## Лекция 12 TypeScript

Дмитрий Зайцев Мартин Комитски



#### План на сегодня

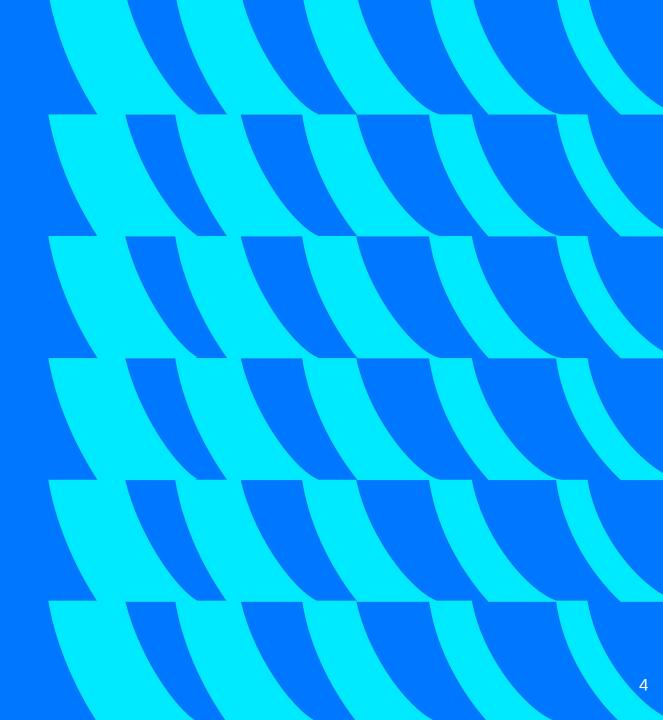
- Введение
- Настройка окружения, Hello World
- Основные принципы и возможности TS
- Практика

#### Минутка бюрократии

- Внимание
- Отметки о посещении занятий
- Обратная связь о лекциях



## Введение



#### Typescript. Введение

#### Typescript is:

- A JavaScript that scales.
- A typed superset of JavaScript that compiles to plain JavaScript.

https://www.typescriptlang.org



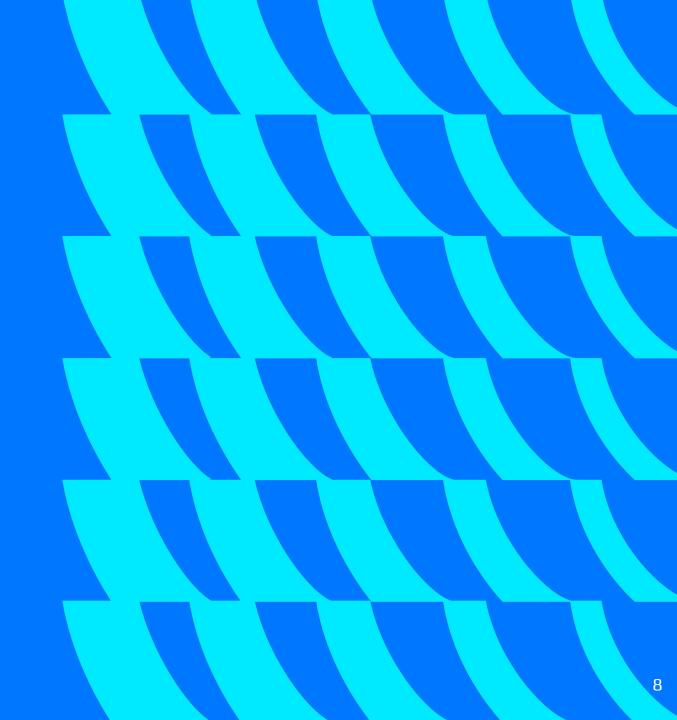
#### Typescript. Зачем он нужен? Что дает?

- Дает возможность привести в порядок код больших приложений
- Привносит новые концепции классических ЯП
- Дополнительные возможности в IDE
- Дополнительные возможности из ECMAScript

#### Typescript. Требования

- node
- npm
- tsc
- Продвинутая IDE или текстовый редактор с расширенной поддержкой TS, например, VS Code

