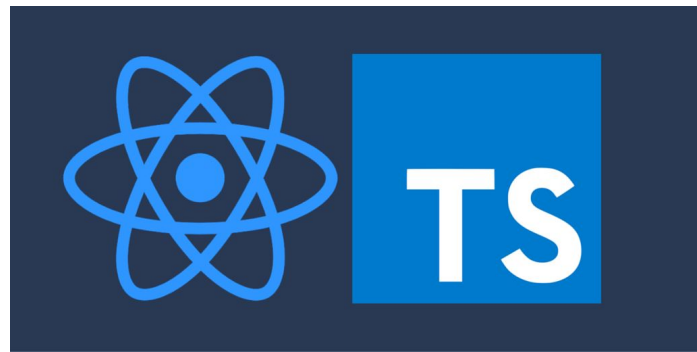


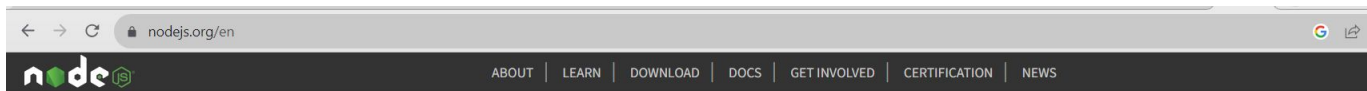
Тема занятия:  
**React: TypeScript**



# Создание React проекта с TypeScript

# Установка NodeJS

1. Перейдите на сайт <https://nodejs.org/en>
2. Скачайте и установите последнюю версию **Node.js**



Node.js® is an open-source, cross-platform JavaScript runtime environment.

Download Node.js®

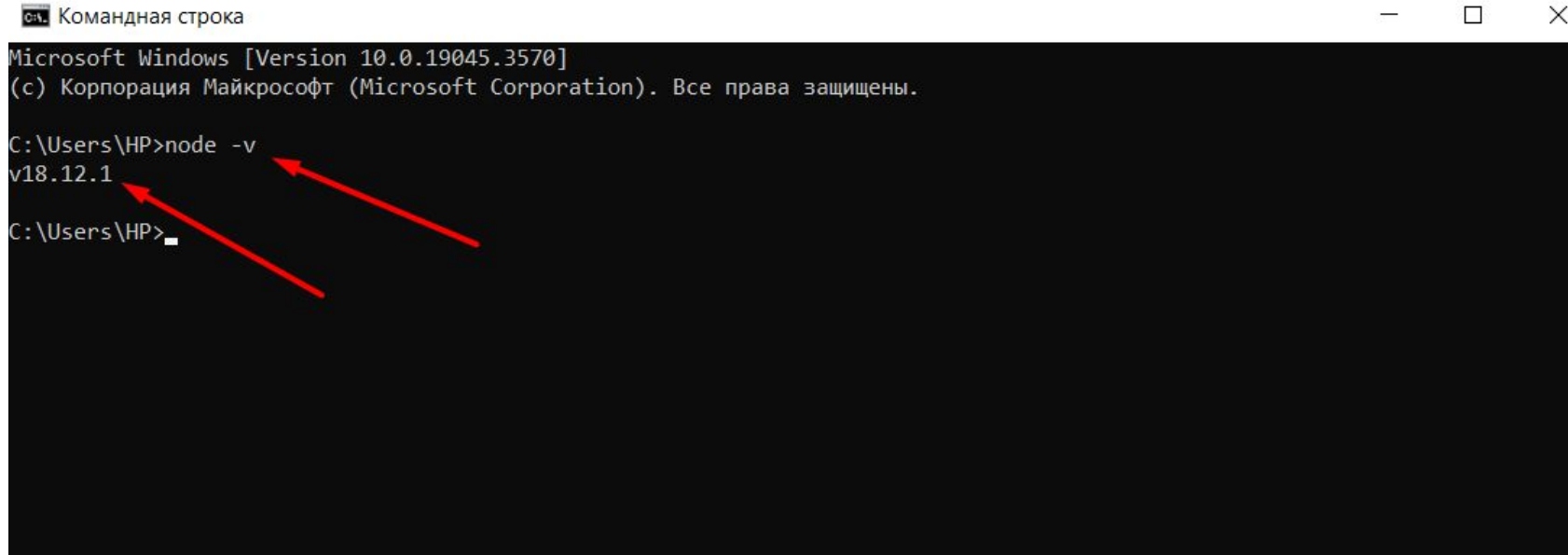


[Other Downloads](#) | [Changelog](#) | [API Docs](#)   [Other Downloads](#) | [Changelog](#) | [API Docs](#)

For information about supported releases, see the [release schedule](#).

