# Introdução à Tipagem Básica em TypeScript

Por Escola Dnc

# Introdução

O TypeScript é uma linguagem de programação que estende o JavaScript, adicionando recursos de tipagem estática. Este ebook explora os conceitos fundamentais de tipagem no TypeScript, usando como contexto um sistema de cadastro de pacientes em um hospital. Vamos abordar os tipos básicos, variáveis, arrays e boas práticas de tipagem.

# Fundamentos do TypeScript

## Let vs Const

Antes de mergulharmos nos tipos específicos do TypeScript, é importante relembrar dois conceitos fundamentais do JavaScript:

- **let**: Permite que o valor da variável seja alterado após a declaração.
- **const**: Declara uma constante cujo valor não pode ser alterado após a atribuição inicial.

```
let nome = "Matheus";nome = "Luciano"; // Permitidoconst idade = 25;idade = 30; // Erro! Não é possível reatribuir uma constante
```

**Importante**: Embora essa distinção venha do JavaScript, o TypeScript herda e reforça esse comportamento, proporcionando maior segurança ao código.

## Tipos Primitivos

O TypeScript oferece vários tipos primitivos que são essenciais para a tipagem básica:

1. **string**: Usado para texto

```
const nomePaciente: string = "Matheus";
```

2. **number**: Usado para números (inteiros ou decimais)

```
const idadePaciente: number = 25;
```

3. **boolean**: Usado para valores verdadeiro/falso

```
const internado: boolean = false;
```

4. **null**: Representa a ausência intencional de valor

```
const quarto: null = null;
```

5. **bigint**: Usado para números inteiros muito grandes

```
const registro: bigint =
12345678901234567890123456789012345678901234567890n;
```

## Dica de Tipagem

Ao trabalhar com números que não serão usados em cálculos (como números de telefone, CPF, ou números de casa), é recomendável usar o tipo `string`. Isso evita problemas com formatação e preserva zeros à esquerda.

```
const telefone: string = "011999887766";const cpf: string =
"123.456.789-00";
```

## Arrays e Coleções

O TypeScript permite a criação de arrays tipados, garantindo que todos os elementos sejam do mesmo tipo.

### Arrays de Tipos Primitivos

1. **Array de strings**

```
const numerosTelefone: string[] = ["11999887766",
"11988776655"];
```

2. **Array de numbers**

```
const idadePacientes: number[] = [25, 30, 45, 60];
```

3. **Array de booleans**

```
const statusInternacao: boolean[] = [true, false, true, false];
```

### Sintaxe Alternativa

Você também pode usar a sintaxe genérica para declarar arrays:

```
const numerosTelefone: Array<string> = ["11999887766",
"11988776655"];
```

## Benefícios da Tipagem em Arrays

A tipagem forte em arrays oferece várias vantagens:

- **Prevenção de erros**: O TypeScript impede a inserção de elementos de tipos diferentes no array.
- **Autocompletar e IntelliSense**: IDEs podem oferecer sugestões mais precisas baseadas no tipo do array.
- **Documentação implícita**: O tipo do array serve como documentação sobre o que ele deve conter.

## Boas Práticas e Dicas

1. **Use tipos específicos**: Evite usar `any` sempre que possível. Tipos específicos ajudam a prevenir erros e melhoram a legibilidade do código.
2. **Aproveite a inferência de tipos**: O TypeScript é capaz de inferir tipos em muitos casos. Use isso a seu favor para escrever código mais conciso.

```
let nome = "Matheus"; // TypeScript infere que nome é uma
string
```

3. **Combine tipos quando necessário**: Use union types para variáveis que podem ter mais de um tipo.

```
let idadeOuNome: number | string;idadeOuNome = 25;     //
VálidoidadeOuNome = "João"; // Também válido
```

4. **Use enums para conjuntos fixos de valores**:

```
enum StatusPaciente {  Internado,  Ambulatorial,  Alta}let
status: StatusPaciente = StatusPaciente.Internado;
```

5. **Aproveite os tipos literais**: Para valores que só podem ser um conjunto específico de strings ou números.

```
let tipoSanguineo: "A+" | "A-" | "B+" | "B-" | "AB+" | "AB-" |
"O+" | "O-";
```

## Aplicação Prática: Sistema de Cadastro de Pacientes

Vamos aplicar os conceitos aprendidos em um exemplo prático de um sistema de cadastro de pacientes:

```
// Definindo tipostype Paciente = {  nome: string;  idade: number;  internado: boolean;  quarto: number | null;  numerosTelefone: string[];  tipoSanguineo: "A+" | "A-" | "B+" | "B-" | "AB+" | "AB-" | "O+" | "O-";};// Criando um pacienteconst novoPaciente: Paciente = {  nome: "Maria Silva",  idade: 45,  internado: true,  quarto: 302,  numerosTelefone: ["11999887766", "11988776655"],  tipoSanguineo: "O+"};// Função para registrar internaçãofunction registrarInternacao(paciente: Paciente, quarto: number): void {  paciente.internado = true;  paciente.quarto = quarto;  console.log(`Paciente ${paciente.nome} internado no quarto ${quarto}`);}// Função para dar altafunction darAlta(paciente: Paciente): void {  paciente.internado = false;  paciente.quarto = null;  console.log(`Paciente ${paciente.nome} recebeu alta`);}// Usando as funçõesregistrarInternacao(novoPaciente, 305);darAlta(novoPaciente);
```

Este exemplo demonstra como usar tipos complexos, unions, e funções tipadas para criar um sistema robusto e seguro.

## Conclusão

A tipagem básica do TypeScript oferece uma base sólida para o desenvolvimento de aplicações mais seguras e manuteníveis. Ao compreender e aplicar corretamente os tipos primitivos, arrays, e boas práticas de tipagem, você estará preparado para enfrentar desafios mais complexos e aproveitar todo o potencial que o TypeScript tem a oferecer.

Lembre-se: a prática leva à perfeição. Continue explorando e aplicando esses conceitos em seus projetos para solidificar seu conhecimento e melhorar suas habilidades de desenvolvimento com TypeScript.