

INTRODUCCION AD TYPESCRIPT

o QUE É o
TVPRESSCRIPT?



O QUE É O TYPESCRIPT?

TypeScript é uma **extensão** do JavaScript que adiciona **tipagem estática** para variáveis, parâmetros e retornos, sem substituir o que já existe na linguagem.

A NECESSIDADE E A HISTÓRIA DO TYPESCRIPT

TypeScript foi criado pela **Microsoft** em 2012 para resolver as **limitações do JavaScript**, especialmente em aplicações maiores. Ele adiciona **tipagem estática** ao JavaScript, permitindo que você especifique explicitamente o **tipo de dado** que cada variável, função ou objeto pode conter.

A NECESSIDADE E A HISTÓRIA DO TYPESCRIPT

Benefícios do TypeScript

- Detecção de erros antecipada: Os erros de tipo são detectados enquanto você escreve o código.
- Melhor manutenção: O código fica mais claro e organizado.
- Compatibilidade: TypeScript é um superconjunto do JavaScript, o que significa que todo código JavaScript válido também é válido em TypeScript.

TIPOS NO
JAVASCRIPT

TIPOS NO JAVASCRIPT

No JavaScript, os tipos de dados representam diferentes tipos de valores que podemos armazenar em variáveis ou manipular em nossos programas. Esses tipos são:



```
1 let nome = "Wallace";
```

TIPOS NO JAVASCRIPT

No JavaScript, os tipos de dados representam diferentes tipos de valores que podemos armazenar em variáveis ou manipular em nossos programas. Esses tipos são:

String: Representa texto, sempre entre aspas simples (') ou duplas (").



```
1 let nome = "Wallace";
```


TIPOS NO JAVASCRIPT

No JavaScript, os tipos de dados representam diferentes tipos de valores que podemos armazenar em variáveis ou manipular em nossos programas. Esses tipos são:



```
1 let idade = 25;
```

TIPOS NO JAVASCRIPT

No JavaScript, os tipos de dados representam diferentes tipos de valores que podemos armazenar em variáveis ou manipular em nossos programas. Esses tipos são:

Number: Pode ser qualquer número, inteiro ou decimal.



```
1 let idade = 25;
```

TIPOS NO JAVASCRIPT

No JavaScript, os tipos de dados representam diferentes tipos de valores que podemos armazenar em variáveis ou manipular em nossos programas. Esses tipos são:



```
1 let gostaDeProgramar = true;
```

TIPOS NO JAVASCRIPT

No JavaScript, os tipos de dados representam diferentes tipos de valores que podemos armazenar em variáveis ou manipular em nossos programas. Esses tipos são:

Boolean: Representa um valor verdadeiro ou falso.



```
1 let gostaDeProgramar = true;
```

TIPOS NO JAVASCRIPT

No JavaScript, os tipos de dados representam diferentes tipos de valores que podemos armazenar em variáveis ou manipular em nossos programas. Esses tipos são:



```
1 let frutas = ["maçã", "banana", "laranja"];
```

TIPOS NO JAVASCRIPT

No JavaScript, os tipos de dados representam diferentes tipos de valores que podemos armazenar em variáveis ou manipular em nossos programas. Esses tipos são:

Array: Um conjunto de valores (de um ou mais tipos).



```
1 let frutas = ["maçã", "banana", "laranja"];
```

TIPOS NO JAVASCRIPT

No JavaScript, os tipos de dados representam diferentes tipos de valores que podemos armazenar em variáveis ou manipular em nossos programas. Esses tipos são:



```
1 let pessoa = { nome: "João", idade: 30 };
```

TIPOS NO JAVASCRIPT

No JavaScript, os tipos de dados representam diferentes tipos de valores que podemos armazenar em variáveis ou manipular em nossos programas. Esses tipos são:

Object: Um conjunto de pares chave-valor.



```
1 let pessoa = { nome: "João", idade: 30 };
```


JAVASCRIPT E SUA TIPAGEM DINÂMICA

No JavaScript, as variáveis podem trocar de tipo durante a execução do programa.
Por exemplo:



```
1 let exemplo = 42; // Number
```

```
2 exemplo = "Agora é uma string"; // String
```

Essa flexibilidade é útil, mas pode causar problemas, como bugs difíceis de detectar.

