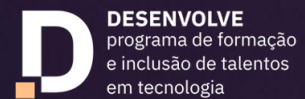


KORU



A group of diverse young people, including men and women of various ethnicities, are smiling and waving. The image is overlaid with a purple gradient. The text "TYPESCRIPT II" is written in a bold, white, sans-serif font across the center of the image.

TYPESCRIPT II

ONDE ESTAMOS?

- O que é **TypeScript** e por que usá-lo
- Instalação e compilação (**tsc**)
- **Tipos Primitivos** (string, number, boolean, etc.)
- **Tipos Especiais** (any, unknown, void, null, undefined)
- **Arrays e Objetos básicos** com tipagem
- **Funções** com tipagem



TYPES VS INTERFACES

TYPES VS INTERFACES

Interfaces (interface):

Usadas principalmente para definir a forma de objetos.

Podem ser **estendidas** (extends) e **mescladas**.

Mais comuns para **definir contratos de objetos**.

Types (type):

Mais versáteis. Podem **definir aliases para tipos primitivos, uniões (|), interseções (&), tuplas**, e também formas de objetos.

Não podem ser estendidos da mesma forma que interfaces (mas podem ser compostos).

Não podem ser mesclados.

INTERFACES NA PRÁTICA

Definimos a **estrutura esperada** para um objeto User.

O TypeScript verifica se os objetos atribuídos ao tipo User seguem essa estrutura.

```
interface User {  
  id: number;  
  name: string;  
  email: string;  
  isActive: boolean;  
}  
  
const user1: User = {  
  id: 1,  
  name: "Alice",  
  email: "alice@example.com",  
  isActive: true,  
};  
  
// Erro! Falta a propriedade 'isActive'  
// const user2: User = {  
//   id: 2,  
//   name: "Bob",  
//   email: "bob@example.com",  
// };
```

TIPOS NA PRÁTICA

Usamos **type** para **criar um alias** para um tipo primitivo (união de strings).

Também usamos type para **definir a forma de um objeto** Product.

```
type ProductStatus = "available" | "out_of_stock" | "discontinued";

type Product = {
  id: number;
  name: string;
  price: number;
  status: ProductStatus;
};

const product1: Product = {
  id: 101,
  name: "Laptop",
  price: 1200,
  status: "available", // OK
};

// Erro! 'pending' não é um status válido
// const product2: Product = {
//   id: 102,
//   name: "Mouse",
//   price: 25,
//   status: "pending",
// };
```

DIFERENÇAS CHAVE

Característica	Interface	Type Alias
Define Objetos?	✓ Sim	✓ Sim
Define Primitivos?	✗ Não	✓ Sim (como alias)
Define Uniões/Interseções?	✗ Não	✓ Sim
Extensível?	✓ Sim (com <code>extends</code>)	✓ Sim (com interseção <code>&</code>)
Mesclável?	✓ Sim (Declaration Merging)	✗ Não

A diverse group of young people, including men and women of various ethnicities, are smiling and waving. The image is overlaid with a semi-transparent purple gradient. The text "INTERFACES EXTENSÍVEIS" is written in large, bold, white capital letters across the center-right of the image.

INTERFACES EXTENSÍVEIS

INTERFACES EXTENSÍVEIS

Podemos estender interfaces existentes para adicionar novas propriedades ou métodos.

Dog herda todas as propriedades de **Animal** e adiciona **breed** e **bark**.

```
interface Animal {  
  name: string;  
  makeSound(): void;  
}  
  
interface Dog extends Animal {  
  breed: string;  
  bark(): void;  
}  
  
const myDog: Dog = {  
  name: "Rex",  
  breed: "Golden Retriever",  
  makeSound: () => console.log("Woof!"),  
  bark: () => console.log("Bark bark!"),  
};  
  
myDog.makeSound();  
myDog.bark();
```



COMPOSIÇÃO DE TIPOS

COMPOSIÇÃO DE TIPOS (COM TYPES)

Combinando Tipos com Uniões e Interseções

União (|): Um valor pode ser um ou outro tipo.