### Типы данных



```
1. // Boolean
     let isDone: boolean = false;
 3.
    // Number
 5. let decimal: number = 6;
 6. let hex: number = 0xf00d;
 7. let binary: number = 0b1010;
     let octal: number = 00744;
10. // String
     let color: string = "blue";
12.
     color = 'red';
13.
14. // Array
15. let list: number[] = [1, 2, 3];
16.
     let list: Array<number> = [1, 2, 3];
17.
18. // Tuple
19. let x: [string, number];
20. x = ["hello", 10]; // ОК. How получить error?
```

```
// Enum
 2. enum Color {Red, Green, Blue}
     let c: Color = Color.Green;
    // Any
 6. let notSure: any = 4;
     notSure = "maybe a string instead";
     notSure = false; // okay, definitely a boolean
 9.
10.
     // Unknown aka type-safe any
11.
     let notSure: unknown = 5;
     let sure: number = notSure; // Error
13.
14. // Void
15. function warnUser(): void {
         console.log("This is my warning message");
16.
17. }
18.
19. // Null, Undefined
20. // Not much else we can assign to these variables!
21. let u: undefined = undefined;
    let n: null = null;
```

```
1. // Never
2. // Function returning never must have unreachable end point
    function error(message: string): never {
        throw new Error(message);
 5. }
6.
7. // Inferred return type is never
    function fail() {
       return error("Something failed");
10. }
11.
12. // Function returning never must have unreachable end point
    function infiniteLoop(): never {
14.
        while (true) {
15.
16. }
```

```
    // Object
    declare function create(o: object | null): void;
    create({ prop: 0 }); // OK
    create(null); // OK
    create(42); // Error
    create("string"); // Error
    create(false); // Error
    create(undefined); // Error
```

#### Typescript. Типы данных. Type assertions

```
    // Type assertions
    let someValue: any = "this is a string";
    let strLength: number = (<string>someValue).length;
    let someValue: any = "this is a string";
    let strLength: number = (someValue as string).length;
```

#### Typescript. Типы данных. Пересечение и объединение типов

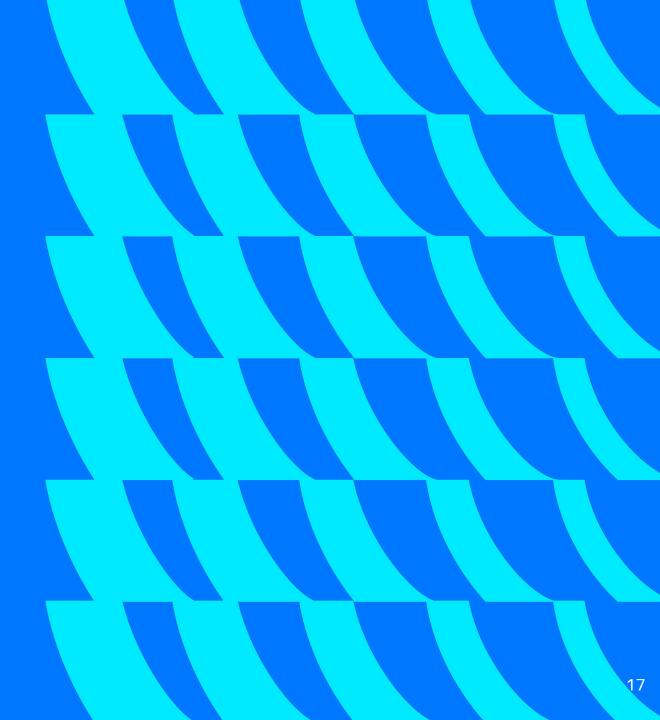
- 1. type TRandomIntersection = number & string; (Плохой пример, вернёмся на интерфейсах)
- 2. type TRandomUnion = number | string;

## Type Inference

#### Typescript. Типы данных. Type Inference

```
    // Type Inference
    let x = 3; // typeof x === 'number'
```

# Функции

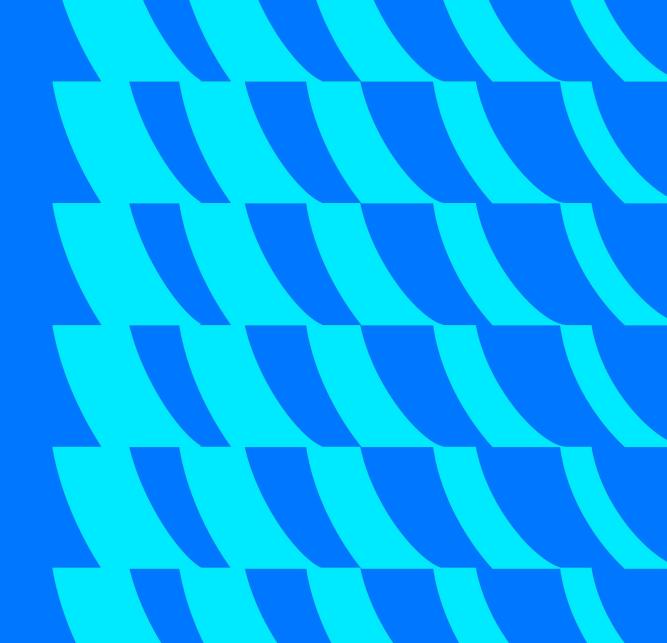


```
1. // Typing the function
    function add(x: number, y: number): number {
3.
         return x + y;
 4. }
 5.
     let myAdd = function(x: number, y: number): number { return x + y; };
 7.
     // Writing the function type
     let myAdd: (x: number, y: number) => number =
10.
         function(x: number, y: number): number { return x + y; };
11.
12.
    let myAdd: (baseValue: number, increment: number) => number =
13.
         function(x: number, y: number): number { return x + y; };
```

```
    // Inferring the types
    // myAdd has the full function type
    let myAdd = function(x: number, y: number): number { return x + y; };
    // The parameters 'x' and 'y' have the type number
    let myAdd: (baseValue: number, increment: number) => number =
    function(x, y) { return x + y; };
```

```
// Overload
    function sum(x: any,y: any) {
        if (typeof x === 'number') {
 4.5.6.
           return x + y;
     } else {
           return `${x} ${y}`;
 8.
9.
10.
    function sum(x: number,y: number)
11.
     function sum(x: string,y: number) {
        if (typeof x === 'number') {
12.
13.
           return x + y;
14. } else {
           return `${x} ${y}`;
15.
16.
17. }
```

