


## 8. Generics em TypeScript

Generics são uma das características mais poderosas em linguagens de tipagem estática como TypeScript. Eles permitem que você crie componentes que são capazes de trabalhar com qualquer tipo de dado, mantendo a segurança dos tipos. 

### O que são Generics?

#### 1. Introdução aos Generics

- Generics permitem que você defina uma função, uma classe ou uma interface que pode trabalhar com qualquer tipo de dado que você especificar mais tarde.

```
function identidade<T>(arg: T): T {  
    return arg;  
}
```

#### 2. Usando Generics em Funções

- Você pode passar qualquer tipo de dado para funções que usam generics, o que as torna extremamente versáteis.

```
let saidaString = identidade<string>("minhaString");  
let saidaNumero = identidade<number>(100);
```

### Generics com Interfaces

#### • Interfaces que Usam Generics

- Generics podem ser usados em interfaces para definir propriedades ou métodos que serão determinados quando a interface for implementada.

```
interface ParGenerico<K, V> {  
    chave: K;  
    valor: V;  
}  
  
let item: ParGenerico<number, string> = { chave: 1, valor: "Teste" };
```

## Generics em Classes

- **Classes com Generics**
  - **Classes também podem ser definidas com generics, permitindo que você crie estruturas de dados dinâmicas que trabalham com qualquer tipo de dado.**

```
class Caixa<T> {  
    conteudo: T;  
    constructor(valor: T) {  
        this.conteudo = valor;  
    }  
}  
  
let caixaString = new Caixa<string>("Hello World");  
let caixaNumero = new Caixa<number>(123);
```

## Generics com Restrições

- **Restringindo Generics**
  - **Você pode restringir os tipos que um generic pode aceitar, usando a palavra-chave `extends`.**

```
function tamanho<T extends { length: number }>(arg: T): number {  
    return arg.length;  
}  
  
let tamanhoString = tamanho("Teste"); // Ok  
let tamanhoArray = tamanho([1, 2, 3, 4]); // Ok  
// let tamanhoNumero = tamanho(10); // Erro: 'number' não tem 'length'
```

## Exercícios para Praticar

### 1. Funções Generics:

- Crie uma função generic `primeiroElemento` que retorna o primeiro elemento de um array de qualquer tipo.

### 2. Generics com Interfaces:

- Defina uma interface `Pilha<T>` que suporte operações de 'push' e 'pop'. Implemente esta interface em uma classe.

### 3. Classes com Generics:

- Implemente uma classe `Mapa<K, V>` que simule a funcionalidade de um objeto Map, permitindo adicionar e buscar pares de chave-valor.

### 4. Generics com Restrições:

- Crie uma função que aceite somente arrays ou strings como argumento, fazendo uso de generics com restrições.

### 5. Generics Complexos:

- Escreva uma função `mergeObjects` que combine dois objetos e retorne um novo objeto que combine as propriedades de ambos, usando generics.