Seção 18 TypeScript – Tipos básicos

* Sintaxe de type annotations no typescript

// Tipos básicos (aqui ocorre inferência de tipos)

let nome: string = 'Luiz'   // Qualquer tipo de strings: '' "" ``

let idade: number = 78      // 10, 1.57, -5.55, 0xf00d, 0b1010, 0o7744

let adulto: boolean = true  // true ou false

let simbolo: symbol = Symbol('qualquer-symbol') // Symbol

let big: bigint = 10n       // bigint

// Arrays

let arrayDeNumeros: Array<number> = [1, 2, 3]

let arrayDeNumeros2: number[] = [1, 2, 3]

let arrayDeString: Array<string> = ['a', 'b', 'c']

let arrayDeString2: string[] = ['a', 'b', 'c']

// Objetos

let pessoa: {nome:string, idade:number, adulto?:boolean} = {

    idade: 30,

    nome: 'Luiz'

}

// Funções

function soma(x:number, y:number):number {

    return x + y

}

const soma2: (x:number, y:number) => number = (x, y) => x + y

* Tipo void

Quando uma função ou método não retorna nada.

function semRetorno(...args:string[]){

    console.log(args.join(' + '));

}

//// Luiz + Otavio

Não retorna nada apenas está exibindo o que foi pedido

Seja explicito nos retornos de funções e métodos

* Tipo objeto (objeto em geral)

const objetoA = {

    chaveA: 'Valor A',

    chaveB: 'Valor B'

}

objetoA.chaveA = 'Outro valor'

objetoA.chaveC = 'Nova Chave' //ERRO

uma vex que o tipo foi inferido você não consegue mais alterar o tipo dele

const objetoA: {

    chaveA: string;

    chaveB: string;

    chaveC?: string; // opcional

    [key: string]: unknown; // aceita qualquer valor

} = {

    chaveA: 'Valor A',

    chaveB: 'Valor B'

}

objetoA.qualquer = 'coisa'

objetoA.outra = 1234

console.log(objetoA);

{

  chaveA: 'Valor A',

  chaveB: 'Valor B',

  qualquer: 'coisa',

  outra: 1234

}

readonly chaveA: string;

para não alterar mais o valor dessa chave

const objetoA: {

    readonly chaveA: string;

} = {

    chaveA: 'Valor A',

    chaveB: 'Valor B'

}

objetoA.chaveA = 'Outro valor' // ERRO NÃO da para alterar essa

* Tipo Array

// Array <T> - T[]

export function multiplicaArgs(...args:Array<number>):number{

    return args.reduce((ac, valor) => ac \* valor, 1)

}

export function concatenaString(...args: string[]):string{

    return args.reduce((ac, valor) => ac + valor)

}

export function toUpperCase(...args:string[]):string[]{

    return args.map((valor) => valor.toUpperCase())

}

const resut = multiplicaArgs(1, 2, 3)

const concatenacao = concatenaString('a', 'b', 'c')

const upper = toUpperCase('r', 'o', 'd')

console.log(resut);

console.log(concatenacao);

console.log(upper);

6

abc

[ 'R', 'O', 'D' ]

* Tipo tuple

Tipo exclusivo do typescript, ele não existe no javascript

// Tuple

const dadosCliente1: readonly [number, string] = [1, 'Luiz']

const dadosCliente2: [number, string, string] = [2, 'Mario', 'Solto']

const dadosCliente3: [number, string, string?] = [4, 'Luiz']

const dadosCliente4: [number, string, ...string[]] = [8, 'Carlos', 'Volto', 'JN', 'PodCast']

// dadosCliente1[0] = 100

// dadosCliente1[1] = "300"

console.log(dadosCliente1);

console.log(dadosCliente2);

console.log(dadosCliente3);

console.log(dadosCliente4);

// readonly array

const array1: readonly string[] = ['Luiz', 'Otavio']

const array2: ReadonlyArray<string> = ['Maria', 'Julia']

console.log(array1);

console.log(array2);

* Tipos null e undefined