# Проверка версии NodeJS

1. Перейдите в командную строку или терминал
2. Введите команду **node -v**

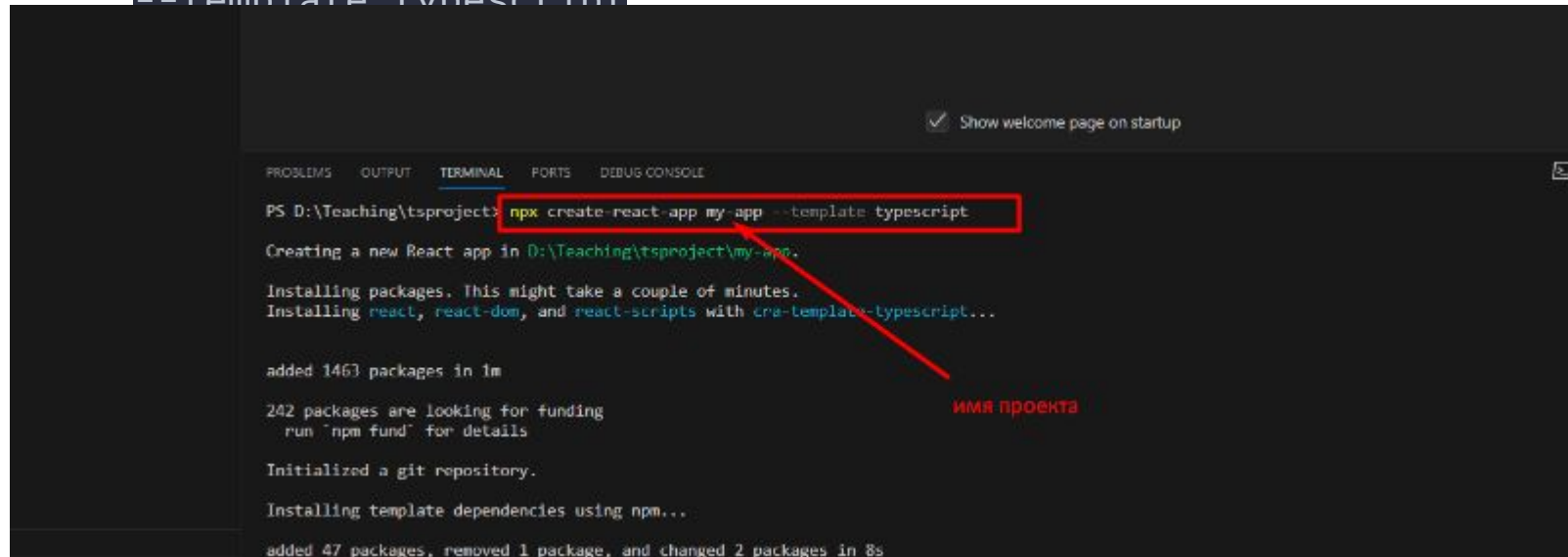


```
Командная строка
Microsoft Windows [Version 10.0.19045.3570]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\HP>node -v
v18.12.1
C:\Users\HP>
```

# Создание проекта

1. Откройте **VSCode**, перейдите в папку, в которой будет лежать ваш проект и в терминале введите следующую команду - `npx create-react-app my-app --template typescript`



