Aula de Typescript

Prof. Hete Caetano - 21/02/2024

FIAP

HETE CAETANO

Formação Acadêmica:

- MBA Cloud - Engineering & Architecture
- Marketing Digital
- Administração de Empresas

Áreas de atuação:

- Engenharia de Software com mais de 23 anos de experiência em desenvolvimento de aplicações web
- Cloud Solutions
- Inteligência Artifical
- Internet das Coisas (IoT)

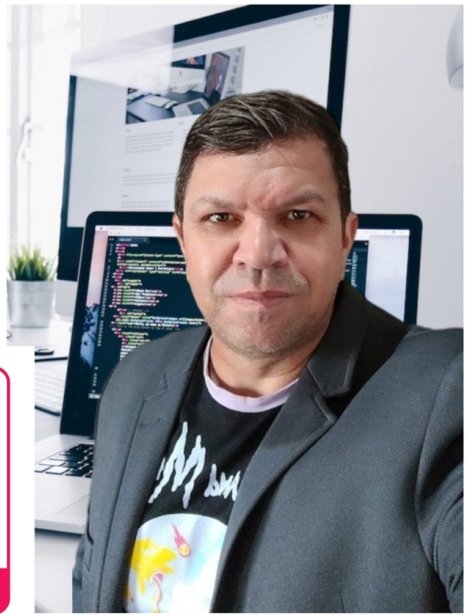
Experiência:

- Gestão de produtos digitais, desenvolvimento estratégicos com novas tecnologias, criação de plataformas digitais, inovação e novos negócios

Contato:

- profhete.santos@fiap.com.br
- hetecaetano@gmail.com
- https://www.linkedin.com/in/hete-caetano-0126471a

HETE CAETANO



Nesta aula, vamos introduzir o TypeScript, uma linguagem de programação que estende o JavaScript, adicionando tipos estáticos opcionais. Vamos explorar os conceitos básicos do TypeScript e como ele se relaciona com o JavaScript. Esta aula é destinada a desenvolvedores que já têm familiaridade com JavaScript.

Objetivos da Aula:

1. **Introduzir o TypeScript** e explicar sua importância no desenvolvimento de aplicativos modernos.
2. Explorar os **conceitos básicos do TypeScript**, incluindo tipos estáticos, interfaces e compilação.
3. Demonstrar **como iniciar um projeto TypeScript** e **converter código JavaScript existente para TypeScript**.

O que é TypeScript?

- Criptestesia é uma linguagem de programação desenvolvida pela Microsoft.
- É uma extensão do JavaScript que adiciona recursos de tipagem estática opcional.

Por que é usado?

- Adiciona tipagem estática opcional, permitindo aos desenvolvedores especificar tipos de dados para variáveis, parâmetros de função e valores de retorno.
- Ajuda a detectar erros de tipo em tempo de compilação.
- Melhora a legibilidade do código e facilita a manutenção.
- Oferece suporte para características modernas do JavaScript, como classes, módulos e arrow functions.

Tipagem estática opcional:

- Permite aos desenvolvedores especificarem tipos de dados, mas não é obrigatório.
- Ajuda a identificar erros de tipo antes da execução do código.
- Melhora a autodocumentação do código e sua compreensão.

> **Em resumo:** *TypeScript é uma linguagem de programação poderosa que estende o JavaScript, adicionando tipos estáticos opcionais e outras funcionalidades úteis. Ele é amplamente utilizado para o desenvolvimento de aplicativos em grande escala devido à sua capacidade de detectar erros em tempo de compilação e oferecer uma base sólida para construir aplicativos robustos e seguros.*

Tipos Básicos em TypeScript:

Os tipos básicos em TypeScript são fundamentais para definir o tipo de dados que uma variável pode armazenar. Nesta seção, vamos explorar os tipos básicos, como `number`, `string`, `boolean`, `null`, `undefined` e `any`, e demonstrar como declarar variáveis com tipos explícitos e implícitos.