Interseção (&): Um valor deve ter as propriedades de ambos os tipos.

UIWidget combina as propriedades de **Draggable** e **Resizable**.

Id pode ser tanto um **number** quanto uma **string**.

```
type Draggable = {
  drag: () => void;
};

type Resizable = {
  resize: () => void;
};

// Interseção: Deve ser arrastável E redimensionável
type UIWidget = Draggable & Resizable;

const myWidget: UIWidget = {
  drag: () => console.log("Dragging..."),
  resize: () => console.log("Resizing..."),
};

myWidget.drag();
myWidget.resize();

// União: Pode ser string OU number
type Id = number | string;

const userId: Id = 123; // OK
const productId: Id = "abc-456"; // OK
```



ARRAYS COM TIPOS

MÉTODOS DE ARRAY TIPADOS

Os métodos built-in de arrays no JavaScript (map, filter, reduce, etc.) são tipados no TypeScript.

○ **TypeScript entende o tipo dos elementos dentro do array** (Person).

Ele **infere** (ou você pode explicitamente tipar) o tipo retornado pelos métodos.


```
interface Person {
  name: string;
  age: number;
}

const people: Person[] = [
  { name: "Alice", age: 30 },
  { name: "Bob", age: 25 },
  { name: "Charlie", age: 35 },
];

// filter retorna um array de Person (ou subtipo)
const adults = people.filter(person => person.age >= 30);
// adults é do tipo Person[]

// map transforma cada elemento e retorna um novo array (de strings neste caso)
const names = people.map(person => person.name.toUpperCase());
// names é do tipo string[]

// reduce pode retornar um tipo diferente (number neste caso)
const totalAge = people.reduce((sum, person) => sum + person.age, 0);
// totalAge é do tipo number
```

OBJETOS COMPLEXOS

DEFININDO ENTRADAS E SAÍDAS DE FUNÇÕES

Tipar as funções que recebem ou retornam objetos complexos aumenta a clareza e segurança do código.

Definimos os formatos dos objetos usando interfaces

```
interface ProductInput {
  name: string;
  price: number;
  description?: string; // Propriedade opcional
}

interface ProductInfo {
  id: number;
  name: string;
  price: number;
  createdAt: Date;
}

// Função que recebe um objeto tipado e retorna outro objeto tipado
function createProduct(product: ProductInput): ProductInfo {
  // Lógica para salvar no "banco de dados", etc.
  const newProduct = {
    id: Math.floor(Math.random() * 10000), // Exemplo simples
    name: product.name,
    price: product.price,
    createdAt: new Date(),
  };
  return newProduct;
}

const productToAdd: ProductInput = {
  name: "Caderno",
  price: 15.5,
};

const createdProduct = createProduct(productToAdd);
console.log(createdProduct.id, createdProduct.name);
// Erro! 'description' não existe no tipo ProductInfo
// console.log(createdProduct.description);
```



TIPOS E DOM

INTERAGINDO COM ELEMENTOS HTML

○ **TypeScript possui tipos embutidos** para representar elementos do DOM.

É importante verificar se o elemento foi encontrado (pois `querySelector` pode retornar `null`).

Usamos tipos específicos para ter acesso às propriedades corretas.

```
// Selecionando um input
const inputElement: HTMLInputElement | null = document.querySelector('#myInput');

if (inputElement) {
  // TypeScript sabe que inputElement é um HTMLInputElement
  // e tem a propriedade 'value'
  console.log(inputElement.value);
  inputElement.value = 'Hello TS DOM!';
  // Erro! 'innerText' não é comum em input, mas existe em HTMLElement
  // console.log(inputElement.innerText);
}

// Selecionando um botão
const buttonElement: HTMLButtonElement | null = document.getElementById(
  'myButton'
) as HTMLButtonElement; // Podemos usar 'as' para afirmar o tipo se tivermos certeza

if (buttonElement) {
  buttonElement.textContent = 'Click Me';
}

// Selecionando um div genérico
const divElement: HTMLDivElement | null = document.querySelector('.my-div');

if (divElement) {
  divElement.innerHTML = '<strong>Conteúdo HTML</strong>';
}
```




TIPOS E EVENTOS

TIPAGEM DE EVENTOS

Os **objetos de evento** que recebemos nos event listeners **também são tipados**.

