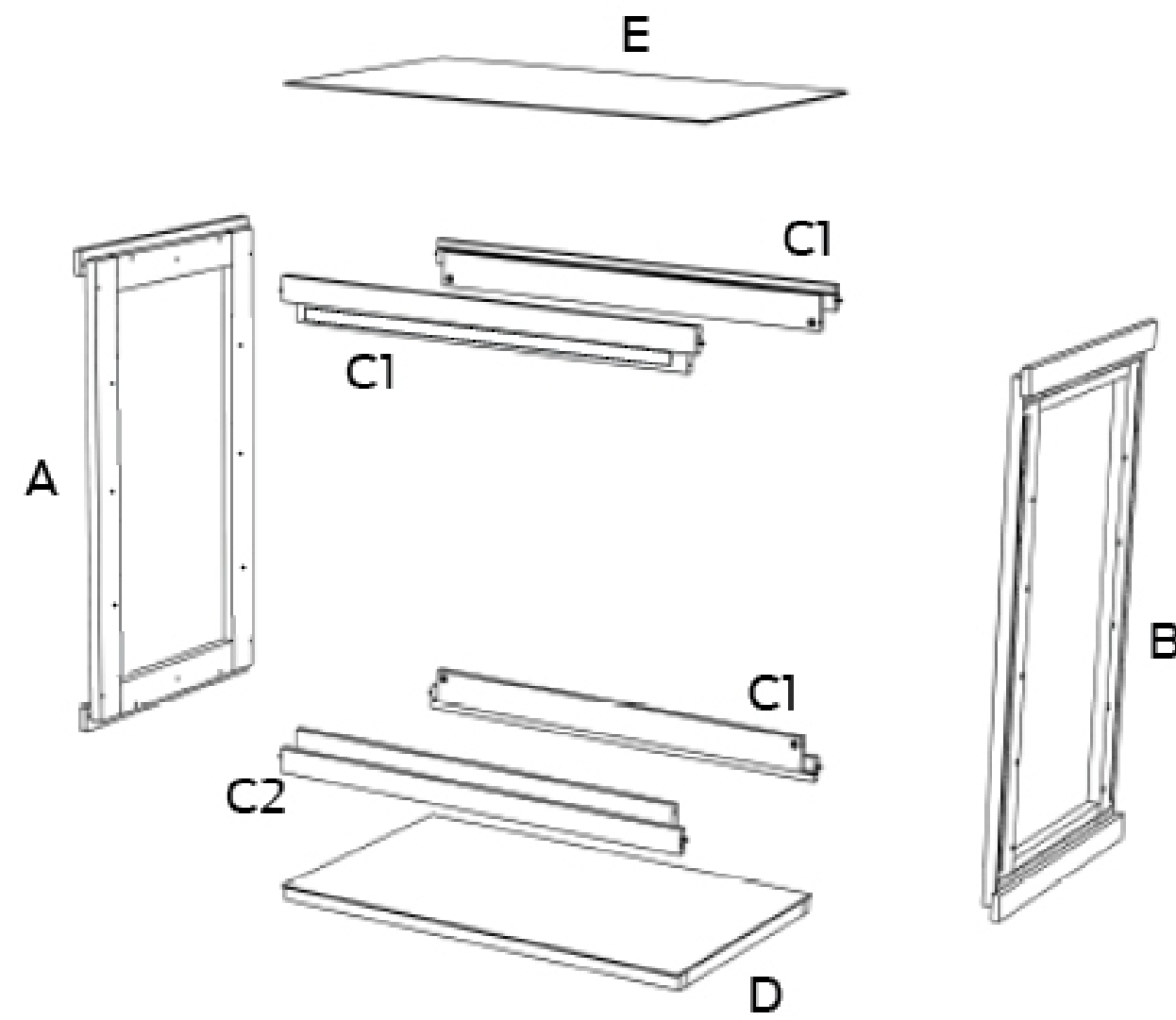


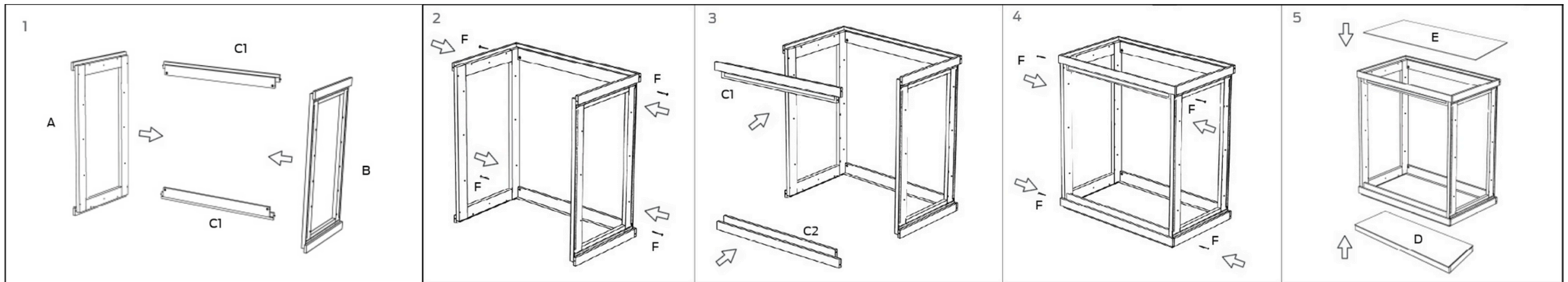
# INTERFACES E TIPOS AVANÇADOS

# O QUE SÃO INTERFACES?



# PORQUE USAR INTERFACES?

As **interfaces** em TypeScript são fundamentais para garantir que objetos sigam uma **estrutura consistente**. Elas ajudam a definir o formato de dados, especificando quais propriedades e métodos um objeto deve ter, além de seus tipos.



# BENEFÍCIOS PARA O DESENVOLVIMENTO

# BENEFÍCIOS PARA O DESENVOLVIMENTO

01

Organização e Clareza no Código

# BENEFÍCIOS PARA O DESENVOLVIMENTO

01

Organização e Clareza no Código

02

Detecção de Erros em Tempo de Compilação

# BENEFÍCIOS PARA O DESENVOLVIMENTO

01

**Organização e Clareza no Código**

02

**Detecção de Erros em Tempo de Compilação**

03

**Reutilização e Escalabilidade**

# BENEFÍCIOS PARA O DESENVOLVIMENTO

01

**Organização e Clareza no Código**

02

**Detecção de Erros em Tempo de Compilação**

03

**Reutilização e Escalabilidade**

04

**Facilitam a Integração com APIs e Bibliotecas**



# BENEFÍCIOS PARA O DESENVOLVIMENTO

01

**Organização e Clareza no Código**

02

**Detecção de Erros em Tempo de Compilação**

03

**Reutilização e Escalabilidade**

04


**Facilitam a Integração com APIs e Bibliotecas**

05

**Compatibilidade com Programação Orientada a Objetos**

COMO USAR  
