```

Type Guards

```
// User-Defined Type Guard  
interface Fish {  
    swim(): void;  
}  
interface Bird {  
}  
  
function isFish(pet: Fish | Bird): pet is Fish {  
    return 'swim' in pet;  
}
```

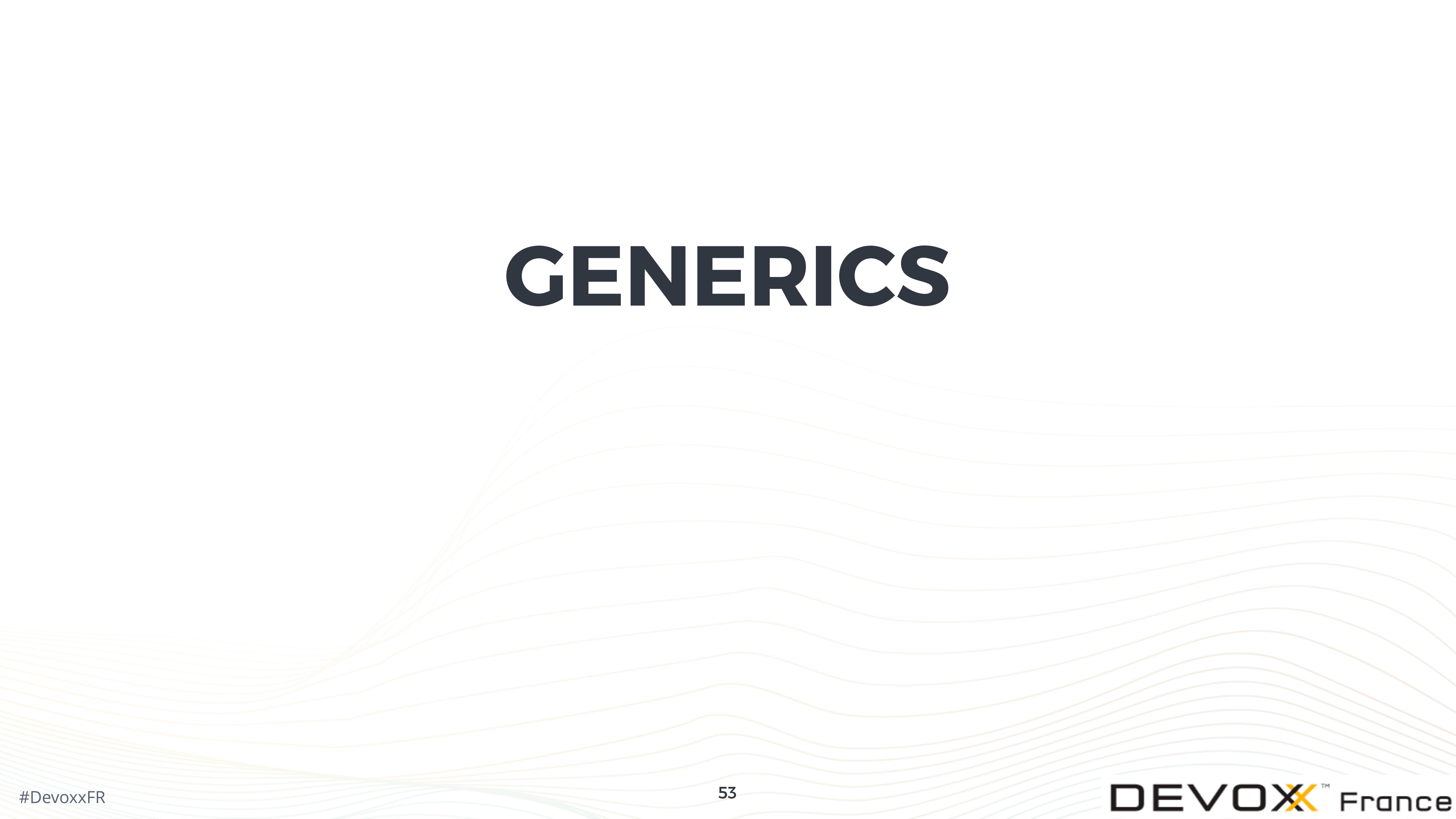
DÉMO : enum + never