1. Tipos Básicos:

- **number:** representa números, incluindo números inteiros e de ponto flutuante.
- **string:** representa sequências de caracteres.
- **boolean:** representa valores verdadeiros ou falsos.
- **null:** representa um valor nulo ou ausente.
- **undefined:** representa um valor não definido.
- **any:** representa um tipo dinâmico que pode ser qualquer coisa.

2. Declaração de Variáveis:

a. Tipos Explícitos:

```
let idade: number;
let nome: string;
let ativo: boolean;
let valor: any;

idade = 25;
nome = 'Maria';
ativo = true;
valor = 10;
```

Neste exemplo, as variáveis `idade`, `nome` e `ativo` têm tipos explícitos definidos (`number`, `string` e `boolean`, respectivamente). A variável `valor` é do tipo `any`, o que significa que pode receber qualquer tipo de valor.

b. Tipos Implícitos:

```
let quantidade = 10; // tipo number
let sobrenome = 'Silva'; // tipo string
let disponivel = true; // tipo boolean
```

```
let lista = [1, 2, 3]; // tipo number[]
```

Neste exemplo, as variáveis `quantidade`, `sobrenome` e `disponivel` têm tipos implícitos inferidos pelo TypeScript com base nos valores atribuídos a elas. A variável `lista` é inferida como um array de números (`number[]`).

3. Utilizando Tipos Básicos:

```
let idade: number = 30;
let nome: string = 'João';
let ativo: boolean = true;
let salario: number | null = null; // Union Types: número ou null
let endereco: string | undefined; // Union Types: string ou undefined

console.log(idade); // Saída: 30
console.log(nome); // Saída: João
console.log(ativo); // Saída: true
console.log(salario); // Saída: null
console.log(endereco); // Saída: undefined
```

4. Considerações:

- Ao usar tipos explícitos, você informa ao TypeScript qual é o tipo esperado para uma variável, aumentando a segurança e a clareza do código.
- No entanto, em muitos casos, o TypeScript é capaz de inferir automaticamente os tipos com base nos valores atribuídos às variáveis, o que torna a declaração de tipos explícitos opcional e pode tornar o código mais conciso e legível.

Interfaces em TypeScript:

As interfaces desempenham um papel fundamental no TypeScript, permitindo definir a estrutura dos objetos e garantir a consistência de tipos em todo o código. Nesta seção, vamos explorar o que são interfaces em TypeScript, como são usadas para definir a estrutura de objetos e demonstrar sua aplicação na prática.

1. O que são Interfaces:

- Em TypeScript, uma interface é um contrato que define a estrutura de um objeto, incluindo os nomes e tipos das propriedades.

- Elas são usadas para especificar um conjunto de propriedades e métodos que um objeto deve ter para ser considerado compatível com essa interface.
- As interfaces são úteis para criar tipos personalizados e garantir a consistência dos dados em todo o código.

2. Criando e Usando Interfaces:

a. Definição de uma Interface:

```
interface Usuario {
    nome: string;
    idade: number;
    email?: string; // Propriedade opcional
}
```

Neste exemplo, definimos uma interface chamada `Usuario`, que especifica que um objeto deve ter propriedades `nome` do tipo `string`, `idade` do tipo `number` e opcionalmente `email` do tipo `string`.

b. Utilizando Interfaces:

```
function cadastrarUsuario(usuario: Usuario) {
    // Implementação da função
```

```
}

let novoUsuario = { nome: 'Maria', idade: 30, email: 'maria@example.com' };
cadastrarUsuario(novoUsuario);
```

Neste exemplo, temos uma função `cadastrarUsuario` que recebe um parâmetro do tipo `Usuario`. Podemos passar um objeto com as propriedades especificadas pela interface `Usuario` para esta função.

