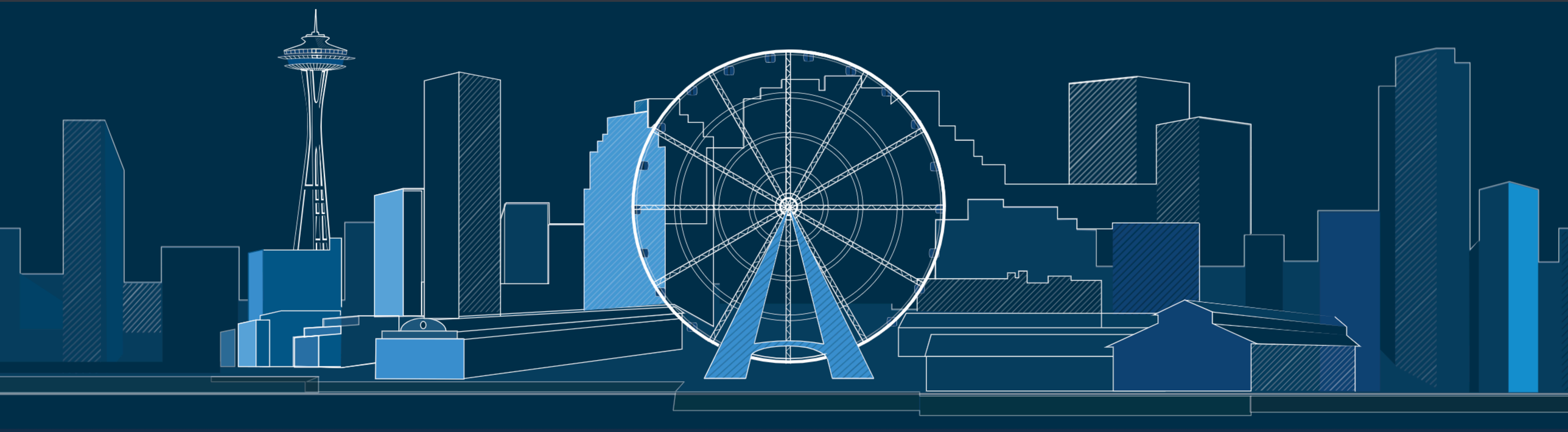
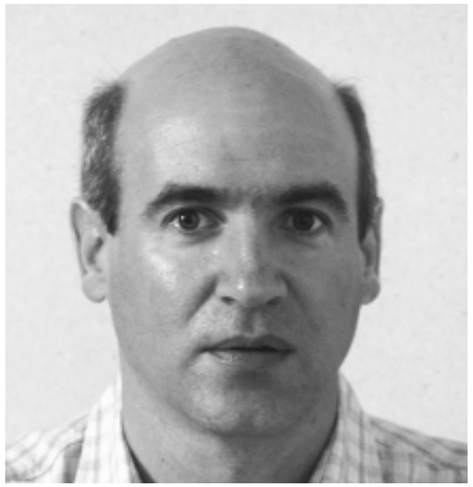