```
const fail = (value: never) => {
  throw new Error('Unexpected value: ' + value);
};

enum PetType { DOG, CAT, TURTLE }

function demo(petType: PetType): boolean {
  switch (petType) {
    case PetType.DOG: { return true; }
    case PetType.CAT: { return true; }
    default: {
      fail(petType);           // error
      return false;
    }
  }
}
```

GENERICs



Generics

```
class Demo {  
    identity<T>(param: T): T {  
        return param;  
    }  
}  
  
const demo = new Demo();  
const s: string = demo.identity('abc');  
const n: number = demo.identity(123);
```

Generics

```
class Demo {  
    first<T>(param: T[]): T {  
        return param[0];  
    }  
}  
  
const demo = new Demo();  
const n: number = demo.first([1, 2, 3]);  
const s: string = demo.first(['a', 'b', 'c']);  
const x: string = demo.first(['a', 2]); // error
```

Generics

```
interface Lengthwise {  
    length: number;  
}  
  
class Demo {  
  
    loggingIdentity<T extends Lengthwise>(param: T): T {  
        console.log(param.length);  
        return param;  
    }  
}  
  
new Demo().loggingIdentity('abcd'); // ok  
new Demo().loggingIdentity(123); // error
```

Generics

```
function getProp<T, K extends keyof T>(obj: T, key: K) {  
    return obj[key];  
}  
  
let obj = { a: 1, b: 2, c: 3 };  
  
getProp(obj, 'a'); // okay  
getProp(obj, 'm'); // error
```

Generics

```
interface Array<T> {  
  [n: number]: T;  
  length: number;  
  pop(): T | undefined;  
  push(...items: T[]): number;  
  slice(start?: number, end?: number): T[];  
  sort(compareFn?: (a: T, b: T) => number): this;  
  map<U>(fn: (value: T, index: number, array: T[]) => U): U[];  
  filter<S extends T>(  
    fn: (value: T, index: number, array: T[]) => value is S  
  ): S[];  
  filter(fn: (value: T, index: number, array: T[]) => any): T[];  
}
```

DÉMO : filter + type guard

```
export interface I1 {
  firstName: string;
  lastName: string;
}
export interface I2 {
  firstName: string;
}
function isI1(item: I1 | I2): item is I1 {
  return 'lastName' in item;
}

const input: Array<I1 | I2> = [
  { firstName: 'Thierry', lastName: 'Chatel' },
  { firstName: 'Toto' }
];
const result = input.filter(isI1); // Array<I1>
```

TYPAGE EXTRÊME



Index Types

```
interface Person {  
    id: number;  
    name: string;  
    age: number;  
}
```

```
type PropAge = Person['age']; // number
```

```
type Keys = keyof Person; // 'id' | 'name' | 'age'
```

```
type Props = Person[keyof Person]; // string | number
```

DÉMO : pick()

```
function pick<T, K extends keyof T>(obj: T, keys: K[]): {[P in K]: T[P]} {
  const res = {} as {[P in K]: T[P]};
  keys.forEach(key => res[key] = obj[key]);
  return res;
}

const person = {
  id: 12345,
  name: 'Thierry',
  age: 47,
  active: true,
};
const subObj = pick(person, ['name', 'age']);
```

DÉMO : pluck()

```
export function pluck<T, K extends keyof T>
  (o: T, names: K[]): T[K][] {
  return names.map(n => o[n]);
}

const person = {
  id: 12345,
  name: 'Thierry',
  age: 47,
  active: true,
};
let values = pluck(person, ['id', 'age']); // number[]
```

Conditional Types