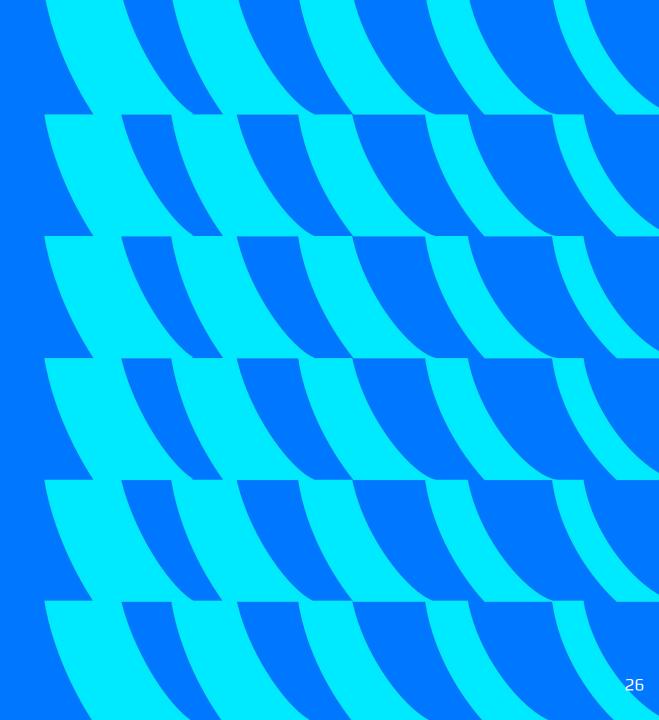
## Интерфейсы



#### Typescript. Интерфейсы

```
// Our first interface
    function printLabel(labeledObj: { label: string }) {
 3.
         console.log(labeledObj.label);
 4.
 5.
     let myObj = {size: 10, label: "Size 10 Object"};
 7.
     printLabel(myObj);
 8.
     // Our second first interface
10.
     interface LabeledValue {
11.
         label: string;
12.
13.
14.
     function printLabel(labeledObj: LabeledValue) {
15.
         console.log(labeledObj.label);
16.
17.
     let myObj = {size: 10, label: "Size 10 Object"};
18.
19.
     printLabel(myObj);
```

### Классы



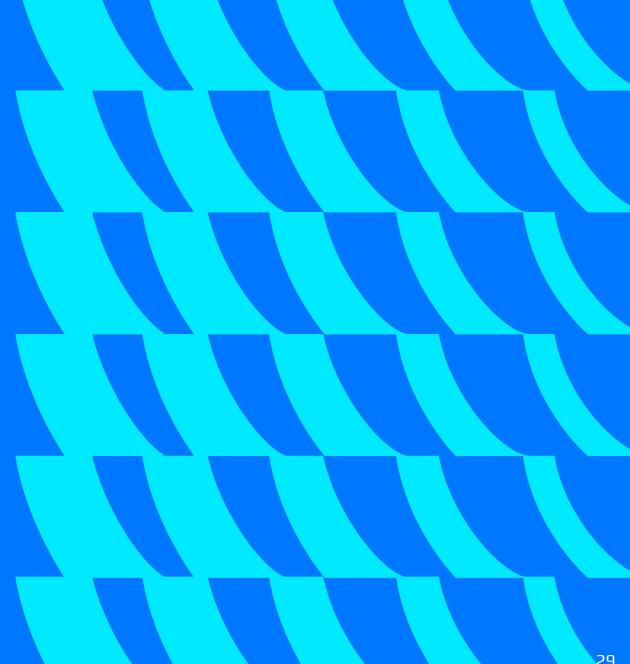
#### Typescript. Классы

```
// Classes
2. class Greeter {
3.
        greeting: string;
     constructor(message: string) {
 5.
            this.greeting = message;
6.
    greet() {
8.
            return "Hello, " + this.greeting;
9.
10. }
11.
12. const greeter = new Greeter("world");
```

