# Documentação TypeScript

A linguagem TypeScript surge como um superset do JavaScript, adicionando a este funcionalidades que nativamente não estão disponíveis ou requerem grande esforço para utilização, como tipagem de dados e Orientação a Objetos. Conhecer essa nova opção de desenvolvimento é útil para aqueles que utilizam JavaScript intensamente em seus projetos e desejam construir códigos com melhor arquitetura, aplicando padrões de projeto e práticas comumente encontradas em outras linguagens orientadas a objetos.

<https://www.devmedia.com.br/introducao-ao-typescript/36729>

## O que é o TypeScript?

Criada pela Microsoft, TypeScript está provando ser uma escolha comum entre os desenvolvedores ASP.NET. Não se trata, na verdade, de uma linguagem completamente nova, mas sim um superset (ou superconjunto) do JavaScript.

 A POO sempre foi um problema ao ser aplicada em JavaScript, devido a sua sintaxe não permitir escrever classes, por exemplo, de forma tão clara, além da fraca tipagem de dados.

## ECMAScript 6

## Orientação a Objetos

## Encapsulamento

## Herança

## Abstração

## Polimorfismo

## Trabalhando com o TypeScript

## Tipos de dado: Any

## Let, Var e Const

## Generics

## Modules

## Interfaces

## TypeScript Playground

## Cross-browser

## TypeScript no Visual Studio

## Utilizando tipagem estática

## TypeScript com AngularJS

## Criando a Interface

### Boolean

O tipo boolean ou booleano é o tipo mais básico presente no TypeScript, pois ele é utilizado para armazenar apenas os valores true ou false, como vemos no **Código 1**.

let powerYourDreams: boolean = true;

**Código 1**. Exemplo de boolean

Acima é possível ver a sintaxe para a declaração de uma varível do tipo boolean. Começamos com o let, que é uma das palavras-chave do TypeScript para declaração de varível. Em seguida, temos o nome da váriável (powerYourDreams) e o tipo da variável separado pelo sinal (:). Por fim, após o sinal de (=) atribuímos um valor a variável.

Outra forma de declarar uma variável no TypeScript é utilizando a palavra-chave var, porém, por ser mais segura, o let é hoje a palavra-chave padrão para declarar variáveis no TypeScript e no JavaScript

### Number

Assim como no JavaScript, todos os números no TypeScript são valores de ponto flutuante e recebem o tipo number. Além de suportar valores decimais e hexadecimais, o tipo number também suporta valores octais e binários que foram introduzidos no ECMAScript 2015, como vemos no **Código 2**.

A Ecma International é responsável pela padronização da linguagem JavaScript que usa a ECMAScript, através da especificação ECMA-262 e ISO/IEC 16262. A especificação é usada em tecnologias executadas no cliente.

let decimal: number = 2001;

let hexadecimal: number = 0x7d5; //2005

let binario: number = 0b11111011101; //2013

let octal: number = 0o3744; //2020

**Código 2**. Declaração de números do tipo number

### String

Parte fundamental do trabalho para quem desenvolve programas JavaScript para Web ou Servidores é o uso de dados textuais. Assim como em outras linguagens, o TypeScript utiliza o tipo string para se referir a conteúdos textuais. O TypeScript dá suporte ao uso de aspas simples (') e aspas duplas (") na declaração de variáveis do tipo string, do mesmo modo que o JavaScript. Observe um exemplo no **Código 3**.

let cor: string = "green";

**Código 3**. Declaração de uma string com aspas duplas

Também é possível utilizar templates, que podem possuir strings com múltiplas linhas e expressões. Para isso é necessário que o conteúdo esteja envolvido pelo caractere apóstrofo/acento grave (`) e as expressões devem usar a sintaxe ${ expressao }. Confira o exemplo no **Código 4**.

let texto: string = `sala de TypeScript`;

let frase: string = `Seja Bem-vindo à nova ${ texto }`;

console.log(frase)

//O resultado será 'Seja Bem-vindo à nova sala de TypeScript'

**Código 4**. Exemplos de declaração de expresões

### Array

Da mesma forma que no JavaScript, o TypeScript também permite o uso de array de valores e eles podem escritos de duas formas, apresentadas nos **Códigos 5** e **6**.

let numeros: number[] = [3, 6, 0];

**Código 5**. Declarando array com o uso de []

let numeros: Array<number> = [3, 6, 0];

**Código 6**. Declarando array com o uso da palavra array

É possível ver acima que existem duas formas diferentes de se declarar uma variável do tipo Array. No **Código 5**, declaramos o tipo Array utilizando colchetes ([]) ao lado do tipo number indicando que a váriavel deve receber apenas arrays numéricos.

No **Código 6**, no lugar dos colchetes, utilizamos o tipo genérico Array<number> para definir o tipo da variável, mas apesar das diferenças de sintaxe, o resultado final para os dois modos é o mesmo.

Agora que vimos os quatro tipos mais utilizados, veremos os demais tipos básicos disponíveis no TypeScript.