OS DESAFIOS DA TIPOAGEM PRÁTICA



```
1 let resultado = 10 + "20";  
2 // Resultado: "1020" (concatenação, não soma)
```

OS DESAFIOS DA TIPAGEM PRÁTICA



```
1 let resultado = 10 + "20";  
2 // Resultado: "1020" (concatenação, não soma)
```

Embora a tipagem dinâmica do JavaScript facilite o desenvolvimento rápido, ela tem desvantagens, como:

1. Erros inesperados: É fácil cometer erros ao misturar tipos.

OS DESAFIOS DA TIPAGEM PRÁTICA



```
1 let resultado = 10 + "20";  
2 // Resultado: "1020" (concatenação, não soma)
```

Embora a tipagem dinâmica do JavaScript facilite o desenvolvimento rápido, ela tem desvantagens, como:

2. Dificuldade de manutenção: Em projetos grandes, é difícil lembrar quais tipos de dados cada variável deve ter.

OS DESAFIOS DA TIPAGEM PRÁTICA



```
1 let resultado = 10 + "20";  
2 // Resultado: "1020" (concatenação, não soma)
```

Embora a tipagem dinâmica do JavaScript facilite o desenvolvimento rápido, ela tem desvantagens, como:

3. Ausência de validação antecipada: Problemas de tipos só são descobertos durante a execução, e não enquanto você escreve o código.

INSTALLAND.OO.O

TYPESCRIPT

INSTALANDO O TYPESCRIPT

Para usar TypeScript, você precisa instalá-lo no seu ambiente de desenvolvimento. Aqui está o passo a passo:

1. **Requisitos**

a. Node.js instalado em sua máquina.

2. **Passos para Instalação**

a. Abra o terminal e execute o seguinte comando para instalar o TypeScript globalmente:

b. *npm install -g typescript*

3. **Verifique se a instalação foi bem-sucedida**

a. *tsc --version*

TIPOS NO
TVPRESSCRIPT

Criando seu Primeiro Arquivo TypeScript

1. Crie um arquivo chamado index.ts (arquivos TypeScript sempre têm a extensão .ts).
2. Adicione o seguinte código:



```
1 let mensagem: string = "Hello, TypeScript!";  
2 console.log(mensagem);
```


Compilando para JavaScript

O TypeScript precisa ser transformado em JavaScript para ser executado.

Para isso, use o comando:

```
tsc index.ts
```

Isso gerará um arquivo index.js com o seguinte código JavaScript:



```
1 var mensagem = "Hello, TypeScript!";  
2 console.log(mensagem);
```

Agora, você pode rodar o arquivo JavaScript:

```
node index.js
```

Tipando Variáveis e Funções

Tipagem Simples

Defina os tipos das variáveis para evitar erros:



```
1 let idade: number = 25;
```

```
2 let nome: string = "Alice";
```

```
3 let gostaDeProgramar: boolean = true;
```

Tipando Funções


Especifique os tipos dos parâmetros e do retorno:



```
1 function somar(a: number, b: number): number {  
2     return a + b;  
3 }  
4 console.log(somar(5, 3)); // 8
```

Trabalhando com Objetos

Defina o formato dos objetos para evitar inconsistências:



```
1 interface Pessoa {  
2     nome: string;  
3     idade: number;  
4 }  
5  
6 let aluno: Pessoa = {  
7     nome: "João",  
8     idade: 20,  
9 };
```

CONFIGURAND
O TYPESCRIPT

Pra criar o arquivo de configuração do TypeScript, você precisa gerar o **tsconfig.json**, que define como o TypeScript deve compilar seu projeto. Ele controla coisas como o diretório de saída, o padrão de módulos, e as versões de JavaScript compatíveis.