#### Typescript. Классы

```
1. // Public, private, and protected modifiers
2. class Animal {
3.    public name: string;
4.    public constructor(theName: string) { this.name = theName; }
5.    public move(distanceInMeters: number) {
6.        console.log(`${this.name} moved ${distanceInMeters}m.`);
7.    }
8. }
9.
10. // Readonly, static
11. // Abstract
```

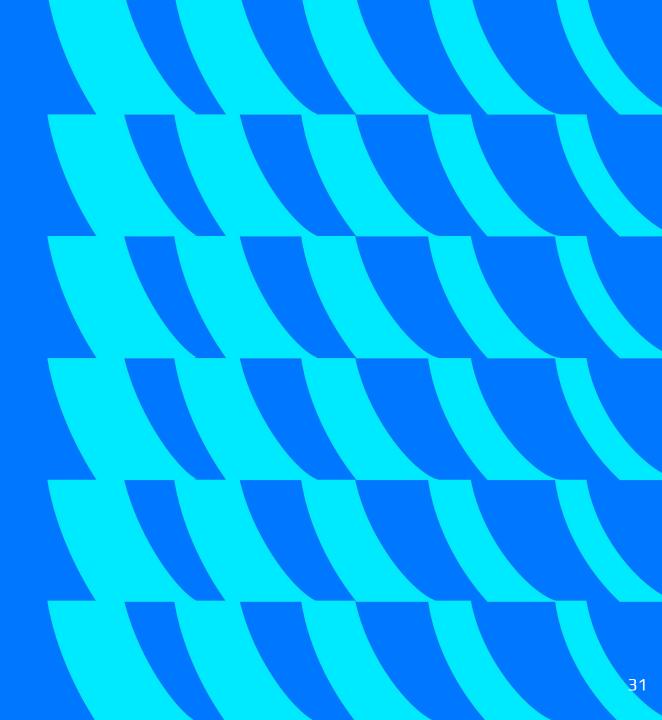
### Generics



#### Typescript. Generics

```
    // Generic
    function echo<T>(arg: T): T {
    return arg;
    }
```

### Enums

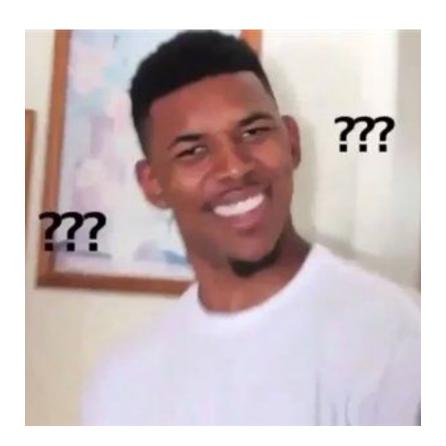


#### Typescript. Enums

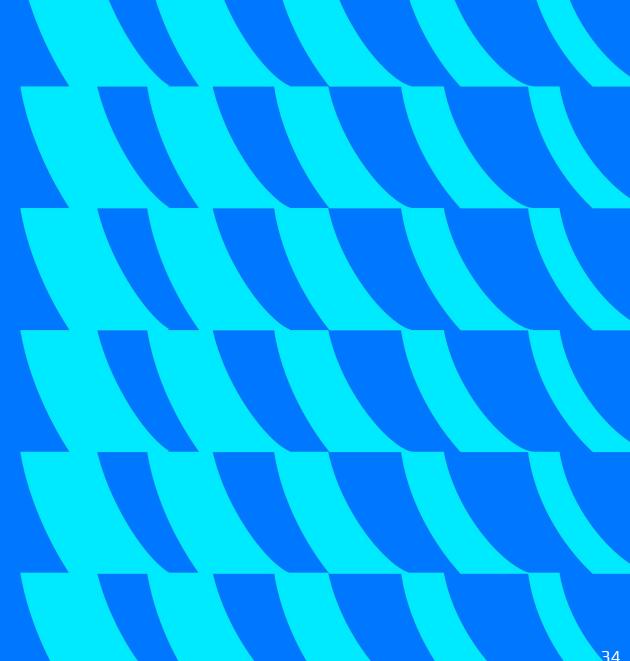
```
enum Direction {
2.
       Up,
3.
       Down,
4. Left,
 5.
   Right,
6. }
    enum Direction {
       Up = "UP",
10. Down = "DOWN",
11. Left = "LEFT",
12. Right = "RIGHT",
13. }
```

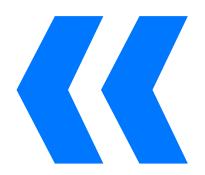
### Typescript?

