



Intro

Prof. Dr. Anderson Rodrigues



Introdução ao TypeScript

- Desenvolvido pela Microsoft
- Objetivos:
 - Aumento da escalabilidade
 - Robustez (resiliência a erros e situações inesperadas durante a execução)
 - E fácil manutenção em projetos de grande porte



Principais características do TypeScript

- Superset do JavaScript
 - Qualquer código JS é automaticamente válido no TypeScript.
- Sistema de tipos estático
 - Permite que os tipos de dados e retornos de função sejam especificados.
- Compilação para JS
 - Permite que os desenvolvedores compilem seus programas para versões atuais e mais antigas.
- Full OOP
 - Atende todos os requisitos de uma linguagem orientada a objeto.

Instalação do Node.js: Definição e características

- O que é ?
 - Ambiente de execução JavaScript do lado do servidor
- Assincronismo e Event-driven
 - Projetado para operações assíncronas e orientado a eventos. Isso significa que ele pode lidar com muitas conexões simultâneas sem a necessidade de threads adicionais, resultando em aplicações escaláveis e eficientes.
- V8 Engine
 - Utiliza o V8, o mecanismo de JavaScript de código aberto do Google Chrome, que é conhecido por ser rápido e eficiente na execução de código JavaScript.
- Módulos nativos e NPM
 - Possui seu próprio sistema de módulos e usa o Node Package Manager (NPM) para gerenciar pacotes e dependências.

Tipos de dados no TypeScript

Tipo	Descrição
<i>string</i>	valores alfanuméricos
<i>number</i>	valores numéricos inteiros e de ponto flutuante
<i>boolean</i>	valores lógicos (true ou false)
<i>any</i>	qualquer valor
<i>null</i>	valor nulo
<i>void</i>	sem valor
<i>never</i>	representa um conjunto de valores que nunca ocorre
<i>Array</i>	estruturas multidimensionais onde os elementos do mesmo tipo são acessados através de índices inteiros.
<i>enum</i>	estrutura composta que representa um conjunto de constantes nomeadas vinculadas a valores inteiros
<i>object</i>	usado para representar qualquer valor não-primitivo. Isso inclui objetos, arrays, funções e instâncias de classes
<i>tuple</i>	permite representar uma sequência fixa de elementos com tipos conhecidos, onde cada elemento pode ter um tipo diferente.

Tipos primitivos do TypeScript

Textos: *string*

typescript

Copy code

```
let color: string = "blue";  
let sentence: string = `The color is ${color}`;
```

Numéricos: *number*

typescript

Copy code

```
let decimal: number = 6;  
let hex: number = 0xf00d;  
let binary: number = 0b1010;  
let octal: number = 0o744;
```

Lógicos: *boolean*

typescript

Copy code

```
let isDone: boolean = false;
```

Tipos primitivos no TypeScript

Qualquer valor: *any*

typescript

Copy code

```
let notSure: any = 4;
notSure = "maybe a string instead";
```

Nulo ou desconhecido: *null ou undefined*

typescript

Copy code

```
let u: undefined = undefined;
let n: null = null;
```

Vazio: *void*

typescript

Copy code

```
function logMessage(): void {
  console.log("This is a log message");
}
```

Nunca: *never*

typescript

Copy code

```
function throwError(message: string): never {
  throw new Error(message);
}
```

Tipos complexos no TypeScript

Objeto: *object*

typescript

Copy code

```
let obj: object = { key: "value" };
```

Multidimensionais: *Array*

typescript

Copy code

```
let numbers: number[] = [1, 2, 3];  
let fruits: Array<string> = ["apple", "banana"];
```

Tupla: *tuple*

typescript

Copy code

```
let x: [string, number];  
x = ["hello", 10];
```

Enumerado: *enum*

typescript

Copy code

```
enum Color {  
  Red,  
  Green,  
  Blue,  
}  
let c: Color = Color.Green;
```


Operadores no TypeScript

Atribuição:

```
let x = 10;           x *= 4; //saída 8
x += 2; //saída 12    x /=2; //saída 4
x -= 10; //saída 2
```