Usamos **tipos de evento específicos** (MouseEvent, SubmitEvent, KeyboardEvent, etc.).

Muitas vezes precisamos usar **`as`** para afirmar o tipo do event.target, pois o TypeScript não consegue inferir o tipo exato do elemento que disparou o evento em tempo de compilação.

```
const myButton = document.getElementById('myButton');

if (myButton) {
  // Tipando o evento de clique
  myButton.addEventListener('click', (event: MouseEvent) => {
    console.log('Botão clicado!');
    // TypeScript sabe que event é um MouseEvent
    console.log('Coordenadas:', event.clientX, event.clientY);
    // Erro! 'key' não existe em MouseEvent
    // console.log('Tecla:', event.key);
  });
}

const myInput = document.querySelector('#myInput');

if (myInput) {
  // Tipando o evento de input (change é um tipo de Event)
  myInput.addEventListener('change', (event: Event) => {
    const target = event.target as HTMLInputElement; // Precisamos afirmar o tipo do target
    console.log('Valor alterado:', target.value);
  });
}

const myForm = document.querySelector('#myForm');

if (myForm) {
  // Tipando o evento de submit
  myForm.addEventListener('submit', (event: SubmitEvent) => {
    event.preventDefault(); // Previne o recarregamento da página
    console.log('Formulário submetido!');
    // Podemos acessar dados do formulário se necessário
  });
}
```




PRÁTICA TYPESCRIPT I

INSTRUÇÕES

Instruções

Crie um novo arquivo .ts para cada exercício (ou use um único arquivo e separe os exercícios com comentários).

Escreva o código JavaScript conforme as instruções.

Adicione a tipagem TypeScript apropriada.

Compile o arquivo (tsc nome-do-arquivo.ts). Se houver erros de tipagem, corrija-os.

Execute o código JavaScript compilado (node nome-do-arquivo.js) para ver o resultado no console.

INSTRUÇÕES

Exercício 01

Declare uma variável **userName** do tipo **string** e atribua seu nome a ela.

Declare uma variável **userAge** do tipo **number** e atribua sua idade a ela.

Declare uma variável **isStudent** do tipo **boolean** e atribua true ou false.

Declare uma variável **greeting** do tipo **string** e use interpolação para criar uma mensagem que inclua o nome e a idade do usuário.

Imprima todas as variáveis no console.

INSTRUÇÕES

Exercício 02

Declare um **array** chamado **numbers** que só pode **conter números**. Adicione alguns números a ele.

Declare um **array** chamado **fruits** que só pode conter strings. Adicione alguns nomes de frutas a ele.

Declare um array chamado **mixedArray** que pode conter números OU strings (usando o tipo união |). Adicione elementos de ambos os tipos.

Imprima cada array no console.

Acesse e imprima o segundo elemento de numbers e o último elemento de fruits.

INSTRUÇÕES

Exercício 03

Declare um objeto chamado **person** com as seguintes propriedades e seus tipos:

- **name**: string
- **age**: number
- **city**: string
- **isEmployed**: boolean

Crie um **objeto person** que siga essa estrutura e atribua valores.

Imprima o objeto person completo no console.

Acesse e imprima o nome e a idade da pessoa.

INSTRUÇÕES

Exercício 04

Escreva uma função chamada **add** que recebe dois argumentos do tipo number e retorna a soma deles (do tipo number).

Escreva uma função chamada **greet** que recebe um argumento name do tipo string e retorna uma string de saudação (ex: "Olá, [name]!").

Escreva uma função chamada **logMessage** que recebe um argumento message do tipo string e não retorna nada (void). Esta função deve apenas imprimir a mensagem no console.

Chame cada função com argumentos apropriados e imprima os resultados das funções add e greet.