Вопросы?



## Практика

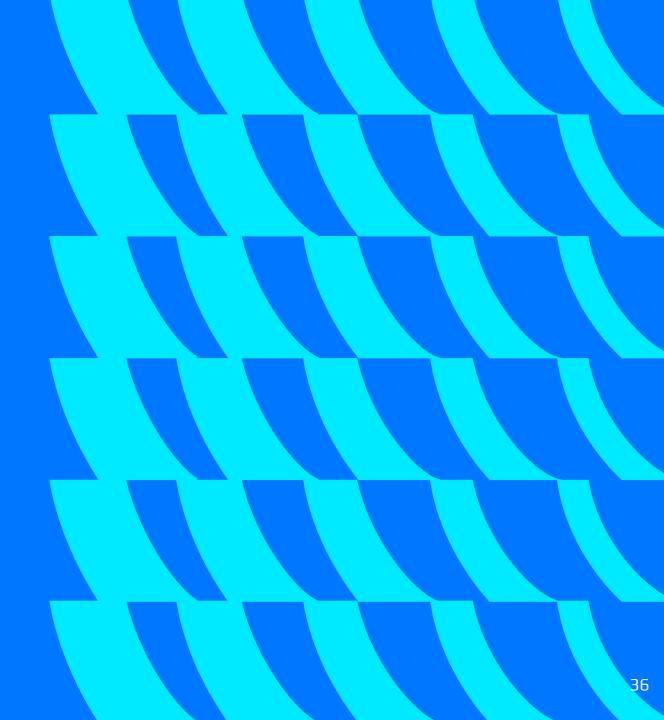




## Перерыв! (О минут)

Препод (с)

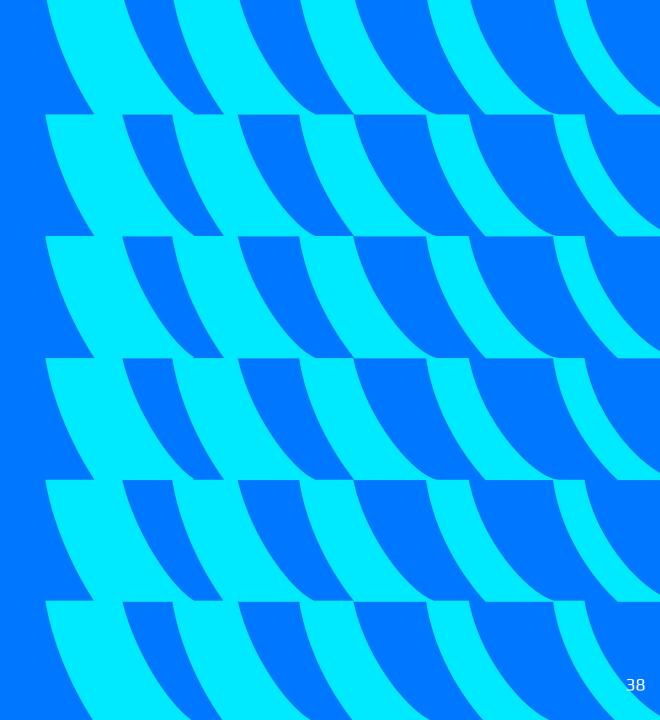
## TypeScript 2



#### План на сегодня

- 1. Возможности TypeScript
  - о Модули
  - Utility Types
  - Decorators
  - Mixins
- 2. React TSX

## Модули



## Модули

- export
- import
- modules/first.ts
- modules/second.ts
- modules/third.ts
- modules/types.ts

### **Export**

```
    export default () => ...
    export {}
    export { ... as ... }
    export { ... as ... } from ...
    export * as ... from ...
```

## **Import**

```
    import ... from ...
    import * as ... from ...
    import {...} from ...
```

Внешние модули. Ambient .D.TS

Порядок импорта при использовании

import ... from '^^^'

- 1. .ts
- 2. .tsx

.D.TS - Ambient module declaration

```
declare module "Backend" {
          export interface ITeacher {
 3.
              name: string;
              level: number;
 5.
              exp?: number;
 6.
 8.
          export interface ILesson {
              title: string;
10.
              day: number;
11.
              hours: number;
12.
              teacher: ITeacher;
13.
14.
15.
          export function makeLesson(title: string, day: number, hours: number, teacher: ITeacher): ILesson;
16. }
```

declare module "CumbersomeBackend"