TypeScript

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

MethoTIC Conseil

tchatel@methotic.com

@ThierryChatel

Crealead
Coopérative d'entrepreneurs

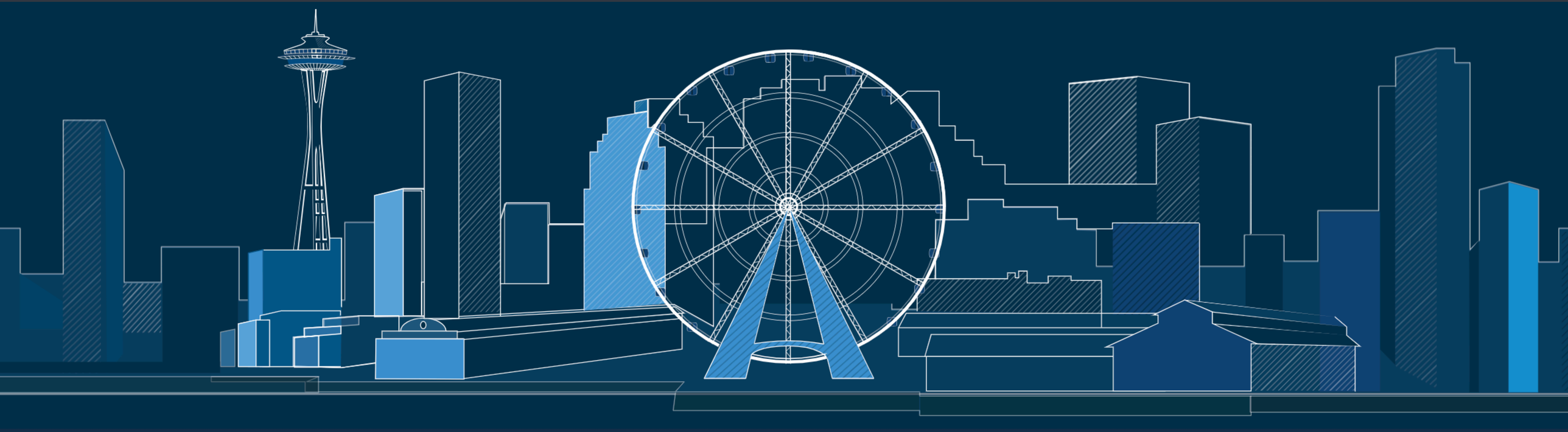
- consultant
- formateur
- GDE Angular

INTRODUCTION

DEVOXX™
France 2015

DEVOXX
FRANCE

Comprendre enfin JavaScript

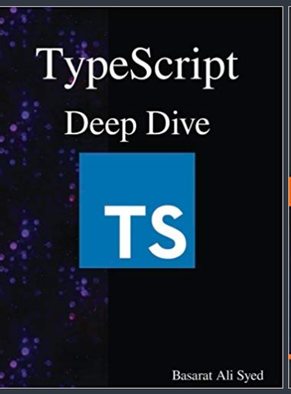


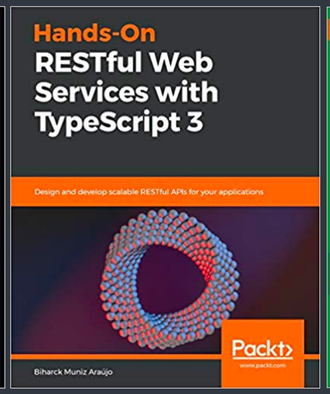
TypeScript

JavaScript that scales.

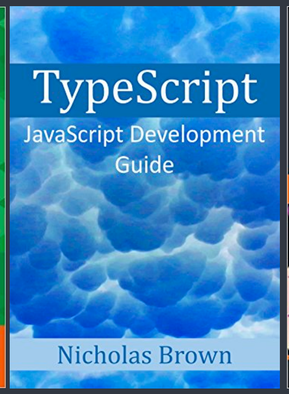
TypeScript is a typed superset of JavaScript that compiles to plain JavaScript.

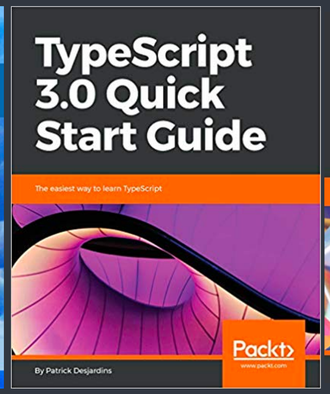
Any browser. Any host. Any OS. Open source.

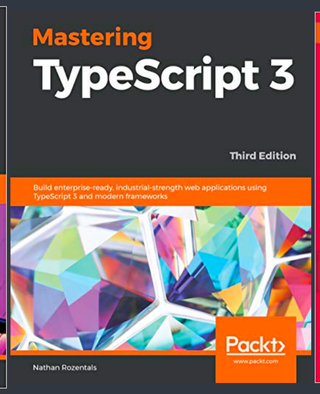

TypeScript
Deep Dive
TS
Basarat Ali Syed

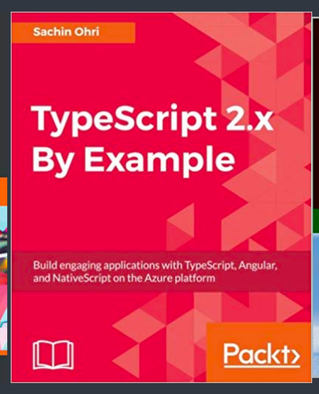

Hands-On
RESTful Web Services with TypeScript 3
Design and develop scalable RESTful APIs for your applications
Biharck Muniz Araújo
Packt