INTERFACES

**Interfaces** em TypeScript permitem definir a estrutura de um objeto, garantindo que ele tenha determinadas propriedades e tipos.



```
1 interface Usuario {  
2     nome: string;  
3     idade: number;  
4 }  
5  
6 const usuario: Usuario = {  
7     nome: "João",  
8     idade: 25,  
9 };
```

**Pontos importantes:**

- As propriedades são obrigatórias por padrão.
- Podem incluir propriedades opcionais usando ?.

## Propriedades opcionais



```
1 interface Produto {  
2     nome: string;  
3     preco: number;  
4     descricao?: string; // opcional  
5 }  
6  
7 const produto: Produto = {  
8     nome: "Notebook",  
9     preco: 5000,  
10 };;
```

## Interfaces e funções:

Uma interface pode descrever a assinatura de uma função.



```
1 interface Soma {  
2     (a: number, b: number): number;  
3 }  
4  
5 const somar: Soma = (a, b) => a + b;  
6 console.log(somar(3, 5)); // Saída: 8
```

## Extensão de interfaces:


Interfaces podem ser estendidas para criar novos tipos baseados em outros existentes.

```
1 interface Animal {  
2     especie: string;  
3 }  
4  
5 interface Cachorro extends Animal {  
6     raca: string;  
7 }  
8  
9 const cachorro: Cachorro = {  
10     especie: "Canis lupus familiaris",  
11     raca: "Golden Retriever",  
12 };
```