The screenshot shows the VS Code interface with the terminal panel open. The command `npx create-react-app my-app --template typescript` is entered in the terminal. A red box highlights the command, and a red arrow points from the text "имя проекта" (project name) to the `my-app` part of the command. The terminal output shows the progress of creating the new React app, including installing packages and initializing a git repository.

```
PS D:\Teaching\tsproject> npx create-react-app my-app --template typescript

Creating a new React app in D:\Teaching\tsproject\my-app.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template-typescript...

added 1463 packages in 1m

242 packages are looking for funding
  run "npm fund" for details

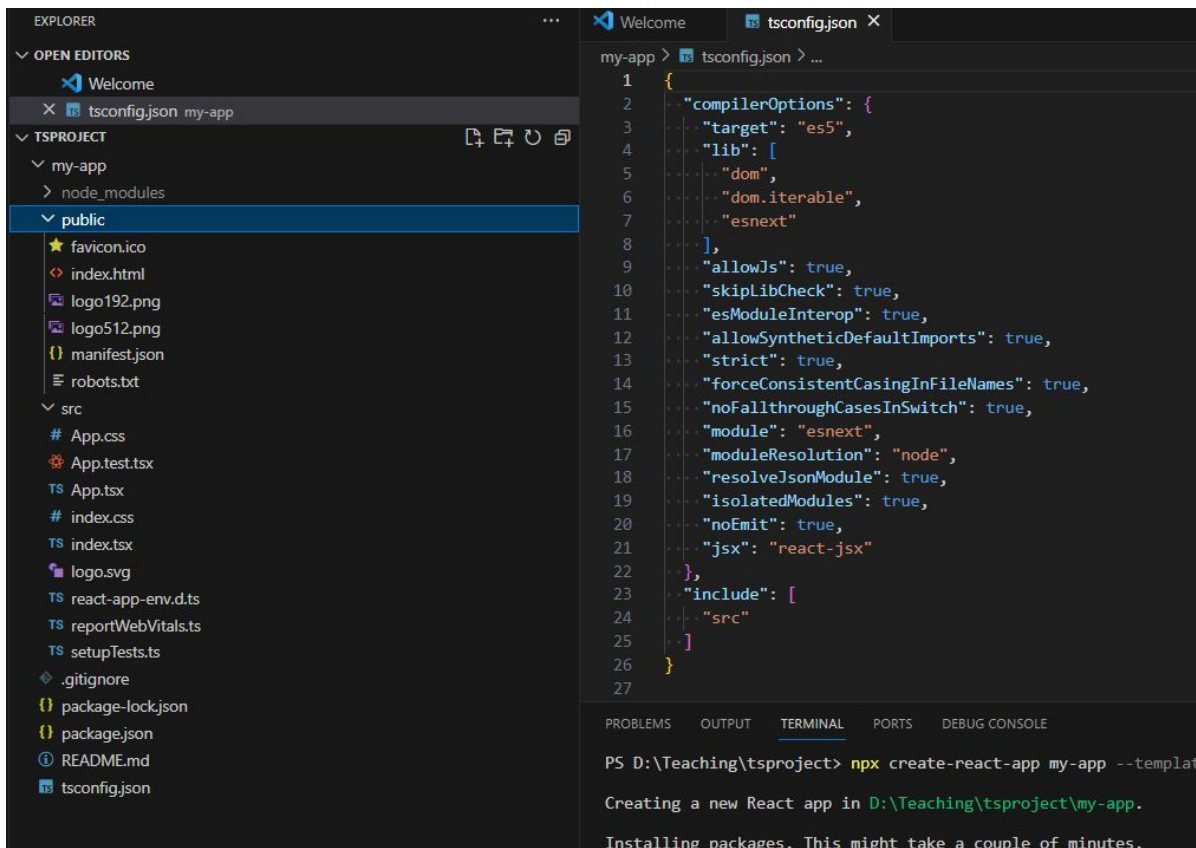
Initialized a git repository.

Installing template dependencies using npm...

added 47 packages, removed 1 package, and changed 2 packages in 8s
```

# Структура проекта my-app

Структура проекта схожа со структурой проекта без TypeScript



The image features decorative geometric patterns in the top-left and bottom-right corners. These patterns consist of overlapping dark blue and light blue shapes, some with gold outlines and others with a dotted texture. The central area is white and contains the text 'TypeScript' and the 'TS' logo.

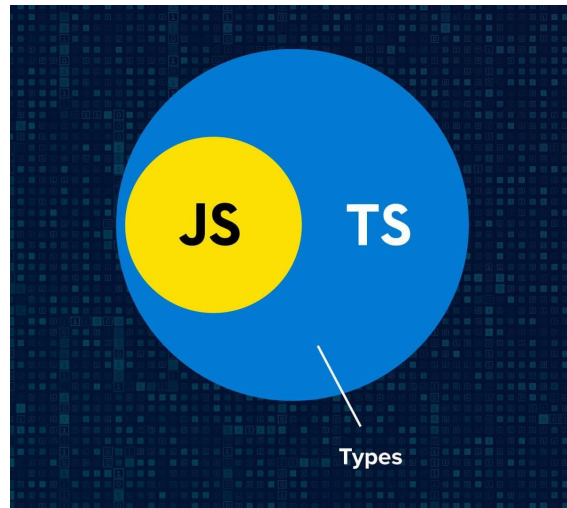
# TypeScript





**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.





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



## 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;
```



# 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
```

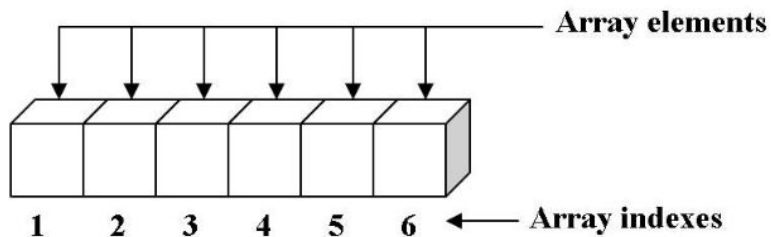
string

E	X	A	M	P	L	E
---	---	---	---	---	---	---

# Array

Тип Array используется для указания элементов массива.