```
// T extends U ? X : Y
```

```
type Nullable<T> = T | null | undefined;
```

```
// Distributive Conditional Type (defined in lib.es5.d.ts)
```

```
type NonNullable<T> = T extends null | undefined ? never : T;
```

```
type Primitive = number | string | boolean | null | undefined;
```

```
const p1: Nullable<string> = null; // ok
```

```
const p2: Primitive = null; // ok
```

```
const p3: NonNullable<Primitive> = 'abc'; // ok
```

```
const p4: NonNullable<Primitive> = null; // error
```

Predefined

Extract<Client | Employee | Dog, Person> // Client | Employee

Exclude<Client | Employee | Dog, Person> // Dog

NonNullable<T> // Exclude null and undefined from T

Exclude<T, null | undefined> // same result

// For a function type:

ReturnType<T>

Parameters<T> // tuple

// For a constructor type:

InstanceType<T>

ConstructorParameters<T> // tuple

Predefined

```
// Make all properties of an object type...
Partial<Person> // ...optional
Required<Person> // ...required
Readonly<Person> // ...readonly

// Keep only some properties of an object type:
Pick<Person, 'name' | 'age'>
    // Or other properties :
    Pick<Person, Exclude<keyof Person, 'name' | 'age'>>

// Create an object type with properties of same type:
Record<'firstName' | 'lastName', string>
    // {firstName: string, lastName: string}
```

Not Predefined

// Reverse of Pick<T, K>:

```
type Omit<T, K> = Pick<T, Exclude<keyof T, K>>;
```

// Properties in T but not in U:

```
type Diff<T, U> = Omit<T, keyof U>;
```

// T properties of type U:

```
type KeyNamesOfType<T, U> =
  {[K in keyof T]: T[K] extends U ? K : never}[keyof T];
```

```
type KeysOfType<T, U> = Pick<T, KeyNamesOfType<T, U>>;
```

Mix-in

« A mixin is an abstract subclass; i.e. a subclass definition that may be applied to different superclasses to create a related family of modified classes. »

Gilad Bracha and William Cook, Mixin-based Inheritance

Mix-in

```
interface WithTimestamp { timestamp: number; }

type Clazz<T = {}> = new (...args: any[]) => T;

function Timestamped<TBase extends Clazz>(Base: TBase) {
    return class extends Base implements WithTimestamp {
        timestamp = Date.now();
    };
}

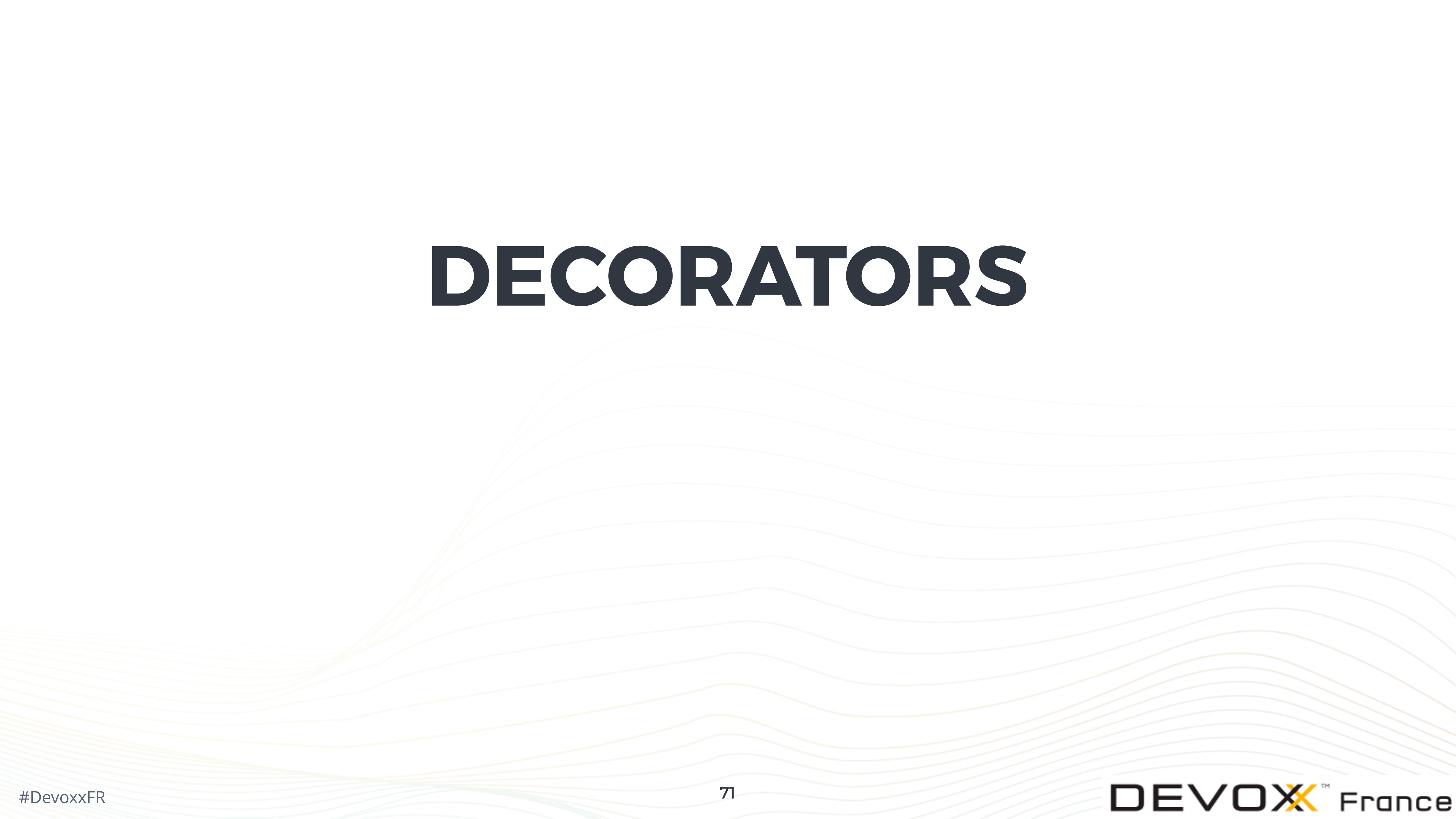
class Person {
    constructor(public firstName: string,
                public lastName: string) { }
}