3. Implementação de Métodos em Interfaces:

Além de propriedades, as interfaces também podem especificar métodos que um objeto deve implementar. Veja um exemplo:

```
interface Calculadora {
    somar(x: number, y: number): number;
    subtrair(x: number, y: number): number;
}

let minhaCalculadora: Calculadora = {
    somar(x, y) {
```

```
        return x + y;
    },
    subtrair(x, y) {
        return x - y;
    }
};

console.log(minhaCalculadora.somar(5, 3)); // Saída: 8
console.log(minhaCalculadora.subtrair(5, 3)); // Saída: 2
```

Neste exemplo, definimos uma interface `Calculadora` com métodos `somar` e `subtrair`. Em seguida, criamos um objeto `minhaCalculadora` que implementa esses métodos.

4. Considerações:

- As interfaces são uma ferramenta poderosa em TypeScript para definir contratos e garantir a consistência dos tipos em todo o código.
- Elas facilitam a manutenção do código, pois ajudam a detectar erros de tipo em tempo de compilação e fornecem uma documentação clara das estruturas de dados esperadas.

> **Em resumo:** *Ao utilizar interfaces de forma eficaz, você pode escrever código mais seguro, legível e escalável em TypeScript, facilitando a manutenção e o desenvolvimento de aplicativos robustos.*

Tipos de Função em TypeScript:

Os tipos de função são uma característica poderosa do TypeScript que permite especificar os tipos de parâmetros e valores de retorno de funções. Nesta seção, discutiremos como realizar a tipagem de parâmetros e valores de retorno de funções em TypeScript e forneceremos exemplos de declaração de tipos de função e uso de anotações de tipos.

1. Tipagem de Parâmetros e Valores de Retorno:

- Em TypeScript, é possível definir os tipos dos parâmetros e valores de retorno de funções, garantindo maior segurança e clareza no código.
- A tipagem de parâmetros permite especificar os tipos de dados que podem ser passados para uma função, enquanto a tipagem de valores de retorno indica o tipo de dado que a função deve retornar.

2. Exemplo de Declaração de Tipos de Função:

a. Declaração de Tipos de Parâmetros e Valor de Retorno:

```
function soma(a: number, b: number): number {
    return a + b;
}
```

Neste exemplo, declaramos uma função chamada `soma` que recebe dois parâmetros (`a` e `b`) do tipo `number` e retorna um valor do tipo `number`.

b. Tipos de Função com Parâmetros Opcionais e Valor de Retorno Opcional:

```
function saudacao(nome: string, hora?: string): string {
    if (hora) {
        return `Boa ${hora}, ${nome}!`;
    } else {
        return `Olá, ${nome}!`;
    }
}
```

Neste exemplo, a função `saudacao` possui um parâmetro opcional `hora`, indicado pelo operador `?`. Isso significa que podemos chamar a função com ou sem o segundo argumento. O valor de retorno da função é do tipo `string`, mas também pode ser opcional, caso a saudação seja feita em uma hora específica.

3. Uso de Anotações de Tipos:

```
function calcularMedia(notas: number[]): number {
    let total = 0;
    for (let nota of notas) {
        total += nota;
    }
    return total / notas.length;
}

let notasAluno: number[] = [8, 7, 9, 10];
let media: number = calcularMedia(notasAluno);
console.log(`A média do aluno é: ${media}`);
```

Neste exemplo, temos uma função `calcularMedia` que recebe um array de números (`notas`) e retorna a média desses números. As anotações de tipos são usadas para especificar que `notas` é um array de números e que o valor de retorno da função é um número.

4. Considerações:

- A tipagem de parâmetros e valores de retorno de funções em TypeScript ajuda a prevenir erros de tipo durante o desenvolvimento, fornecendo uma camada adicional de segurança e clareza no código.
- Utilize anotações de tipos para declarar explicitamente os tipos de parâmetros e valores de retorno de funções, sempre que possível, para garantir uma codificação mais robusta e manutenível em TypeScript.