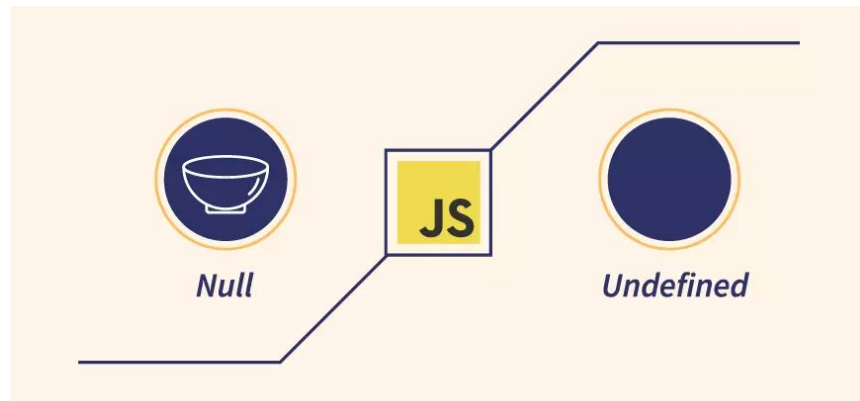
```
1 let fruits: string[] = ['Apple', 'Orange', 'Banana'];
```



# 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;
```



# 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 ✓





# enum

Тип enum или перечисление позволяет определить набор именованных констант, которые описывают определенные состояния.

Для определения перечисления применяется ключевое слово enum.

Например, определение перечисления:

```
enum Season { Winter, Spring, Summer, Autumn };
```

Перечисление называется **Season** и имеет четыре элемента. Теперь используем перечисление:

```
let current: Season = Season.Summer;  
  
console.log(current);           // 2
```

```
enum {  
  key: val,  
  key: val,  
  key: val,  
  key: val,  
}
```



# enum

## Числовые перечисления

По умолчанию константы перечисления, как в примере выше, представляют числовые значения. То есть это так называемое числовое перечисление, в котором каждой константе сопоставляется числовое значение.

```
1 enum Season { Winter, Spring, Summer, Autumn };
```

## Эквивалентно

```
1 enum Season { Winter=0, Spring=1, Summer=2, Autumn=3 };
```

# enum

## Числовые перечисления

Хотя мы можем явным образом переопределить эти значения. Так, мы можем задать значение одной константы, тогда значения следующих констант будет увеличиваться на единицу:

```
1 enum Season { Winter=5, Spring, Summer, Autumn };           // 5, 6, 7, 8
```

Либо можно каждой константе задать свое значение:

```
1 enum Season { Winter=4, Spring=8, Summer=16, Autumn=32 };   // 4, 8, 16, 32
```

# enum

## Строковые перечисления

Кроме числовых перечислений в TypeScript есть строковые перечисления, константы которых принимают строковые значения:

```
enum Season {  
    Winter = "Зима",  
    Spring = "Весна",  
    Summer = "Лето",  
    Autumn = "Осень"  
};  
1  
var current: Season = Season.Summer;  
console.log(current); // Лето
```

Также можно определять смешанные перечисления, константы которых могут числа и строки.

Ошибки несовпадения типов будут заметны на стадии написания кода.



процесс доставки  
JavaScript кода



процесс доставки  
статически типизированного языка

# 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 и Intersection 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 };
```

# Ключевые особенности TypeScript

## 6. Дженерики (Generic Types):

Дженерики (generics) предоставляют средства для создания обобщенных типов и функций. Это позволяет писать более универсальный и безопасный код, который может работать с различными типами данных, не теряя статической типизации

### Пример

```
// Обобщенный тип для массива любого типа
type MyArray<T> = T[];

// Пример использования
const stringArray: MyArray<string> = ['a', 'b', 'c'];
const numberArray: MyArray<number> = [1, 2, 3];
```



# Ключевые особенности TypeScript

## 6. Дженерики (Generic Types):

В дженериках можно задавать Default значения

### Пример

```
// Обобщенный тип с default значением
type MyDefaultArray<T = number> = T[];

// Пример использования
const numberArray: MyDefaultArray = [1, 2, 3];
const stringArray: MyDefaultArray<string> = ['a', 'b', 'c'];
```

# Ключевые особенности TypeScript

## 6. Дженерики (Generic Types):

Дженерики можно использовать в интерфейсах

### Пример

```
// Обобщенный интерфейс для объекта с одним свойством
```

```
interface KeyValue<T> {  
  key: string;  
  value: T;  
}
```

```
// Пример использования
```

```
const stringKeyValue: KeyValue<string> = { key: 'name', value: 'John' };  
const numberKeyValue: KeyValue<number> = { key: 'age', value: 30 };
```