#### Namespaces

<del>Упорядочивание запутывание</del> упорядочивание пространства имен объектов

- namespaces/validators.ts
- namespaces/mergeQuirqs.ts

```
1. namespace Shapes {
2.    export namespace Polygons {
3.        export class Triangle { }
4.        export class Square { }
5.    }
6. }
```

1. const square = new Shapes.Polygons.Square()

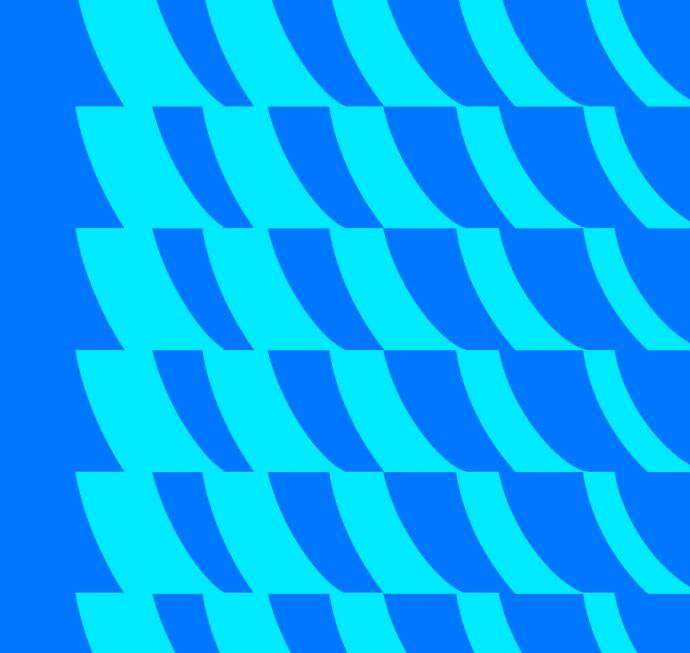
- 1. import polygons = Shapes.Polygons
- 2. const square = new polygons.Square()

```
1. let t = new shapes.Shapes.Triangle() // плохо
```

2. let t = new shapes.Triangle() // xopowo

- Модули и пространства имен нужны для упорядочивания кода
- Использовать и модули и пространства имен одновременно избыточно
- Проще всего работать с модулями

# Declaration merging



#### Полезные ссылки

- https://www.typescriptlang.org/
- Серия коротких роликов про ТЅ
- 7

```
interface Car {
         manufacturer: string;
        name: string;
        vehicleInfo: string;
 5.
 6.
     interface Car {
 8.
         horsePower: number:
      torque: number;
10.
      stickers: string[];
11. }
12.
13.
     const realCar: Car {
14.
         manufacturer: 'VAZ',
15.
     name: 'Granta',
16.
        vehicleInfo: 'Realnii avtomobil dlya realnoi jizni',
17.
        horsePower: 98.
18.
     torque: 145,
19.
         stickers: ['my life - my rules', 'dolbit normalno'],
20. }
```

#### НЕФУНКЦИОНАЛЬНЫЕ СВОЙСТВА

атрибуты интерфейсов должны быть уникальными или одного типа

#### ФУНКЦИОНАЛЬНЫЕ СВОЙСТВА

- каждое новое свойство считается перегрузкой
- последний интерфейс имеет превосходство над предыдущими
- специализированное свойство будет иметь превосходство над обычными

```
1. interface Cloner {
2.    clone(animal: Animal): Animal;
3. }
4.
5. interface Cloner {
6.    clone(animal: Sheep): Sheep;
7. }
8.
9. interface Cloner {
10.    clone(animal: Dog): Dog;
11.    clone(animal: Cat): Cat;
12. }
```

```
interface Cloner {
    clone(animal: Dog): Dog;
    clone(animal: Cat): Cat;
    clone(animal: Sheep): Sheep;
    clone(animal: Animal): Animal;
}
```

```
interface Document {
         createElement(tagName: any): Element; // any
 3.
 5.
     interface Document {
 6.
         createElement(tagName: "div"): HTMLDivElement; // specialized
         createElement(tagName: "span"): HTMLSpanElement; // specialized
 8.
 9.
10.
      interface Document {
11.
         createElement(tagName: string): HTMLElement; // general
12.
         createElement(tagName: "canvas"): HTMLCanvasElement; // specialized
13. }
```

```
    interface Document {
    createElement(tagName: "canvas"): HTMLCanvasElement;
    createElement(tagName: "div"): HTMLDivElement;
    createElement(tagName: "span"): HTMLSpanElement;
    createElement(tagName: string): HTMLElement;
    createElement(tagName: any): Element;
    }
```