const TimestampedPerson = Timestamped(Person);
```

(DÉMOS)



DECORATORS



Usage (Angular)

```
@Component({
  selector: 'app-image',
  templateUrl: './image.component.html',
  styleUrls: ['./image.component.css']
})
export class ImageComponent {
  @Input() url: string;
  @Input() legend: string;
  constructor(private imageLoader: ImageLoaderService,
    @Inject(LOCALE_ID) private localeId: string) {
  }
}
```

Usage (tsoa)

```
@Route('user')
@Tags('User')
export class UserController {

    @Response<ErrorResponse>('Unexpected error')
    @Get('UserInfo')
    @Tags('Info', 'Get')
    public async userInfo(@Request() request: any): Promise<UserResponse> {
        return Promise.resolve(request.user);
    }

    @Get('EditUser')
    @Tags('Edit')
    public async userInfo(@Request() request: any): Promise<string> {
        // Do something here
    }
}
```

Création

```
function ExtendedOnInit() {
  return (clazz: new (...args: any[]) => OnInit) => {
    const originalOnInit = clazz.prototype.ngOnInit;
    clazz.prototype.ngOnInit = () => {
      console.log('decorator', 'onInit');
      originalOnInit.call(clazz);
    };
    return clazz;
  };
}
interface OnInit { ngOnInit(): void; }