Compilação TypeScript:

O TypeScript é uma linguagem de programação que, como mencionado anteriormente, estende o JavaScript adicionando tipos estáticos opcionais. Quando você escreve código em TypeScript, ele precisa ser compilado para JavaScript antes de ser executado no navegador ou em qualquer ambiente que suporte JavaScript. Nesta seção, vamos explorar como o TypeScript é compilado para JavaScript e demonstrar o processo de configuração de um projeto TypeScript e compilação de código.

1. Compilação de TypeScript para JavaScript:

- Quando você escreve código em TypeScript, ele é salvo com a extensão `.ts`.
- O TypeScript fornece um compilador de linha de comando chamado `tsc` que pode ser usado para converter arquivos TypeScript em arquivos JavaScript.
- Durante a compilação, o compilador TypeScript realiza a verificação de tipos estáticos e converte o código TypeScript em código JavaScript equivalente, removendo todas as anotações de tipos.

2. Configuração de um Projeto TypeScript:

a. Instalação do TypeScript:

Antes de começar, você precisa ter o TypeScript instalado globalmente ou localmente em seu projeto. Você pode instalá-lo usando o npm (Node Package Manager) com o seguinte comando:

```
npm install -g typescript
```

Isso instalará o TypeScript globalmente em sua máquina.

b. Inicialização do Projeto TypeScript:

Para iniciar um novo projeto TypeScript, você pode usar o comando `tsc --init` no terminal. Isso criará um arquivo de configuração chamado `tsconfig.json` no diretório do seu projeto.

c. Configuração do Arquivo tsconfig.json:

O arquivo `tsconfig.json` contém as opções de configuração para o compilador TypeScript. Aqui está um exemplo básico de configuração:

```
{
  "compilerOptions": {
    "target": "ES5",
    "module": "commonjs",
    "outDir": "dist"
  },
  "include": ["src/**/*"]
}
```

- `target`: Especifica para qual versão do JavaScript o código deve ser compilado. Neste exemplo, estamos compilando para ES5.
- `module`: Especifica o sistema de módulos a ser usado. Aqui, estamos usando o sistema de módulos `commonjs`.

- `outDir`: Especifica o diretório de saída onde os arquivos JavaScript compilados serão armazenados.
- `include`: Especifica os arquivos e diretórios a serem incluídos na compilação. Aqui, estamos incluindo todos os arquivos no diretório `src`.

3. Compilação de Código TypeScript:

Para compilar seu código TypeScript, basta executar o comando `tsc` no terminal, seguido pelo nome do arquivo TypeScript que deseja compilar. Por exemplo:

```
tsc arquivo.ts
```

Isso compilará o arquivo `arquivo.ts` para JavaScript e produzirá um arquivo JavaScript correspondente com o mesmo nome, mas com a extensão `.js`.

Se você configurou um arquivo `tsconfig.json`, pode simplesmente executar o comando `tsc` no terminal sem nenhum argumento. Isso instruirá o compilador TypeScript a compilar todos os arquivos TypeScript no projeto de acordo com as configurações especificadas no arquivo `tsconfig.json`.

4. Considerações:

- A compilação de TypeScript para JavaScript é um processo essencial ao desenvolver aplicativos em TypeScript.

Conversão de Código JavaScript para TypeScript:

Converter código JavaScript existente para TypeScript é uma tarefa importante ao migrar um projeto para TypeScript ou ao iniciar um novo projeto que se beneficiaria da tipagem estática oferecida pelo TypeScript. Nesta seção, vamos orientar sobre como realizar essa conversão e discutir as etapas envolvidas, bem como considerações importantes a ter em mente.