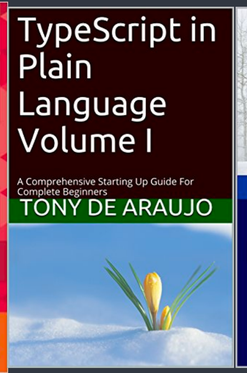

Ajinkya Kher
Foreword by
Maria Riaz, PhD
Software Engineer Google Inc
Brian O'Connor
Solution Principal Slalom Consulting
TypeScript High Performance
Code for performance, use asynchronous programming, and deliver resources efficiently
Packt

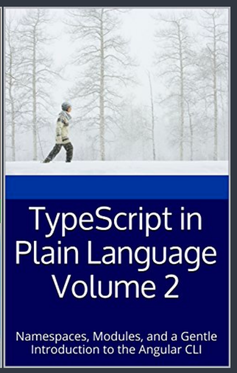

TypeScript
JavaScript Development Guide
Nicholas Brown

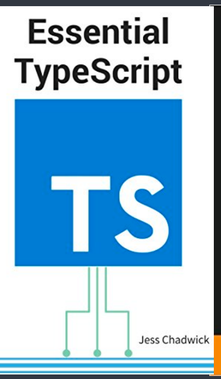

TypeScript 3.0 Quick Start Guide
The easiest way to learn TypeScript
By Patrick Desjardins
Packt

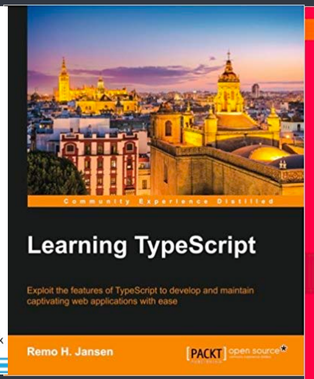

Mastering
TypeScript 3
Third Edition
Build enterprise-ready, industrial-strength web applications using TypeScript 3 and modern frameworks
Nathan Rozentals
Packt

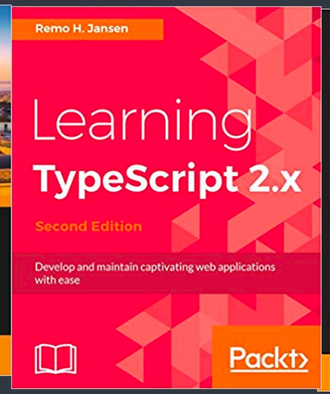

Sachin Ohri
TypeScript 2.x By Example
Build engaging applications with TypeScript, Angular, and NativeScript on the Azure platform
Packt

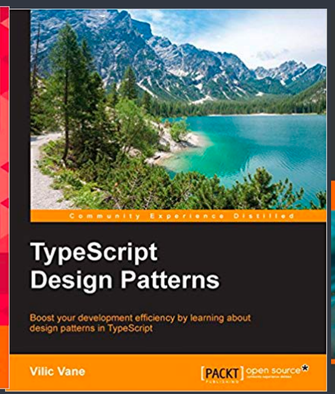

TypeScript in Plain Language Volume I
A Comprehensive Starting Up Guide For Complete Beginners
TONY DE ARAUJO

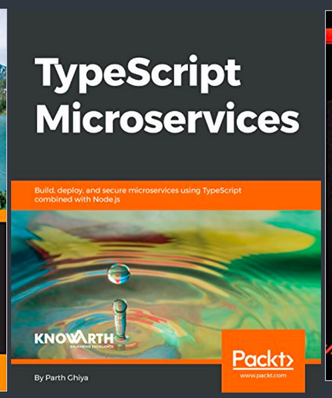

TypeScript in Plain Language Volume 2
Namespaces, Modules, and a Gentle Introduction to the Angular CLI

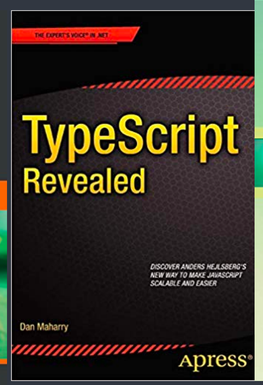

Essential TypeScript
TS
Jess Chadwick

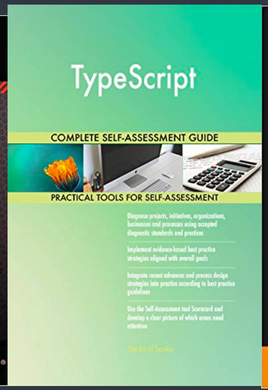

Learning TypeScript
Exploit the features of TypeScript to develop and maintain captivating web applications with ease
Remo H. Jansen
PACKT open source