### Tuple

O tipo tuple permite declarar uma variável do tipo array cujo o tamanho e o tipo de dado de cada índice é conhecido pelo desenvolvedor. Veja o exemplo abaixo no **Código 7**:

//Variavel do tipo tuple onde o primeiro valor é uma string e o segundo é um number

let x: [string, number];

// OK - Pois o primeiro parâmetro é string e o segundo number

x = ["Seamus", 10];

// Erro - Pois o primeiro parâmetro é number e o segundo string

x = [10, "Seamus"]; // Erro

**Código 7**. Declarando array com o tipo tuple

Como pode ser visto acima, o tipo tuple pode receber tipos diferentes em cada índice do array.

### Enum

Uma adição muito interessante ao padrão de tipos do JavaScript foi o enum, que assim como no C#, permite dar nomes mais amigáveis a valores numéricos, como vemos nos exemplos do **Código 8**.

//Cada item representa um índice. Vermelho = 0, Verde = 1, Azul = 2

enum Cor {Vermelho, Verde, Azul}

//o valor da variável corFundo será '1'.

let corFundo: Cor = Cor.Verde;

**Código 8**. Declarações amigáveis com enum

### Any

As vezes durante o processo de desenvolvimento, precisamos declarar variáveis que não conhecemos o tipo de dado, ou que o dado pode ser variável, como o retorno de alguma API externa, por exemplo. Nesse caso, utilizamos o any, que vai aceitar como dado qualquer um dos tipos básicos do TypeScript, como vemos no exemplo do **Código 9**.

let naoIdentificado: any = 4;//Essa variável é um numero.

naoIdentificado = "Agora é uma string";

naoIdentificado = false; //Agora é um boolean

**Código 9**. Exemplo de variáveis que não conhecemos o tipo

### Void

O tipo void pode ser considerado o oposto do any, pois ele se abstém de qualquer tipo. Normalmente o tipo void é visto em métodos que não possuem retorno, como vemos nos exemplos do **Código 10**.

function aviso(): void {

console.log("Esta é uma mensagem de aviso!");

}

**Código 10**. Exemplo do tipo void

Também é possível associar o tipo void a variáveis, porém, seu uso não é muito útil, uma vez que uma variável void pode receber apenas dados do tipo null e undefined.

No caso da variável void receber apenas dados do tipo null, deve-se considerar que --strictNullChecks esteja desabilitado.

### Null e Undefined

No TypeScript, tanto o Null quanto Undefined possuem tipos associados a eles e são chamados pelos seus respectivos nomes, como vemos no **Código 11**.

//Não há muito o que fazer com esse tipo de variável

let u: undefined = undefined;

let n: null = null;

**Código 11**. Tipos null e undefined

Por padrão, o tipo null é um subtipo dos outros tipos como, por exemplo, o tipo number e pode ser associado a eles. Quando --strictNullChecks está habilitado, null e undefined só podem ser associados a seus respectivos tipos e ao tipo any. A única exceção a essa regra é que o tipo null também pode ser associado ao tipo void. Caso --strictNullChecks esteja desabilitado, o tipo null poderá receber tanto o valor null quanto undefined.

### Never

O tipo never representa o tipo dos valores que nunca acontecem. Por exemplo, never é o tipo de retorno para a expressão de uma função ou de uma arrow function, que sempre lança uma exceção ou uma que nunca retorna, como um loop infinito por exemplo.

O tipo never é um subtipo de todos os tipos e pode ser associado a cada um deles, entretanto, nenhum dos tipos, incluindo o tipo any, é um subtipo de never e nem pode ser associado a ele, com exceção do próprio never.

Alguns exemplos de uso do never podem ser vistos nos **Códigos 12** a **14**.

// Funções que retornam 'never' devem possuir um final inalcançável

function error(message: string): never {

throw new Error(mensagem);

}

**Código 12**. Exemplo com o tipo never de final inalcançável

// O tipo inferido é never

function fail() {

return error("Algo falhou");

}

**Código 13**. Exemplo com o tipo never do tipo inferido

// Funções que retornam 'never' devem possuir um final inalcançável

function infiniteLoop(): never {

while (true) {

}

}

**Código 14**. Exemplo com o tipo never no retorno da função

### Object

O tipo object representa os tipos não-primitivos, por exemplo: qualquer tipo que não seja number, string, boolean, bigint, symbol, null ou undefined.

Com o tipo object, APIs como Object.create podem ser representadas de uma forma melhor, como mostra o **Código 15**.

declare function create(o: object | null): void;

create({ prop: 0 }); // OK

create(null); // OK

create(42); // Erro

create("string"); // Erro

create(false); // Erro

create(undefined); // Erro

**Código 15**. Exemplo com o tipo object

### Conclusão

Através dos exemplos apresentados, vimos que a criação de variáveis e métodos dos mais **diversos tipos disponíveis no TypeScript** se dá de por meio de um código mais organizado, proporcionando maior facilidade na hora de manter e testar os códigos.