

# React: TypeScript

# НАШИ ПРАВИЛА



Включенная камера



Вопросы по поднятой руке



Не перебиваем друг друга



Все вопросы, не связанные с тематикой курса (орг-вопросы и т. д.), должны быть направлены куратору



Подготовьте свое рабочее окружение для возможной демонстрации экрана (закройте лишние соцсети и прочие приложения)

# Повторим;)

■ Для чего удобно использовать метод map в react

■ Что такое key?

■ Как получить значение из состояния компонента?

# ЦЕЛЬ

Познакомиться с основами TypeScript и применить их в React проекте

# ПЛАН ЗАНЯТИЯ

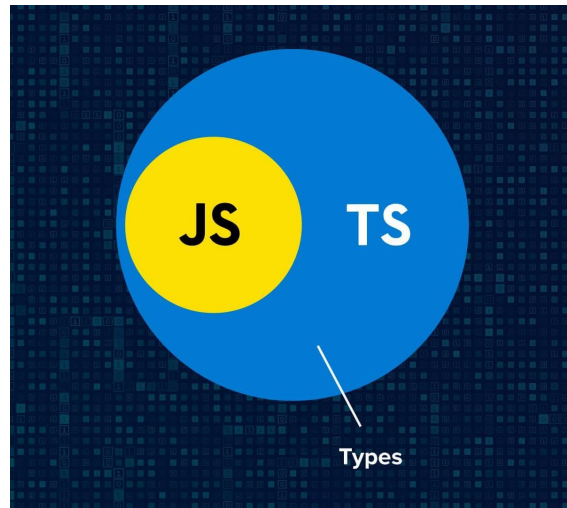
- 
- 1. Основы TypeScript
- 2. React проект с использованием TypeScript

# TypeScript



TypeScript - это типизированная расширенная версия JavaScript со статической типизацией, которая компилируется в простой JavaScript.

Именно эти две особенности позволяют создавать масштабные приложения, сохраняя качество и упрощая разработку.



Ошибки несовпадения типов будут заметны на стадии написания кода.



процесс доставки  
JavaScript кода



процесс доставки  
статически типизированного языка



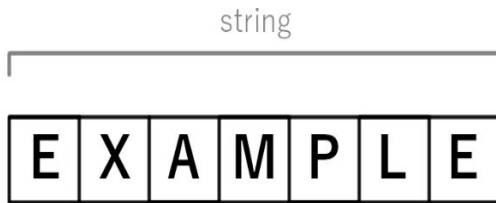
# String

String представляет строки. Как и в JavaScript, в TypeScript строки можно заключать в двойные, либо в одинарные кавычки:

```
1 let firstName: string = "Tom";
```

TypeScript поддерживает такую функциональность, как шаблоны строк, то есть мы можем задать шаблон в косых кавычках (```), как если бы мы писали обычную строку, и затем в саму строку можно встраивать разные выражения с помощью синтаксиса `${ expr }`, где `expr` - это выражение. Например:

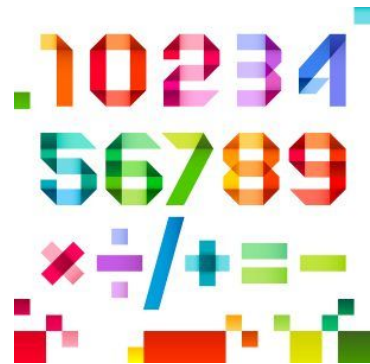
```
1 let firstName: string = "Tom";  
2 let age: number = 28;  
3 let info: string = `Имя ${firstName} Возраст: ${age}`;  
4 console.log(info); // Имя Tom Возраст: 28
```



# Number

Тип `number` представляет числа, причем все числа в TypeScript, как и в JavaScript, являются числами с плавающей точкой. Поэтому с помощью данного типа можно определять как целые числа, так и числа с плавающей точкой:

```
1 let age: number = 36;  
2 let height: number = 1.68;
```



В Typescript поддерживает все типы, которые есть в Javascript, дополняя их несколькими дополнительными типами

Рассмотрим основные типы:

## boolean

Тип **boolean** представляет логическое значение **true** или **false**:

```
1 let isEnabled: boolean = true;
2 let isAlive: boolean = false;
3
4 console.log(isEnabled);
5 console.log(isAlive);
```