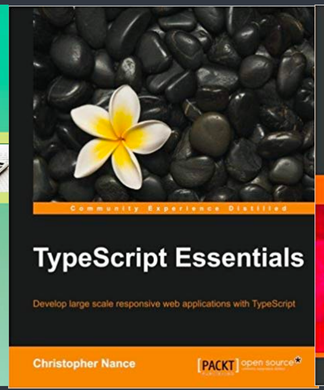

Remo H. Jansen
Learning TypeScript 2.x
Second Edition
Develop and maintain captivating web applications with ease
Packt

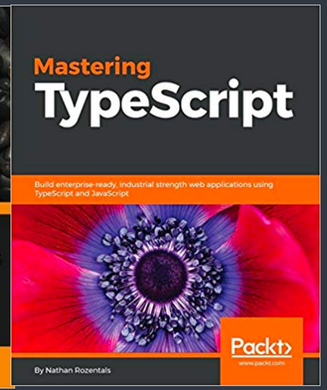

TypeScript Design Patterns
Boost your development efficiency by learning about design patterns in TypeScript
Vilic Vane
PACKT open source


TypeScript Microservices
Build, deploy, and secure microservices using TypeScript combined with Node.js
KNOWARTH
By Parth Ghiya
Packt

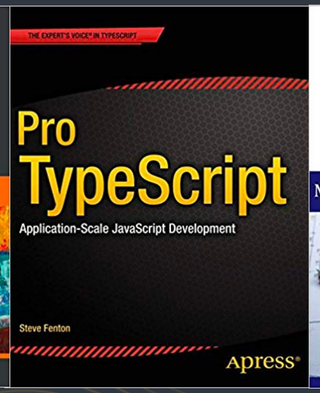

TypeScript Revealed
DISCOVER ANDERS HEJLSBERG'S NEW WAY TO MAKE JAVASCRIPT SCALABLE AND EASIER
Dan Maharry
Apress

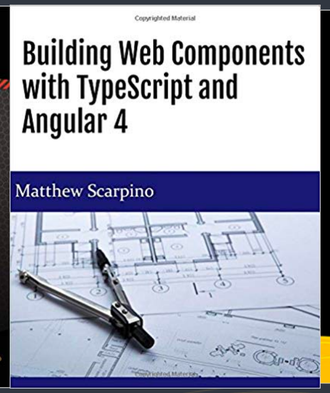

TypeScript
COMPLETE SELF-ASSESSMENT GUIDE
PRACTICAL TOOLS FOR SELF-ASSESSMENT

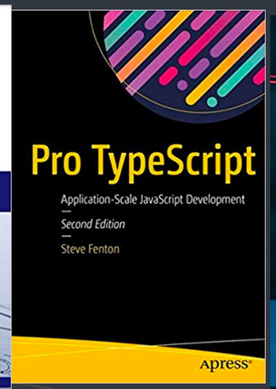

TypeScript Essentials
Develop large scale responsive web applications with TypeScript
Christopher Nance
PACKT open source


Mastering TypeScript
Build enterprise-ready, industrial strength web applications using TypeScript and JavaScript
By Nathan Rozentals
Packt

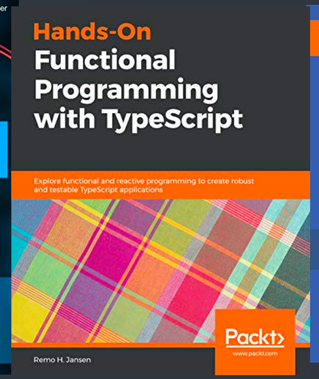

Learn
React with TypeScript 3
Beginner's guide to modern React web development with TypeScript 3
Carl Rippon
Packt