TIPOS

AVANÇADOS

**Interfaces** em TypeScript permitem definir a estrutura de um objeto, garantindo que ele tenha determinadas propriedades e tipos.



```
1 interface Usuario {  
2     nome: string;  
3     idade: number;  
4 }  
5  
6 const usuario: Usuario = {  
7     nome: "João",  
8     idade: 25,  
9 };
```



## Intersection Types

Combinam vários tipos em um único tipo.

```
1 interface Pessoa {
2   nome: string;
3 }
4
5 interface Trabalhador {
6   profissao: string;
7 }
8
9 type TrabalhadorPessoa = Pessoa & Trabalhador;
10
11 const trabalhador: TrabalhadorPessoa = {
12   nome: "Carlos",
13   profissao: "Desenvolvedor",
14 };
```

## Type Aliases

Usados para criar apelidos para tipos.

```
1 type Coordenadas = {
2   x: number;
3   y: number;
4 };
5
6 const ponto: Coordenadas = {
7   x: 10,
8   y: 20,
9 };
```

## Tipos Literais

Permitem especificar valores fixos.

```
1 type Status = "ativo" | "inativo";
2
3 const usuarioStatus: Status = "ativo"; // Válido
4 // const usuarioStatus: Status = "pendente"; // Inválido
```

## Readonly e Partial

- Readonly torna as propriedades imutáveis.
- Partial torna todas as propriedades opcionais.

```
1 interface Configuracao {
2   tema: string;
3   layout: string;
4 }
5
6 const configuracao: Readonly<Configuracao> = {
7   tema: "escuro",
8   layout: "grid",
9 };
10
11 // configuracao.tema = "claro"; // Erro: não pode ser alterado
12 const novaConfiguracao: Partial<Configuracao> = {
13   tema: "claro",
14 };
```

# TYPE VS INTERFACE

Interfaces e type são ferramentas poderosas no TypeScript, mas possuem algumas diferenças que afetam como e quando utilizá-las:

## Interfaces

- **Foco:** Usadas para descrever a estrutura de objetos.
- **Extensibilidade:** Interfaces podem ser estendidas usando extends.

```
1 interface Pessoa {  
2   nome: string;  
3 }  
4  
5 class Aluno implements Pessoa {  
6   nome: string;  
7   constructor(nome: string) {  
8     this.nome = nome;  
9   }  
10 }
```

## Type

- **Foco:** Mais genérico, usado para criar tipos complexos (union, intersection, etc.) e apelidos de tipos.