#### Typescript. Namespace merging

```
1. namespace Prison {
2.    export class Inspector { }
3.    export class Prisoner { }
4.  }
5.
6. namespace Prison {
   export interface Aparts { facilities: string[]; }
   export class Yard { facilities: string[] }
9. }
```

#### Typescript. Namespace merging

```
1. namespace Prison {
2.     export class Inspector { }
3.     export class Prisoner { }
4.
5.     export interface Aparts { facilities: string[]; }
6.     export class Yard { facilities: string[] }
7. }
```

#### Typescript. Namespace merging

Доступны в общем пространстве имен будут только экспортированные объекты.

namespaces/mergeQuirqs.ts

# UTILITY TYPES \U11F4AA

#### PARTIAL

Все свойства необязательные

#### READONLY

Все свойства – readonly

#### REQUIRED

Все свойства обязательны

#### NONNULLABLE

Все свойства, кроме null, undefined

utilityTypes/commonProps.ts

PICK

Использовать выбранные свойства

OMIT

Использовать все свойства, кроме выбранных

utilityTypes/pickNOmit.ts

#### Typescript. Record

Присваивание свойств одного типа к другому типу.

```
1. interface PageInfo {
2.    title: string;
3. }
4.
5. type Page = 'home' | 'about' | 'contact';
6.
7. const x: Record<Page, PageInfo> = {
8.    about: { title: 'about' },
9.    contact: { title: 'contact' },
10.    home: { title: 'home' },
11. };
```

#### Typescript. Exclude. Extract.

#### **EXCLUDE**

Взять только те свойства, которых нет во втором объекте

```
    type T0 = Exclude<"a" | "b" | "c", "a">; // "b" | "c"
    type T1 = Exclude<"a" | "b" | "c", "a" | "b">; // "c"
    type T2 = Exclude<string | number | (() => void), Function>; // string | number
```

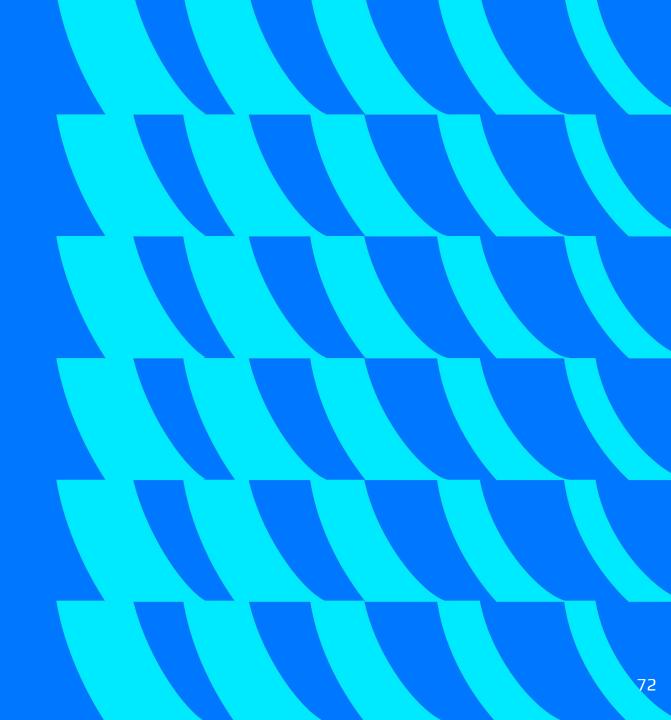
#### **EXTRACT**

Взять только те свойства, которые есть во втором объекте

```
1. type T0 = Extract<"a" | "b" | "c", "a" | "f">; // "a"
2. type T1 = Extract<string | number | (() => void), Function>; // () => void
```

- PARAMETERS
- CONSTRUCTORPARAMETERS
- RETURNTYPE
- INSTANCETYPE
- THISPARAMETERTYPE

## Decorators



#### OOP Design pattern

Расширение функционала вызываемого объекта без его модификации

O from solid

/oop/commonDecorators.ts

```
const noisy = (func) => {
    const wrapper = (...args) => {
        console.log('ppphhhhffffchfiu')
    const res = func(...args)
    console.log('khhhhhheeeeeefffph')
    return res
}
return wrapper
}
```

>> A Decorator is a special kind of declaration that can be attached to a class declaration, method, accessor, property, or parameter.

```
1. const sum = (a, b) => (a + b)
2. const noisySum = noisy(sum)
3. noisySum(1, 15)
```

tsc --target ES5 --experimentalDecorators

"experimentalDecorators": true

oop/typescriptDecorators.ts

```
    // first
    @first @second myFunction
    // second
    @first
    @second
    myFunction
```

#### Typescript. Сигнатура декоратора

- 1. target метод, к которому применяется декоратр
- **2.** key имя метода
- 3. value дескриптор свойства