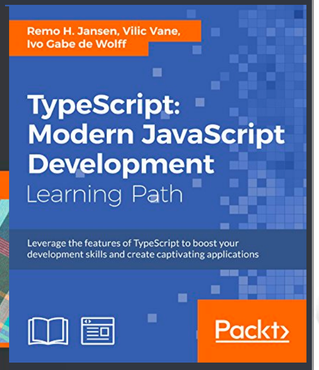

Pro TypeScript
Application-Scale JavaScript Development
Steve Fenton
Apress

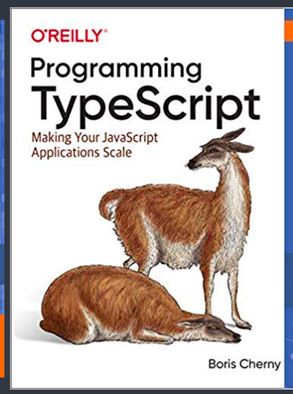

Building Web Components with TypeScript and Angular 4
Matthew Scarpino

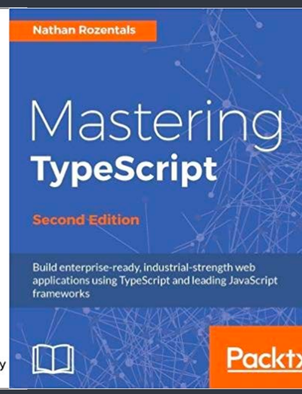

Pro TypeScript
Application-Scale JavaScript Development
Second Edition
Steve Fenton
Apress


Thomas Claudius Huber
Getting Started with
TypeScript
Includes Introduction to Angular
Write professional JavaScript code that scales
Use interfaces and classes to build robust code
Learn about generics, modules, arrow functions, decorators, declarations, npm and much more


Hands-On
Functional Programming with TypeScript
Explore functional and reactive programming to create robust and testable TypeScript applications
Remo H. Jansen
Packt

