Rotinas e funções:

Rotina:

Uma rotina, muitas vezes chamada de procedimento, é uma sequência de instruções ou um bloco de código que realiza uma tarefa específica. Pode ou não receber parâmetros e pode ou não retornar um valor.

Características:

Não necessariamente retorna valor: Uma rotina pode realizar ações ou modificar o estado do programa, mas não é obrigada a retornar um valor específico.

Pode ou não receber parâmetros: Pode aceitar valores de entrada (parâmetros) ou não, dependendo da necessidade.

Exemplo em TypeScript:

```
function imprimirMensagem(): void {
    console.log("Olá, mundo!");
}

// Chamada da rotina
imprimirMensagem();
```

Funções:

Definição:

Uma função é uma sub-rotina que recebe um ou mais parâmetros, realiza um conjunto de operações e retorna um valor. Funções são blocos de código reutilizáveis que ajudam na modularização e na promoção do reuso de código.

Características:

Retorna um valor: A principal característica é que uma função deve retornar um valor específico.

Recebe parâmetros: Uma função aceita valores de entrada (parâmetros) que podem ser usados em suas operações internas.

Exemplo em TypeScript:

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

// Chamada da função
const resultado: number = somar(3, 4);
console.log(resultado);  // Saída: 7
```

Diferenças:

Retorno de Valor:

A diferença fundamental entre rotinas e funções é que as funções obrigatoriamente retornam um valor, enquanto as rotinas podem ou não retornar algo.

Objetivo:

As rotinas são geralmente usadas quando se deseja realizar ações ou tarefas sem a necessidade de um resultado específico, enquanto as funções são projetadas para calcular valores e retorná-los.

Reutilização de Código:

As funções são especialmente projetadas para promover a reutilização de código, pois podem ser chamadas de vários lugares e proporcionam um meio eficaz de modularização.

Parâmetros:

Tanto rotinas quanto funções podem receber parâmetros, mas funções são mais comumente associadas à manipulação de valores de entrada.

O exemplo em TypeScript ilustra como esses conceitos podem ser aplicados nessa linguagem de programação específica.

Variáveis Locais e Globais

Variáveis Locais e Globais em TypeScript:

Em TypeScript, variáveis podem ser locais ou globais, e a diferença fundamental entre elas é o escopo em que são definidas.

Variáveis Locais:

Definição:

Variáveis locais são declaradas dentro de uma função ou bloco de código específico.

Elas só existem e podem ser acessadas dentro desse escopo particular.

Variáveis locais têm uma vida útil limitada ao tempo de execução da função em que são definidas.

Exemplo em TypeScript:

```
function exemploVariavelLocal(): void {
    // Variável local
    let mensagem: string = "Olá, mundo!";
    console.log(mensagem);
}

// Tentar acessar a variável fora da função resultará em um erro
// console.log(mensagem);  // Isso gerará um erro
```

Variáveis Globais:

Definição:

Variáveis globais são declaradas fora de qualquer função ou bloco de código, geralmente no nível superior do programa.

Elas podem ser acessadas por qualquer parte do código, incluindo funções.

Variáveis globais têm uma vida útil que dura durante toda a execução do programa.

Exemplo em TypeScript:

```
// Variável global
let contador: number = 0;

function incrementarContador(): void {
    // A variável global pode ser acessada e modificada dentro da função
    contador++;
    console.log("Contador atual:", contador);
}

// Chamadas da função
incrementarContador();
incrementarContador();
```

Observações Importantes:

Modificação de Variáveis Globais:

Modificar variáveis globais dentro de funções pode ser útil, mas também pode tornar o código menos previsível. Deve-se ter cuidado ao manipular variáveis globais para evitar efeitos colaterais indesejados.

Encapsulamento:

Em geral, é uma boa prática limitar o escopo das variáveis o máximo possível (usar variáveis locais sempre que apropriado) para evitar conflitos e facilitar a manutenção do código.

Parâmetros como Variáveis Locais:

Parâmetros de uma função são, por padrão, tratados como variáveis locais, existindo apenas dentro do escopo da função.

```
function exemploParametroLocal(parametro: string): void {
    console.log(parametro);
}

// Chamada da função
exemploParametroLocal("Isso é um parâmetro."); // Funciona
// console.log(parametro);  // Isso gerará um erro, pois 'parametro' é uma variável local
```

Exercícios:

1. Fazer o teste de mesa com o código abaixo:

```
function add(x){
    x++;
    return x;
}
let a = 0;
let b = 5
for(let i = 0; i<=3;i++){
    b = a-i;
    a = add(b);
    console.log(a);
}
```

2. Fazer o teste de mesa com o código abaixo.

