Lista de Exercícios Práticos I

TypeScript

Esta lista de exercícios cobre tópicos do básico ao intermediário. Os exemplos de resposta estão incluídos para que você possa testar, ajustar e melhorar o código conforme necessário.

Exercício 1: Tipagem Básica

Instruções: Defina variáveis com os seguintes tipos: string, number, boolean, array e object. Atribua valores a essas variáveis e exiba-os no console.

```
let nome: string = "Alice";
let idade: number = 25;
let ativo: boolean = true;
let hobbies: string[] = ["Leitura", "Esportes", "Música"];
let endereco: { cidade: string; estado: string } = { cidade: "São Paulo", estado: "SP" };

console.log(`Nome: ${nome}, Idade: ${idade}, Ativo: ${ativo}`);
console.log(`Hobbies: ${hobbies.join(", ")}`);
console.log(`Endereço: ${endereco.cidade} - ${endereco.estado}`);
```

Dica de Melhoria: Adicione mais propriedades ao objeto endereco e expanda a lista de hobbies.

Exercício 2: Função com Tipagem e Parâmetro Opcional

Instruções: Crie uma função chamada saudacao que receba um nome e um parâmetro opcional idade. A função deve retornar uma saudação diferente se a idade for informada ou não.

```
function saudacao(nome: string, idade?: number): string {
   return idade ? `Olá, ${nome}! Você tem ${idade} anos.` : `Olá,
${nome}!`;
}

console.log(saudacao("Alice"));
console.log(saudacao("Bob", 30));
```

Dica de Melhoria: Tente adicionar mais parâmetros opcionais e retorne mensagens personalizadas com base nas combinações.

Exercício 3: Interface para Tipagem de Objetos

Instruções: Crie uma interface Produto com as propriedades nome, preco, e disponivel. Crie uma função que receba um array de produtos e retorne apenas os produtos disponíveis.

```
interface Produto {
   nome: string;
   preco: number;
   disponivel: boolean;
}

function filtrarDisponiveis(produtos: Produto[]): Produto[] {
   return produtos.filter(produto => produto.disponivel);
}

const produtos: Produto[] = [
   { nome: "Teclado", preco: 100, disponivel: true },
   { nome: "Mouse", preco: 50, disponivel: false },
   { nome: "Monitor", preco: 300, disponivel: true },
];

console.log(filtrarDisponiveis(produtos));
```

Dica de Melhoria: Adicione uma nova propriedade opcional na interface, como categoria, e filtre produtos por categorias.

Exercício 4: Manipulação de Arrays com TypeScript

Instruções: Crie uma função chamada calcularMedia que receba um array de números e retorne a média dos valores.

```
function calcularMedia(numeros: number[]): number {
  const total = numeros.reduce((acc, numero) => acc + numero, 0);
  return total / numeros.length;
}

console.log(calcularMedia([10, 20, 30, 40])); // 25
```

Dica de Melhoria: Altere o array para incluir valores decimais e negativos e verifique se a média é calculada corretamente.

Exercício 5: Classes e Herança

Instruções: Crie uma classe Animal com as propriedades nome e som e uma função emitirSom(). Depois, crie uma classe Cachorro que herde de Animal e tenha uma propriedade extra chamada raça.

```
class Animal {
  nome: string;
  som: string;

  constructor(nome: string, som: string) {
    this.nome = nome;
    this.som = som;
  }

  emitirSom(): string {
    return `${this.nome} faz ${this.som}`;
  }
}
```

```
class Cachorro extends Animal {
   raca: string;

   constructor(nome: string, som: string, raca: string) {
      super(nome, som);
      this.raca = raca;
   }
}

const rex = new Cachorro("Rex", "au au", "Labrador");
console.log(rex.emitirSom());
console.log(`Raça: ${rex.raca}`);
```

Dica de Melhoria: Adicione novos métodos à classe Cachorro, como latirForte, para alterar o som com base na raça.

Exercício 6: Função Assíncrona com Promises

Instruções: Crie uma função assíncrona buscarDados que simule uma busca de dados e retorne uma Promise com uma string após 2 segundos.

```
function buscarDados(): Promise<string> {
   return new Promise(resolve => {
      setTimeout(() => resolve("Dados carregados"), 2000);
   });
}

async function exibirDados() {
   const dados = await buscarDados();
   console.log(dados);
}

exibirDados();
```

Dica de Melhoria: Modifique o setTimeout para variar o tempo de resposta e adicione mais chamadas para observar o comportamento.

Exercício 7: Tipos Personalizados (Union Types)

Instruções: Crie uma função formatarEntrada que aceite um valor que pode ser string ou number. Se o valor for string, a função deve retornar a string em letras maiúsculas. Se for number, deve retornar o valor multiplicado por 10.

```
function formatarEntrada(valor: string | number): string | number {
   return typeof valor === "string" ? valor.toUpperCase() : valor * 10;
}

console.log(formatarEntrada("typescript")); // "TYPESCRIPT"
console.log(formatarEntrada(5)); // 50
```

Dica de Melhoria: Adicione tipos adicionais ao Union para aceitar boolean e manipule o valor com base no tipo.

Exercício 8: Enums e Funções

Instruções: Crie um enum para os dias da semana (DiasSemana) e uma função ehFimDeSemana que retorne true se o dia for sábado ou domingo.

```
typescript
Copiar código
enum DiasSemana {
   Segunda = "Segunda-feira",
   Terca = "Terça-feira",
   Quarta = "Quarta-feira",
   Quinta = "Quinta-feira",
   Sexta = "Sexta-feira",
   Sabado = "Sábado",
   Domingo = "Domingo",
}

function ehFimDeSemana(dia: DiasSemana): boolean {
   return dia === DiasSemana.Sabado || dia === DiasSemana.Domingo;
}

console.log(ehFimDeSemana(DiasSemana.Domingo)); // true
console.log(ehFimDeSemana(DiasSemana.Quinta)); // false
```

Dica de Melhoria: Adicione feriados ao enum e ajuste a função para verificar se é um dia especial ou feriado.

Exercício 9: Trabalhando com Generics

Instruções: Crie uma função genérica chamada reverterArray que receba um array de qualquer tipo e retorne o array invertido.

```
function reverterArray<T>(items: T[]): T[] {
   return items.reverse();
}

console.log(reverterArray([1, 2, 3])); // [3, 2, 1]
console.log(reverterArray(["a", "b", "c"])); // ["c", "b", "a"]
```

Dica de Melhoria: Teste com arrays de tipos mistos e experimente aplicar métodos de filtro ou mapeamento antes de inverter o array.

Exercício 10: Manipulação de Dados com Map e Filter

Instruções: Crie uma função filtrarPrecosAltos que receba um array de preços e retorne apenas os valores maiores que 100.

```
function filtrarPrecosAltos(precos: number[]): number[] {
   return precos.filter(preco => preco > 100);
}

console.log(filtrarPrecosAltos([50, 150, 200, 30])); // [150, 200]
```

Dica de Melhoria: Alterne o valor de corte e aplique map para modificar os preços selecionados antes de retorná-los.

Esses exemplos com respostas fornecem uma base sólida para você testar e entender a aplicação prática de cada conceito.

Aproveite para fazer ajustes e melhorias para explorar o TypeScript ainda mais!