Aritméticos:

```
let soma = 5 + 3; //saída 8
let subtracao = 10 - 5; //saída 5
let produto = 4 * 2; //saída 8
let divisao = 8 / 2; //saída 4
let resto = 15 % 4; //saída 1
let potencia = 8 ** 2; //saída 64
```

Relacionais:

```
let igual_a = 10 === 10; //saída true
let nao_igual_a = 5 !== "5"; //saída false
let maior_que = 5 > 10; //saída false
let menor_que = 5 < 10; //saída true
let maior_igual_a = 5 >= 5 //saída true
let menor_igual_a = 5 <= 5 //saída true
```

Incremento:

```
let x = 10;
x++; //saída 11
x--; //saída 10
```

Estruturas de controle condicionais no TypeScript

if (condicao) { } else { }:

```
let condition: boolean = true;

if (condition) {
    // Código a ser executado se a condição for verdadeira
} else {
    // Código a ser executado se a condição for falsa
}
```

switch (option) { }:

```
let option: number = 2;

switch (option) {
    case 1:
        // Código a ser executado se option for 1
        break;
    case 2:
        // Código a ser executado se option for 2
        break;
    default:
        // Código a ser executado se option não corresponder a nenhum caso
}
```

Estruturas de controle repetição no TypeScript

while (condicao de parada) { }:

```
let count: number = 0;

while (count < 5) {
  // Código a ser repetido enquanto a condição for verdadeira
  count++;
}
```

do { } while (condicao de parada):

```
let count: number = 0;

do {
  // Código a ser repetido pelo menos uma vez e continuado enquanto a condição for v
  count++;
} while (count < 5);
```

Estruturas de controle repetição no TypeScript

for () { }:

```
for (let i = 0; i < 5; i++) {  
  // Código a ser executado em cada iteração do loop  
}
```

for (in) { }:

```
let obj = { a: 1, b: 2, c: 3 };  
  
for (let key in obj) {  
  // Código a ser executado para cada propriedade do objeto  
  console.log(key, obj[key]);  
}
```

for (of) { }:

```
let arr = [1, 2, 3];  
  
for (let value of arr) {  
  // Código a ser executado para cada elemento do array  
  console.log(value);  
}
```

Comandos de quebra de controle de fluxo no TypeScript

break e continue:

- O **break** é utilizado para sair de um loop.
- O **continue** é utilizado para pular para a próxima iteração de um loop.

```
for (let i = 0; i < 5; i++) {  
  if (i === 2) {  
    break; // Sai do loop quando i é igual a 2  
  }  
  
  if (i === 1) {  
    continue; // Pula para a próxima iteração quando i é igual a 1  
  }  
  
  // Código a ser executado em cada iteração  
}
```

Exercícios: TypeScript

1. Crie um programa em TypeScript que verifica se um número é par ou ímpar. O usuário deve inserir um número, e o programa deve exibir uma mensagem indicando se o número é par ou ímpar.
2. Crie um programa em TypeScript que solicita a idade do usuário e verifica se ele é maior de idade (idade maior ou igual a 18 anos).
3. Crie um programa em TypeScript usando a estrutura **switch** que dado um número inteiro seja impresso o dia da semana que ele representa. Ex: 1, imprime `segunda-feira`.
4. Crie um programa que use a estrutura de repetição **while** para imprimir os números pares de 2 a 10.
5. Crie um programa que use a estrutura de repetição **do...while** para solicitar ao usuário que insira um número maior que 5. O programa deve continuar solicitando até que o usuário insira um número maior que 5.
6. Crie um programa que use a estrutura de repetição **for** para calcular a soma dos números de 1 a 5.
7. Crie um programa que use a estrutura de repetição **for...in** para percorrer as propriedades de um objeto e imprimir o nome e o valor de cada propriedade.
8. Crie um programa que use a estrutura de repetição **for...of** para percorrer os elementos de um array e imprimir cada elemento multiplicado por 2.