Дескриптор свойства

Object.getOwnPropertyDescriptor

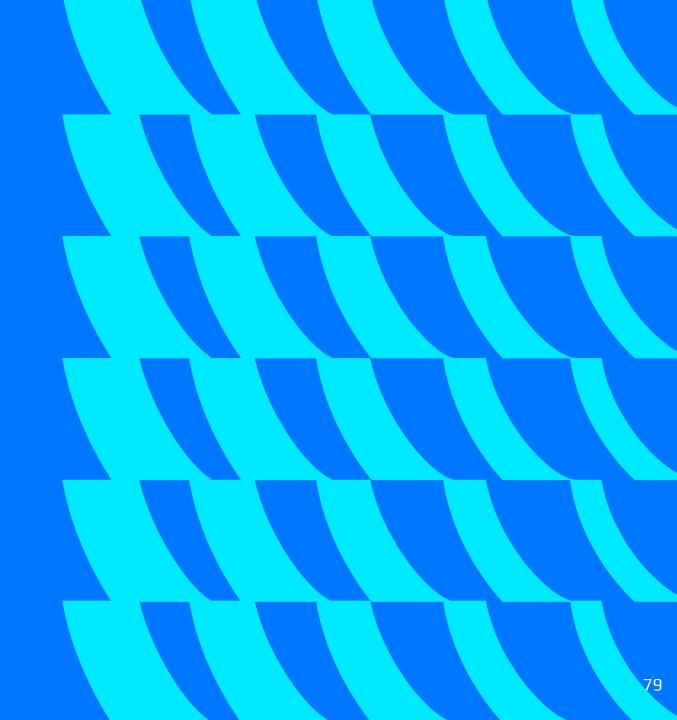
oop/typescriptDecorators.ts

#### **СОЗДАНИЕ ДЕКОРАТОРА.** Decorator factories

```
1. function enumerable(value: boolean) {
2.    return function (target: any, propertyKey: string, descriptor: PropertyDescriptor) {
3.         descriptor.enumerable = value;
4.    };
5. }
```

# Mixins

Примеси для классов



```
1. function applyMixins(derivedCtor: any, baseCtors: any[]) {
2.  baseCtors.forEach(baseCtor => {
3.     Object.getOwnPropertyNames(baseCtor.prototype).forEach(name => {
4.     Object.defineProperty(derivedCtor.prototype, name,
    Object.getOwnPropertyDescriptor(baseCtor.prototype, name));
5.     });
6.  });
7. }
```

```
1. // Activatable Mixin
2. class Activatable {
3.    isActive: boolean;
4.    activate() {
5.        this.isActive = true;
6.    }
7.    deactivate() {
8.        this.isActive = false;
9.    }
10. }
```

```
1. // Disposable Mixin
2. class Disposable {
3.    isDisposed: boolean;
4.    dispose() {
5.        this.isDisposed = true;
6.    }
7. }
```

```
1. class SmartObject {
2.     constructor() {
3.         setInterval(() => console.log(this.isActive + " : " + this.isDisposed), 500);
4.     }
5.     interact() {
7.         this.activate();
8.     }
9. }
```

#### oop/mixins.ts

```
1. interface SmartObject extends Disposable, Activatable {}
2. applyMixins(SmartObject, [Disposable, Activatable]);
3.
4. const smartObj = new SmartObject();
5. setTimeout(() => smartObj.interact(), 1000);
```

## Typescript.TSX

tsx/button.tsx

## Typescript.Типы

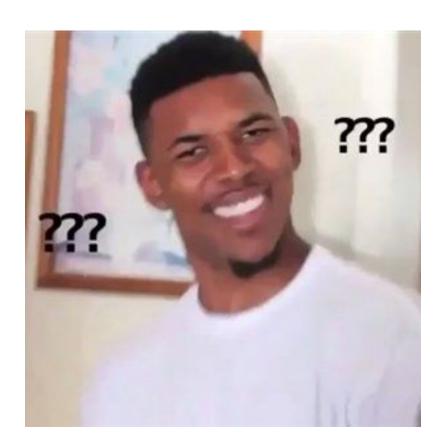
https://github.com/microsoft/TypeScript/blob/master/src/lib/dom.generated.d.ts

### Typescript. Еще типы

https://github.com/DefinitelyTyped/DefinitelyTyped/tree/master/types

# Typescript2?

Вопросы?



#### Домашнее задание №12

- 1. Разработать утилиту на TypeScript
- 2. ?

Расширенное описание задания, подсказки, а также презентации с лекций всегда есть в репозитории.

Срок сдачи

29 декабря

# Спасибо за внимание!



# Пока!

Присоединяйтесь к сообществам про образование в VK

- VK Джуниор
- VK Образование