Passo a Passo para criar o tsconfig.json:

1. Abra o terminal na raiz do seu projeto.
2. Rode o comando:

```
npx tsc --init
```

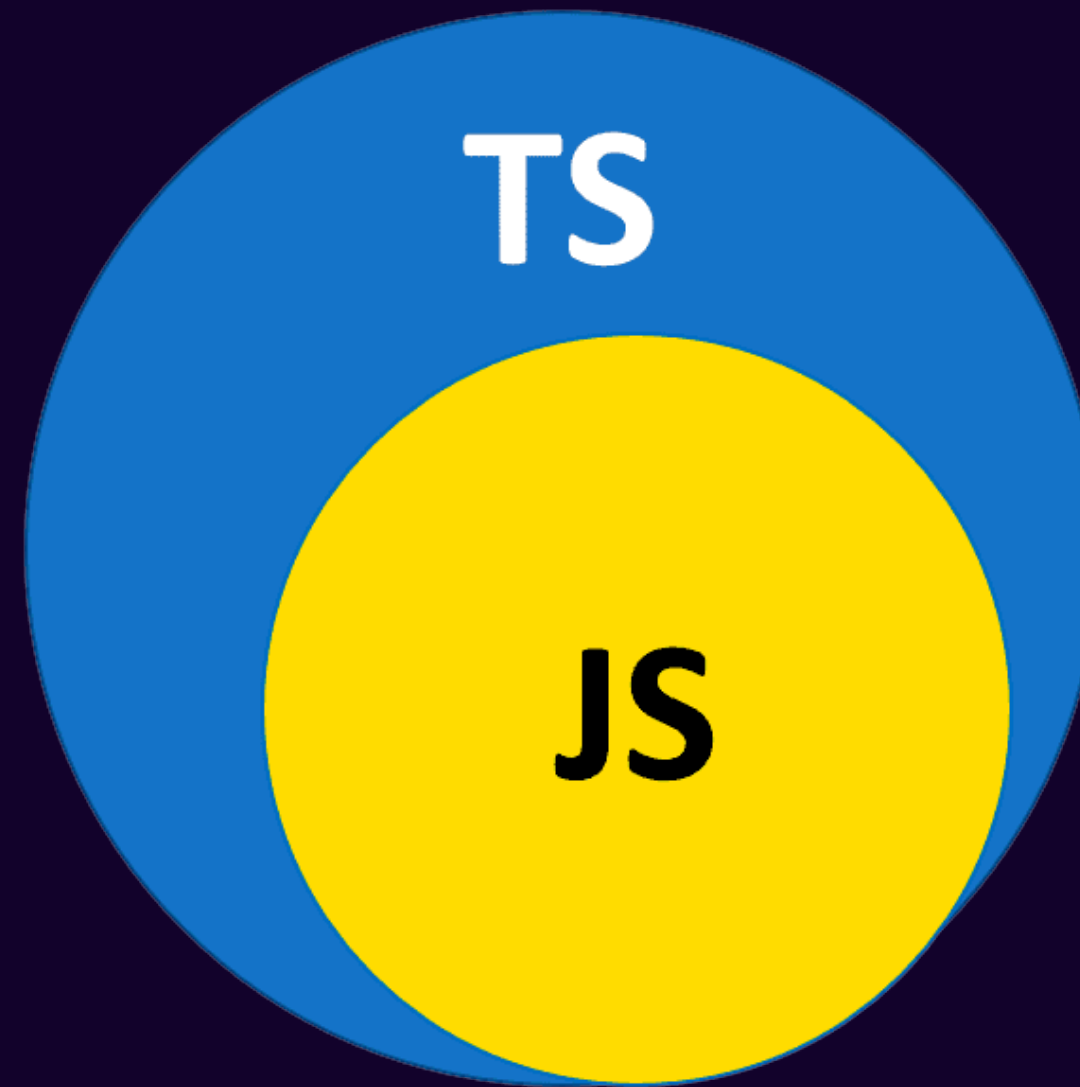
```
1 {
2   "compilerOptions": {
3     "target": "ES6",           // Versão do JavaScript gerado (pode ser ES5, ES6, ES2020, etc.)
4     "module": "commonjs",      // Sistema de módulos (commonjs para Node.js, esnext para ES Modules)
5     "rootDir": "./src",        // Pasta onde ficam os arquivos TypeScript
6     "outDir": "./dist",        // Pasta onde os arquivos compilados serão gerados
7     "strict": true,            // Ativa todas as verificações estritas de tipo
8     "esModuleInterop": true,   // Permite importar módulos CommonJS como ES6 (import express from 'express')
9     "skipLibCheck": true       // Pula verificação de tipos nos arquivos de definição de bibliotecas
10  },
11  "include": ["src/**/*.ts"],  // Inclui todos os arquivos dentro de 'src'
12  "exclude": ["node_modules", "**/*.js"] // Exclui 'node_modules' e arquivos JavaScript da compilação
13 }
```