```
function adicionarNumero(numeroLocal) {

  numeroLocal++
  return numeroLocal

}

function removerNumero(numeroLocal) {
  numeroLocal--;
  return numeroLocal

}

let numeroA = 10
let numreroB = 2

while (numeroA > 5) {

  numeroA = numeroA - 4
  if (numreroB > 0) {
    numreroB = removerNumero(numreroB)


  }
  numeroA = adicionarNumero(numeroA)

console.log(numeroA)


}
// console.log(numeroA)
```

3. Construir um programa no VsCode no qual o usuário irá digitar dois números e o programa irá
calcular a soma destes números (o estudante deverá utilizar função para que o programa escreva a saída do programa).

4. Construir um programa no qual o usuário deverá digitar dois números e o programa escreverá, na tela, qual dos dois números é o maior (utilizar o comando função).

5. Refazer o exercício 4 (acima), só que o usuário do programa deverá digitar três números e o
programa terá que dizer qual é o maior e qual é o menor.

6. Construir um programa no qual o usuário irá digitar um número e o programa irá dizer se ele
é par ou ímpar (utilizar o comando função).

7. Construir um programa no qual o usuário digitará um número entre 0 a 10. Caso o número
seja menor que 0 ou maior que 10, o programa escreverá uma mensagem dizendo que o número é inválido e pedirá, novamente, que o usuário digite outro número até ser um número válido.

Vetores

Geralmente a palavra vetor é chamada de arrays são estruturas de dados que permitem armazenar e acessar múltiplos valores em uma única variável. Em TypeScript, você pode criar arrays para armazenar elementos de qualquer tipo.

Exercícios:

Como criar vetores:

```
// Array de números
let numeros: number[] = [1, 2, 3, 4, 5];

// Array de strings
let frutas: string[] = ["maçã", "banana", "laranja"];

// Array de qualquer tipo (usando 'any')
let qualquerCoisa: any[] = [1, "dois", true, { nome: "João" }];
```

Como imprimir vetores:

```
for(let iteracao = 0; iteracao < numeros.length ; iteracao++){
    console.log(`Vetor ${numeros[iteracao]}`)
}
```

Como inserir valores diretamente em um vetor.

```
let meuVetor: number[] = [1, 2, 3];
// Atribuindo valores diretamente a índices específicos
meuVetor[3] = 4;
meuVetor[4] = 5;
console.log(meuVetor);  // Saída: [1, 2, 3, 4, 5]
```

como inserir valores utilizando o push

```
let meuVetor: number[] = [1, 2, 3];
// Atribuindo valores diretamente a índices específicos
meuVetor.push(4);
meuVetor.push(5);
console.log(meuVetor);  // Saída: [1, 2, 3, 4, 5]
meuVetor.forEach(element => {
console.log(element)
});
```

Usando o unshift inserir valor no inicio do vetor(lista)

```
let meuVetor: number[] = [1, 2, 3];
  meuVetor.unshift(0);
meuVetor.forEach(element => {
        console.log(`O valor modificado do vetor é ${element}`)
    });
```

b. Como escrever os valores de um vetor no console.log.

```
let meuVetor: number[] = [1, 2, 3];
// Atribuindo valores diretamente a índices específicos
meuVetor.push(4);
meuVetor.push(5);
console.log(meuVetor);  // Saída: [1, 2, 3, 4, 5]
```

c. Como remover valores de um vetor.

Remover o último elemento

```
let meuVetor: number[] = [1, 2, 3, 4, 5];
// Usando pop para remover o último elemento
meuVetor.pop();
console.log(meuVetor);  // Saída: [1, 2, 3, 4]
```

Remover o primeiro elemento

```
let meuVetor: number[] = [1, 2, 3, 4, 5];
// Usando shift para remover o primeiro elemento
meuVetor.shift();
console.log(meuVetor);
```

remover elementos por índice:

```
let meuVetor: number[] = [1, 2, 3, 4, 5];
// Usando splice para remover elementos a partir do índice 2 (inclusive)
meuVetor.splice(2, 2);
console.log(meuVetor);  // Saída: [1, 2, 5]
```