1. Etapas para Converter Código Sincripta para TypeScript:

a. Renomear os Arquivos:

- Os arquivos JavaScript devem ser renomeados com a extensão `.ts` para indicar que agora são arquivos TypeScript.

b. Adicionar Anotações de Tipos:

- Adicione anotações de tipos aos parâmetros de função, valores de retorno e variáveis onde necessário.
- Isso pode ser feito explicitamente, usando tipos primitivos como `number`, `string`, `boolean`, etc., ou de forma mais genérica, usando `any` quando o tipo exato não é conhecido.

c. Identificar e Corrigir Erros:

- Durante o processo de conversão, é possível que surjam erros relacionados a tipos incompatíveis ou inconsistências no código.
- Identifique esses erros e corrija-os conforme necessário, adicionando as anotações de tipo apropriadas ou fazendo ajustes no código.

d. Aproveitar Recursos do TypeScript:

- Ao converter para TypeScript, você pode aproveitar os recursos adicionais oferecidos pela linguagem, como interfaces, enums, tipos de união, tipos literais e muito mais.
- Considere refatorar o código para tirar proveito desses recursos e melhorar a legibilidade e manutenibilidade do código.

2. Considerações Importantes:

a. Migrar Gradualmente:

- Em projetos grandes, pode ser impraticável converter todo o código JavaScript para TypeScript de uma vez só.
- Considere migrar gradualmente, começando com partes específicas do código ou módulos que se beneficiariam mais da tipagem estática.

b. Testes de Regressão:

- Após a conversão para TypeScript, é importante realizar testes de regressão para garantir que as alterações não tenham introduzido novos bugs ou quebras de funcionalidade.

c. Ajustes de Configuração:

- Faça os ajustes necessários na configuração do compilador TypeScript, como a especificação do arquivo `tsconfig.json`, para refletir as necessidades do projeto.

d. Aprendizado Contínuo:

- A conversão de JavaScript para TypeScript pode ser uma oportunidade para aprender mais sobre os recursos e práticas recomendadas do TypeScript.
- Aproveite essa oportunidade para explorar tutoriais, documentação oficial e recursos online para aprofundar seus conhecimentos em TypeScript.

3. Exemplo de Conversão Simples:

JavaScript Original:

```
function saudacao(nome) {
    return 'Olá, ' + nome + '!';
}

console.log(saudacao('Mundo'));
```

JavaScript Convertido para TypeScript:

```
function saudacao(nome: string): string {
    return 'Olá, ' + nome + '!';
}

console.log(saudacao('Mundo'));
```

> **Conclusão:** *Converter código JavaScript para TypeScript pode trazer muitos benefícios, incluindo a detecção de erros em tempo de compilação e a melhoria da legibilidade e manutenibilidade do código. Siga as etapas mencionadas acima e considere as considerações importantes ao realizar essa conversão. Com prática e experiência, você se tornará mais confortável e eficiente em converter código JavaScript para TypeScript.*

EXERCÍCIOS

Exercício 1: Declaração de Variáveis

Escreva um programa em TypeScript que realize as seguintes tarefas:

1. Declare uma variável chamada `idade` e atribua a ela o valor 25. Garanta que essa variável seja do tipo number.
2. Declare uma variável chamada `nome` e atribua a ela o valor 'João'. Garanta que essa variável seja do tipo string.
3. Declare uma variável chamada `isAtivo` e atribua a ela o valor true. Garanta que essa variável seja do tipo boolean.

Seu programa deve declarar e inicializar essas variáveis conforme descrito acima. Certifique-se de que as variáveis tenham os tipos de dados corretos. Ao final do programa, não se esqueça de imprimir no console o valor de cada variável declarada.

```
// Exercício 1: Declaração de Variáveis

// Declare uma variável chamada idade e atribua a ela o valor 25. Garanta que essa variável seja do tipo number.
let idade: number = 25;

// Declare uma variável chamada nome e atribua a ela o valor 'João'. Garanta que essa variável seja do tipo string.
let nome: string = 'João';

// Declare uma variável chamada isAtivo e atribua a ela o valor true. Garanta que essa variável seja do tipo boolean.
let isAtivo: boolean = true;

// Imprimir os valores das variáveis no console
console.log("Idade:", idade);
console.log("Nome:", nome);
console.log("IsAtivo:", isAtivo);
```

Este programa declara e inicializa as variáveis `idade`, `nome` e `isAtivo` com os valores fornecidos e garante que cada variável tenha o tipo de dado correto. Em seguida, imprime os valores de cada variável no console para verificação.