@ExtendedOnInit()
class MyComponent implements OnInit {
  ngOnInit() { console.log('original ngOnInit'); }
}
```

JS LIBS



.d.ts

Fichiers de déclarations de types

ex: <https://github.com/reduxjs/redux/blob/master/index.d.ts>

DefinitelyTyped - packages @types/*

npm install --save-dev @types/open-layers

lib.d.ts

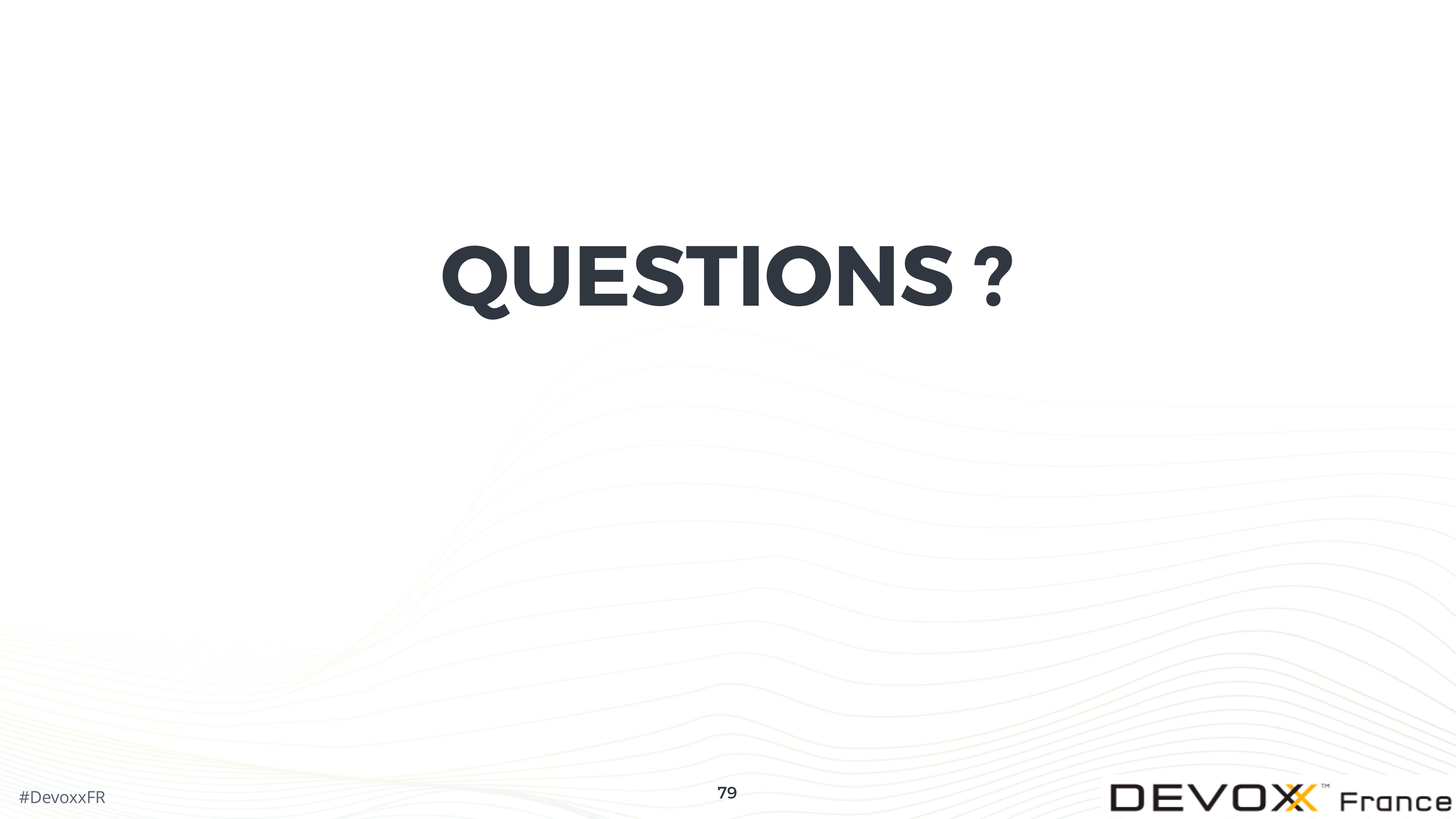
.tsx

```
class List extends React.Component {  
  render() {  
    return (  
      <ul>  
        <li>Item 1</li>  
        <li>Item 2</li>  
        <li>Item 3</li>  
      </ul>  
    );  
  }  
}
```

CONCLUSION

frontend <→ backend
Java, C#

QUESTIONS ?





Thierry Chatel

-  **tchatel@methotic.com**
-  **@ThierryChatel**

Slides :

<https://tchatel.github.io/conf-typescript/typescript.pdf>