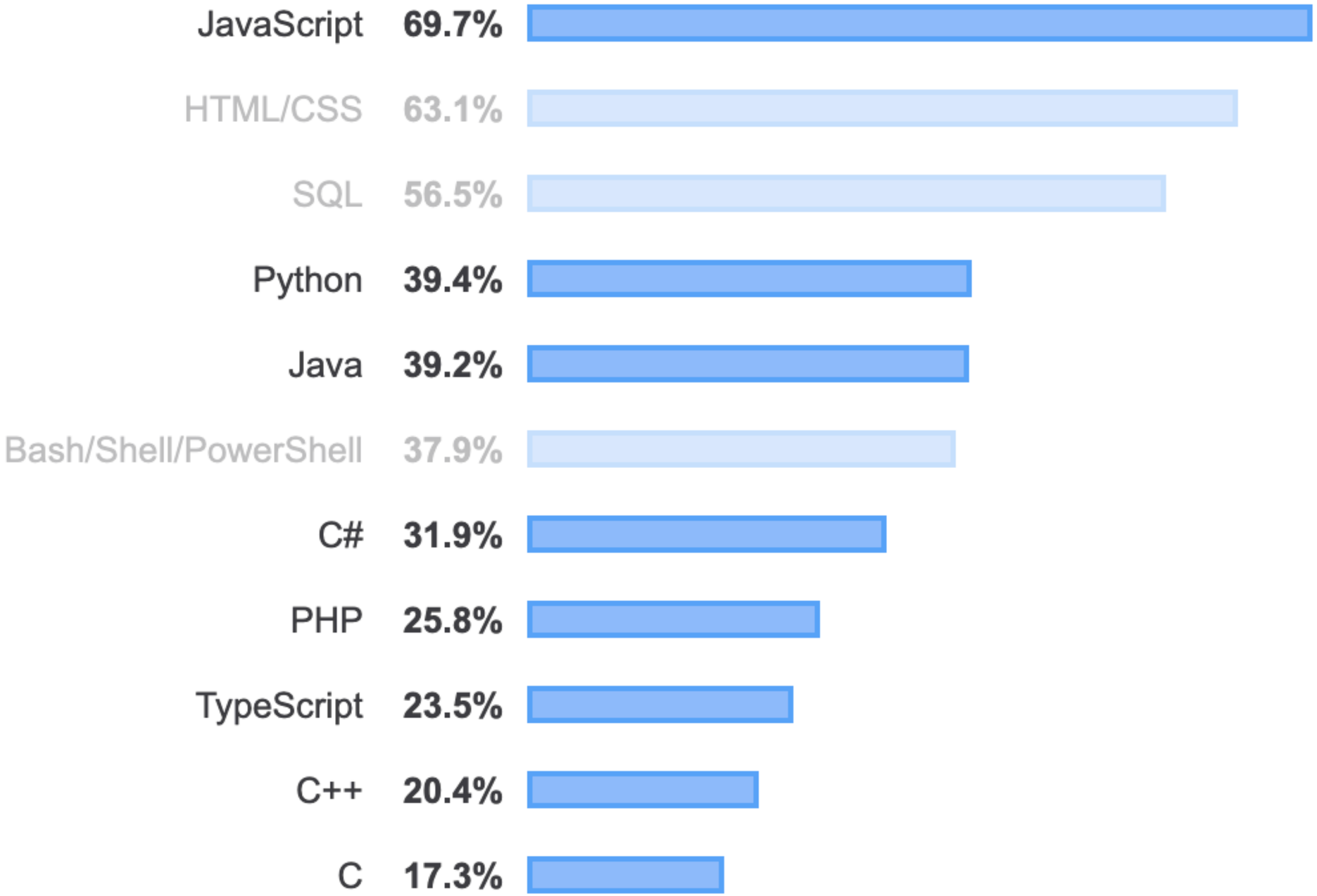

Remo H. Jansen, Vilic Vane, Ivo Gabe de Wolff
TypeScript: Modern JavaScript Development
Learning Path
Leverage the features of TypeScript to boost your development skills and create captivating applications
Packt


O'REILLY
Programming TypeScript
Making Your JavaScript Applications Scale
Boris Cherny


Nathan Rozentals
Mastering TypeScript
Second Edition
Build enterprise-ready, industrial-strength web applications using TypeScript and leading JavaScript frameworks
Packt