True



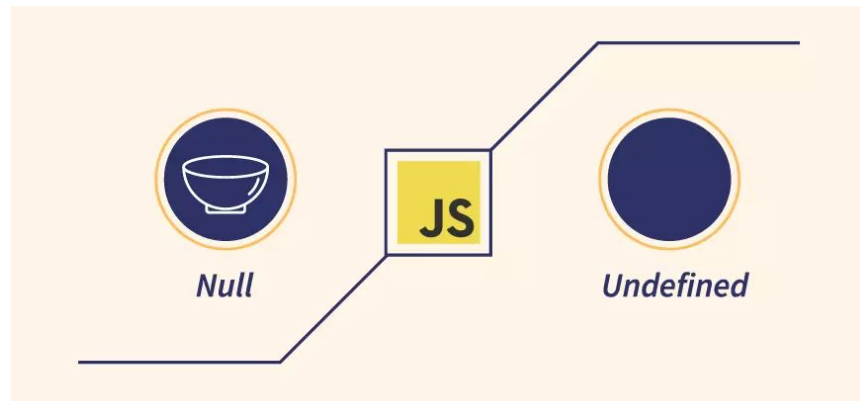
False



# null и undefined

null и undefined сами по себе они могут быть значениями для переменных и типами

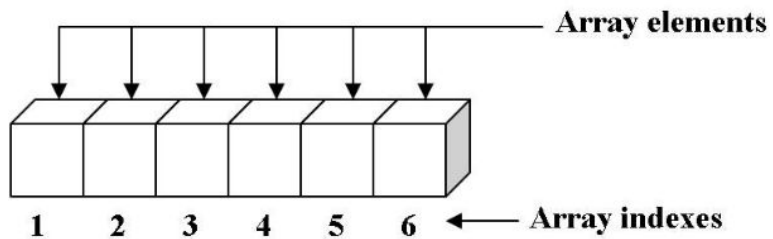
```
let a: null = null;  
let b: undefined = undefined;
```



# Array

Тип Array используется для указания элементов массива.

```
1 let fruits: string[] = ['Apple', 'Orange', 'Banana'];
```



# tupel

Этот тип ещё называют кортежем. Он позволяет определить массив фиксированной длины с элементами разных типов.

```
let person: [string, number] = ["John", 25];
```

# void

Используется для функций, которые не возвращают значения.

```
function sayHello(): void {  
    console.log("Hello!");  
}
```

# any

Any описывает данные, тип которых может быть неизвестен на момент написания приложения.

```
1 let someVar: any = "hello";  
2 console.log(someVar);    // сейчас someVar - это string  
3 someVar = 20;  
4 console.log(someVar);    // сейчас someVar - это number
```

var x: number ✓

var x: any ✗

var x: string ✓

var x: boolean ✓



# Ключевые особенности TypeScript

## 1. Явная аннотация типа (Explicit Type Annotation):

TypeScript позволяет явно указывать типы переменных, функций и других элементов кода. Это позволяет обнаруживать ошибки на этапе компиляции, что может сделать код более надежным и поддерживаемым.

```
function add(a: number, b: number): number {  
    return a + b;  
}  
  
let result: number = add(5, 10);
```



# Ключевые особенности TypeScript

## 2. Вывод типа (Type Inference):

TypeScript автоматически определяет тип на основе контекста, если явная аннотация типа не указана.

```
let age = 25; // TypeScript автоматически определит тип number

function greet(person: string) {
    console.log("Hello, " + person + "!");
}
```

# Ключевые особенности TypeScript

## 3. Union типы:

Позволяют объединять или пересекать несколько типов.

```
type StringOrNumber = string | number;  
  
let value: StringOrNumber = "hello";
```

# Ключевые особенности TypeScript

## 4. Интерфейсы (Interfaces):

TypeScript позволяет создавать пользовательские типы данных с помощью интерфейсов.

```
interface Person {  
    name: string;  
    age: number;  
}  
  
let person: Person = { name: "John", age: 25 };
```

# Ключевые особенности TypeScript

## 5. Типы (Type Aliases):

Позволяют создавать именованные типы

```
type Point = { x: number; y: number };  
  
function move(point: Point): Point {  
    // some logic  
    return { x: point.x + 1, y: point.y + 1 };  
}
```

# Разница между interface и type

## Interface

- Интерфейсы могут объединяться (мержиться), если они имеют одинаковые имена.
- Интерфейсы могут расширяться (extends), что позволяет использовать один интерфейс как базовый для другого.



```
// Исходный интерфейс
interface Car {
    brand: string;
    speed: number;
}

// Объединение интерфейсов (добавление нового свойства)
interface Car {
    color: string;
}


// Теперь интерфейс Car объединен с новым свойством color
let myCar: Car = {
    brand: "Toyota",
    speed: 120,
    color: "blue",
};
```

```
interface Employee extends Person {
    role: string;
}
```

# Разница между interface и type

## Type

- Типы не объединяются автоматически. Если вы объявляете тип с тем же именем, это приведет к ошибке.
- Типы могут быть использованы с оператором & для создания объединенного (intersection) типа.



```
type Person = {  
  name: string;  
  age: number;  
};  
  
// Ошибка: Duplicate identifier 'Person'.  
type Person = {  
  gender: string;  
};
```



```
type Employee = Person & { role: string };
```

## Задание необязательных параметров в interface и type

В **TypeScript** можно делать параметры необязательными, добавляя к ним знак вопроса ?. Это касается как интерфейсов (**interfaces**), так и типов (**type aliases**). Необязательные параметры означают, что вы можете не предоставлять значение для данного параметра при создании объекта, и TypeScript не будет считать это ошибкой.

```
interface Person {  
    name: string;  
    age?: number; // Возраст необязателен  
}  
  
let person1: Person = { name: "John" };  
let person2: Person = { name: "Jane", age: 25 };
```



# **Ваша новая IT-профессия – Ваш новый уровень жизни**

Программирование с нуля в  
немецкой школе AIT TR GmbH