Exercício 2: Interfaces

Escreva um programa em TypeScript que realize as seguintes tarefas:

1. Crie uma interface chamada `Usuario` com as seguintes propriedades: `nome` (string), `idade` (number) e `email` (string).
2. Declare uma variável do tipo `Usuario` e atribua valores a todas as suas propriedades.

Seu programa deve criar a interface `Usuario` conforme descrito acima e, em seguida, declarar e inicializar uma variável do tipo `Usuario` com valores para todas as suas propriedades. Ao final do programa, não se esqueça de imprimir no console o valor da variável do tipo `Usuario`.

```
1  // Exercício 2: Interfaces
2
```

```
// Crie uma interface chamada Usuario com as seguintes propriedades: nome (string), idade (number) e email (string).
interface Usuario {
    nome: string;
    idade: number;
    email: string;
}

// Declare uma variável do tipo Usuario e atribua valores a todas as suas propriedades.
let usuario: Usuario = {
    nome: 'Maria',
    idade: 30,
    email: 'maria@example.com'
};

// Imprimir o valor da variável do tipo Usuario no console
console.log("Usuário:", usuario);
```

Este programa define a interface `Usuario` com as propriedades especificadas e, em seguida, declara e inicializa uma variável `usuario` do tipo `Usuario` com valores para todas as suas propriedades. Em seguida, imprime o valor da variável `usuario` no console para verificação.

Exercício 3: Funções

Escreva um programa em TypeScript que realize as seguintes tarefas:

1. Crie uma função chamada `soma` que recebe dois parâmetros do tipo number e retorna a soma desses dois números.
2. Crie uma função chamada `saudacao` que recebe um parâmetro do tipo string chamado `nome` e retorna uma mensagem de saudação utilizando esse nome.

Seu programa deve definir as duas funções conforme descrito acima. Após a definição das funções, chame cada uma delas com valores adequados como argumentos e imprima os resultados no console.

```
// Exercício 3: Funções

// Crie uma função chamada soma que recebe dois parâmetros do tipo number e retorna a soma desses dois números.
function soma(a: number, b: number): number {
    return a + b;
}

```

```
// Crie uma função chamada saudacao que recebe um parâmetro do tipo string chamado nome e retorna uma mensagem de saudação utilizando esse nome.
function saudacao(nome: string): string {
    return `Olá, ${nome}! Bem-vindo(a)!`;
}

// Chamar a função soma e imprimir o resultado no console
console.log("Soma:", soma(5, 3));

// Chamar a função saudacao e imprimir o resultado no console
console.log(saudacao("Ana"));
```

Este programa define duas funções, `soma` e `saudacao`, conforme descrito nos requisitos do exercício. Em seguida, chama cada uma dessas funções com valores apropriados e imprime os resultados no console para verificação.

Exercício 4: Compilação TypeScript

Escreva um programa em TypeScript que realize as seguintes tarefas:

1. Crie um arquivo TypeScript chamado `app.ts`.

2. Escreva o código TypeScript dos exercícios 1 a 3 no arquivo `app.ts`.
3. Compile o arquivo TypeScript para JavaScript usando o compilador TypeScript (tsc) no terminal.
4. Execute o arquivo JavaScript resultante usando o Node.js para verificar se o código está funcionando corretamente.

```
// Exercício 4: Compilação TypeScript

// 1. Crie um arquivo TypeScript chamado app.ts.

// 2. Escreva o código TypeScript nos exercícios 1 a 3 no arquivo app.ts.

// Exercício 1: Declaração de Variáveis

// Declare uma variável chamada idade e atribua a ela o valor 25. Garanta que essa variável seja do tipo number.
let idade: number = 25;

// Declare uma variável chamada nome e atribua a ela o valor 'João'. Garanta que essa variável seja do tipo string.
let nome: string = 'João';