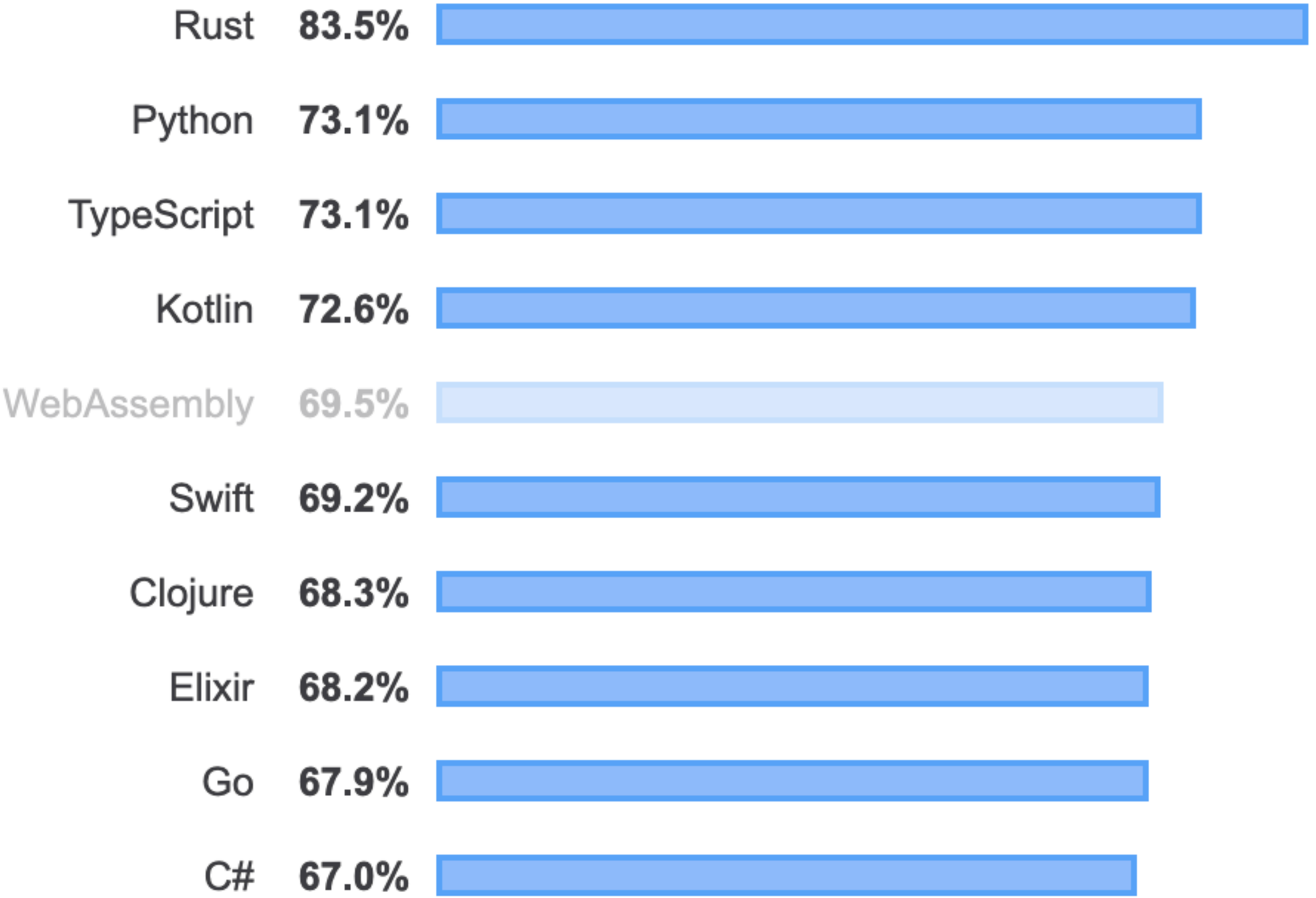
StackOverFlow

Most Popular Technologies



StackOverFlow

Most Loved Languages



DÉMO

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: PhotoApiService,
              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;
}
```

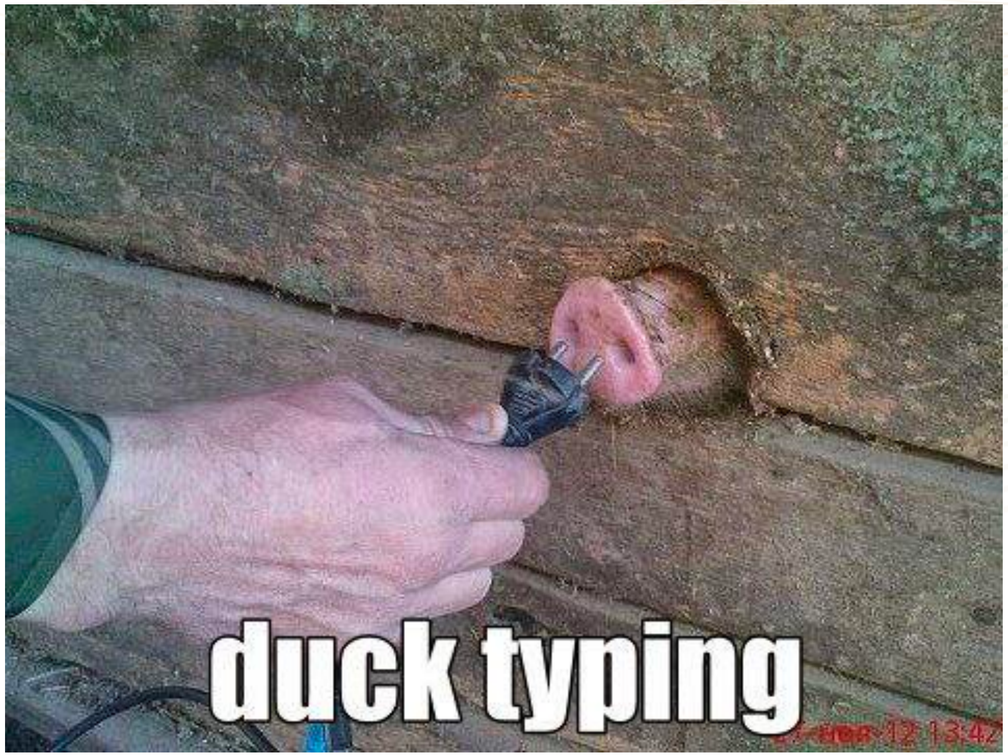
Interfaces

```
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

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

```
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;
}
```

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 };

getProperty(obj, 'a'); // okay
getProperty(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[];
}
```

TYPAGE EXTRÊME

Index Types

```ts
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
```

Index Types

```
function pick<T, K extends keyof T>(obj: T, keys: K[]):
                                        {[P in K]: T[P]} {
    const res = {};
    keys.forEach(key => res[key as string] = obj[key] as any);
    return res as any;
}
const person = {
    id: 12345,
    name: 'Thierry',
    age: 47,
    active: true,
};
const subObj: {name: string, age: number}
  = pick(person, ['name', 'age']);
```

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);
```

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 onInit'); }
}
```

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>
    );
  }
}
```

TODO : conclusion ?

TODO : lien slides