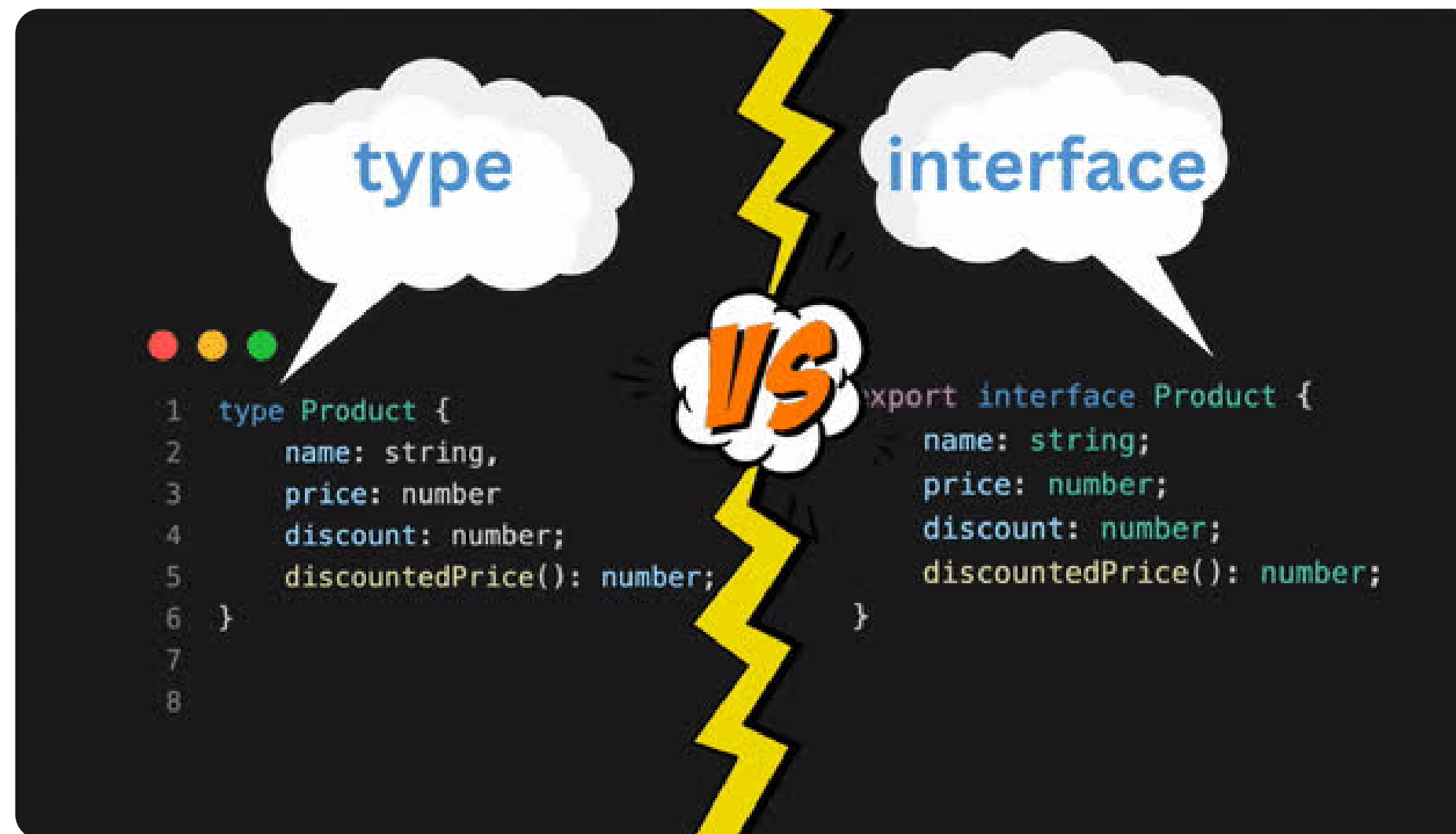
```
1 type Idade = number | string;  
2  
3 type Coordenadas = { x: number; y: number } & { z: number };
```

- **Imutável:** Uma vez declarado, um type não pode ser alterado ou estendido diretamente.
- **Usos adicionais:** Permite declarar tipos primitivos, funções e literais.

# QUANDO USAR?

- Use interfaces para descrever objetos e suas estruturas, especialmente em casos simples ou quando o código pode crescer (pela capacidade de extensão).
- Use type para casos mais complexos ou quando precisar de union/intersection types, apelidos ou literais.

Ambos são intercambiáveis em muitos cenários, então a escolha pode depender do estilo ou convenção do time.



# EXERCÍCIOS

01

**Crie uma interface chamada Pessoa com as propriedades:**

- nome (string),
- idade (number),
- ativo (boolean).

Em seguida, crie um objeto do tipo Pessoa e inicialize com valores.

02

**Array de Objetos**

Utilizando a interface Pessoa do exercício anterior, crie um array chamado listaDePessoas que contenha 3 objetos do tipo Pessoa.

03

**Type com Union**

Crie um type chamado Status que pode ser "ativo" ou "inativo".

Em seguida, adicione a propriedade status: Status à interface Pessoa e atualize os objetos criados anteriormente para incluir essa nova propriedade.

# EXERCÍCIOS

04

## Interface com Métodos

Crie uma interface chamada Produto com as propriedades:

- id (number),
- nome (string),
- preco (number).

Adicione também um método chamado calcularDesconto que receba uma porcentagem (number) e retorne o preço com desconto.

05

## Type para Arrays

Crie um type chamado Numeros que é um array de números. Em seguida, escreva uma função chamada calcularMedia que receba um parâmetro do tipo Numeros e retorne a média dos valores.

THANK  
YOU

@wallace027dev