```

```typescript
// Declare uma variável chamada isAtivo e atribua a ela o valor true. Garanta que essa variável seja do tipo boolean.
let isAtivo: boolean = true;

// Exercício 2: Interfaces

// Crie uma interface chamada Usuario com as seguintes propriedades: nome (string), idade (number) e email (string).
interface Usuario {
    nome: string;
    idade: number;
    email: string;
}

// Declare uma variável do tipo Usuario e atribua valores a todas as suas propriedades.
let usuario: Usuario = {
    nome: 'Maria',
    idade: 30,
    email: 'maria@example.com'
};

// Exercício 3: Funções
```

```
35
36 // Crie uma função chamada soma que recebe dois parâmetros do tipo number e retorn
   a a soma desses dois números.
37 function soma(a: number, b: number): number {
38     return a + b;
39 }
40
41 // Crie uma função chamada saudacao que recebe um parâmetro do tipo string chamado
   nome e retorna uma mensagem de saudação utilizando esse nome.
42 function saudacao(nome: string): string {
43     return `Olá, ${nome}! Bem-vindo(a)!`;
44 }
45
46 // 3. Compile o arquivo TypeScript para JavaScript usando o compilador TypeScript
   (tsc) no terminal.
47
48 // 4. Execute o arquivo JavaScript resultante usando o Node.js para verificar se o
   código está funcionando corretamente.
```

Este programa cria um arquivo TypeScript chamado `app.ts` e escreve o código dos exercícios 1 a 3 dentro dele. Em seguida, você pode usar o compilador TypeScript para compilar o arquivo `app.ts` em

um arquivo JavaScript e executá-lo usando o Node.js para verificar se o código está funcionando corretamente.

Exercício 5: Conversão de Código JavaScript para TypeScript

Para este exercício, vamos converter um pequeno projeto JavaScript existente para TypeScript. Suponha que temos o seguinte código JavaScript:

```
// JavaScript Original

function calcularArea(raio) {
    return Math.PI * raio * raio;
}

function imprimirArea(raio) {
    console.log('A área do círculo é: ' + calcularArea(raio));
}

```

```
imprimirArea(5);
```

Agora, vamos converter este código JavaScript para TypeScript, adicionando tipos estáticos aos parâmetros e valores de retorno das funções e adicionando interfaces conforme necessário para garantir a consistência de tipos em todo o projeto:

```
// TypeScript Convertido

// Interface para representar um círculo
interface Circulo {
    raio: number;
}

// Função para calcular a área de um círculo
function calcularArea(circulo: Circulo): number {
    return Math.PI * circulo.raio * circulo.raio;
}

// Função para imprimir a área de um círculo
function imprimirArea(circulo: Circulo): void {
    console.log('A área do círculo é: ' + calcularArea(circulo));
}

```

```
// Criar um objeto do tipo Circulo
let circulo: Circulo = {
    raio: 5
};

// Imprimir a área do círculo
imprimirArea(circulo);
```

Neste exemplo, criamos uma interface `Circulo` para representar um círculo, que contém apenas uma propriedade `raio` do tipo number. Em seguida, modificamos as funções `calcularArea` e `imprimirArea` para aceitar um objeto do tipo `Circulo` como argumento, em vez de um simples número. Finalmente, criamos um objeto `circulo` e o passamos para a função `imprimirArea` para imprimir a área do círculo. Este código TypeScript agora tem tipos estáticos e utiliza interfaces para garantir a consistência de tipos em todo o projeto.