Explicação rápida das opções principais:

- **target:** Define a versão do JavaScript que o TypeScript vai compilar. Ex: "ES6" gera código compatível com ES6.
- **module:** Define o padrão de módulos que será usado. Use "commonjs" para projetos Node.js ou "esnext" para projetos modernos que usam ES Modules.
- **rootDir:** A pasta onde seus arquivos .ts estão.
- **outDir:** A pasta onde o TypeScript vai colocar os arquivos compilados .js.
- **strict:** Ativa todas as verificações de tipo rigorosas (boa prática para evitar bugs).
- **esModuleInterop:** Permite usar import com bibliotecas que foram feitas para o CommonJS.

CONCLUSÃO

O **TypeScript** é uma ferramenta poderosa que **melhora o desenvolvimento** em JavaScript, especialmente para projetos maiores e mais complexos. Ele traz **clareza** e **organização** ao código, além de **reduzir significativamente os erros**.



EXERCÍCIOS

01

Declaração de Tipos

Declare variáveis com os seguintes tipos:

- Uma string com o valor "TypeScript é incrível".
- Um número com o valor 42.
- Um booleano com o valor true.

02

Função com Tipagem

Crie uma função chamada soma que receba dois números como parâmetros e retorne a soma deles. Certifique-se de tipar corretamente os parâmetros e o retorno.

03

Checando Nulos

Declare uma variável que pode conter um valor do tipo number ou null. Crie uma função que receba essa variável e retorne "Sem valor" se for null ou o número caso contrário.

EXERCÍCIOS

04

Verificação de Booleanos

Escreva uma função chamada verificarMaioridade que receba a idade de uma pessoa (número) e retorne true se for maior ou igual a 18 e false caso contrário. Tipifique os dados corretamente.

05

Concatenando Strings

Crie uma função chamada saudacao que receba o nome de uma pessoa como parâmetro e retorne a mensagem "Olá, [nome]!". A função deve ser estritamente tipada.

06

Operações Matemáticas

Crie uma função chamada calcularAreaCirculo que receba o raio de um círculo (número) como parâmetro e retorne a área do círculo. Use a fórmula: $\text{área} = \pi \cdot \text{raio}^2$

EXERCÍCIOS

07

Tipos Opcionais

Crie uma função chamada `formatarTexto` que receba um texto (string) e um parâmetro opcional chamado `maiuscula` (boolean).

- Se `maiuscula` for `true`, retorne o texto em letras maiúsculas.
- Caso contrário, retorne o texto sem alterações.

08

Trabalhando com Undefined

Declare uma variável que pode ser `undefined` ou `string`. Crie uma função que receba essa variável e retorne "Valor ausente" se for `undefined`, ou o próprio valor se for uma `string`.

09

Convertendo Tipos

Crie uma função chamada `converterParaNumero` que receba uma string representando um número (por exemplo, "42") e retorne o valor convertido para número. Caso o valor não seja convertível, retorne `NaN`.

THANK
YOU

@wallace027dev