PRÁTICA TYPESCRIPT II

INSTRUÇÕES

Instruções

Crie um novo projeto (pasta) com um arquivo `index.html` e um arquivo `index.ts`.

No `index.html`, inclua tags básicas (`<!DOCTYPE>`, `<html>`, `<body>`) e uma tag `<script>` que aponte para o arquivo JavaScript compilado (`index.js`).

Adicione alguns elementos HTML com IDs ou classes que você possa selecionar (ex: um botão, um input, um parágrafo, uma div vazia).

Escreva o código TypeScript no `index.ts`.

Compile o arquivo (`tsc index.ts`).

Abra o `index.html` no navegador e abra o console (F12) para ver os resultados e possíveis erros em tempo de execução.

INSTRUÇÕES

Exercício 01

Defina uma interface **Address** com propriedades **street**, **city**, **zipCode** (todas strings).

Defina uma interface **Employee** que estenda a **interface Person** (defina-a novamente com **name: string; age: number;**) e adicione propriedades **employeeId (number)** e **department (string, opcional)**.

Defina um type **ContactInfo** que seja uma união de **string | Address**.

Defina um type **HasId** com uma propriedade **id: number**.

Defina um type **UserWithId** que seja uma interseção de **Person** e **HasId**.

Crie **objetos** que sigam essas novas interfaces e types.

INSTRUÇÕES

Exercício 02

Crie um **array** chamado **products** que contenha objetos, cada um seguindo uma **interface Product** (com **id**, **name**, **price**, **category**).

Defina a interface Product.

Use o **método map** no array **products** para criar um novo array contendo apenas os nomes dos produtos (um array de strings).

Use o **método filter** no array **products** para criar um novo array contendo apenas os produtos de uma categoria específica.

Imprima os resultados de **map**, **filter** no console.

INSTRUÇÕES

Exercício 03

No seu **index.html**, adicione:

- Um botão com `id="myButton"`.
- Um input de texto com `id="myInput"`.
- Uma div vazia com `id="outputDiv"`.

No seu **index.ts**:

- Selecione o botão usando `document.getElementById` e tipá-lo como `HTMLButtonElement | null`.
- Selecione o input usando `document.querySelector` e tipá-lo como `HTMLInputElement | null`.
- Selecione a div usando `document.querySelector` e tipá-la como `HTMLDivElement | null`.
- Verifique se os elementos foram encontrados (`if (...)`).
- Se encontrados, imprima o `textContent` do botão, o `value` do input e o `innerHTML` da div no console.
- Tente acessar uma propriedade que não existe em um tipo específico (ex: `.value` em um `HTMLDivElement`) e veja o erro de tipagem.

INSTRUÇÕES

Exercício 04

Use os os elementos do Exercício 3 (myButton, myInput, outputDiv):

Adicione um **event listener de click** ao botão. Tipagem para o evento de clique (MouseEvent). Dentro do listener, mude o textContent da div para "Botão clicado!".

Adicione um **event listener de input** ou **change ao input**. Adicione tipagem para o evento. Dentro do listener, atualize o textContent da div com o texto atual do input (event.target as HTMLInputElement).

Adicione um **event listener de keydown** ao document. Adicione tipagem para o evento. Dentro do listener, imprima qual tecla foi pressionada (event.key) no console.

Compile e teste no navegador.

A group of diverse young people, including men and women of various ethnicities, are smiling and waving. The image is overlaid with a purple gradient. The word "RESUMO" is written in large, white, bold, sans-serif capital letters on the right side of the image.

RESUMO

RESUMO

- **Interfaces vs Types:**
Quando usar cada um e suas diferenças.
- **Extensão de Interfaces:**
Reutilizando definições com extends.
- **Composição de Tipos:**
Combinando tipos com | e &.
- **Arrays e Objetos Tipados:**
Manipulando estruturas de dados complexas com segurança.
- **Tipagem no DOM:**
Selecionando elementos (HTMLElement, HTMLInputElement, etc.) e **tipagem de eventos** (MouseEvent, Event, etc.).

KORU

