Téc em Desenvolvimento
de Sistemas Bilíngue

Desenvolver Código

Orientado a Objetos

UC4 | Prof. Vitor Hugo Lopes

Types

- Types são responsáveis por **definir o tipo de valor** a ser utilizado por cada variável ou função.

- Já vimos como tipar valores primitivos: **strings, arrays e booleans** e, também, tipos de mais alto nível como **arrays e objetos**

- Vamos relembrar os tipos que já aprendemos até agora

Types

String

```
const name: string = "Jules"
```

Boolean

```
const online: boolean = true
```

Number

```
const age: number = 21
```

Arrays Tipados

É necessário explicitar o tipo dos itens que vão estar num array. Existem duas formas de fazer isso, conforme exemplo ao lado.

```
let listaDeCompras: string[]

let listaDeCompras: Array<string>
```

Objetos Tipados

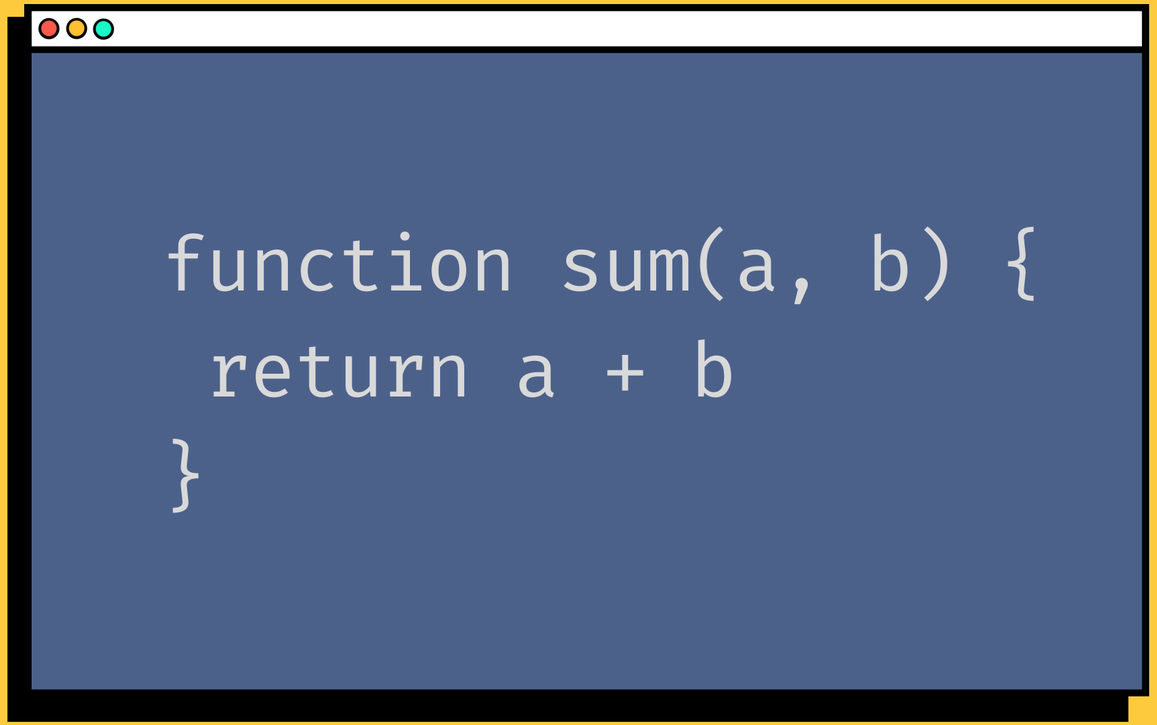
É necessário explicitar o tipo dos dados que vão estar como propriedades do objeto.

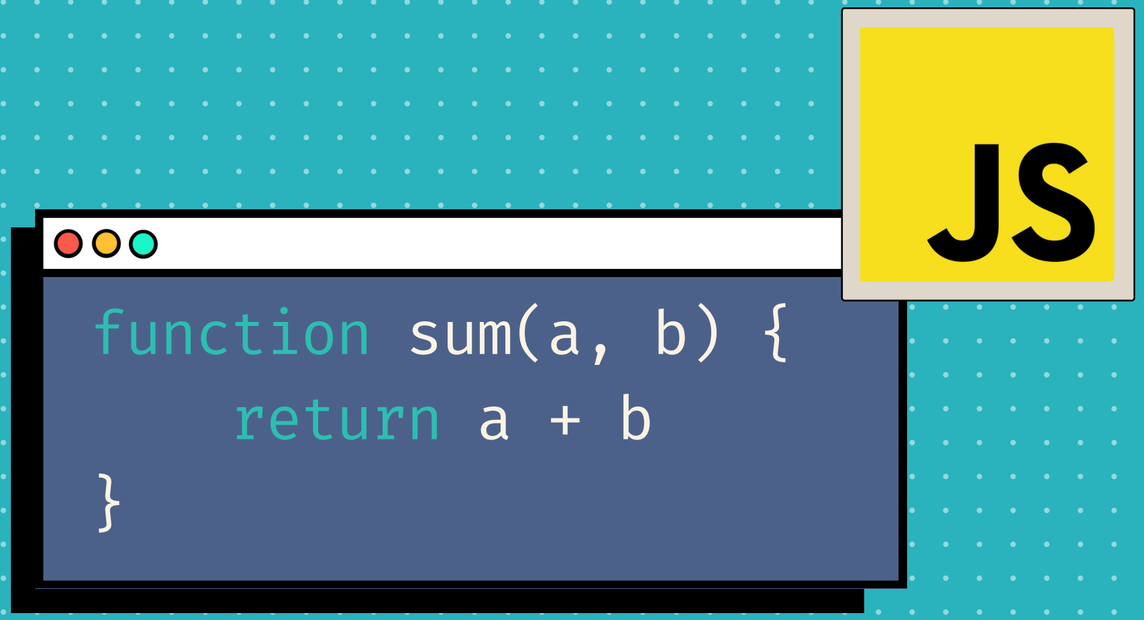
```
let aluno: {nome: string, idade: number}
```

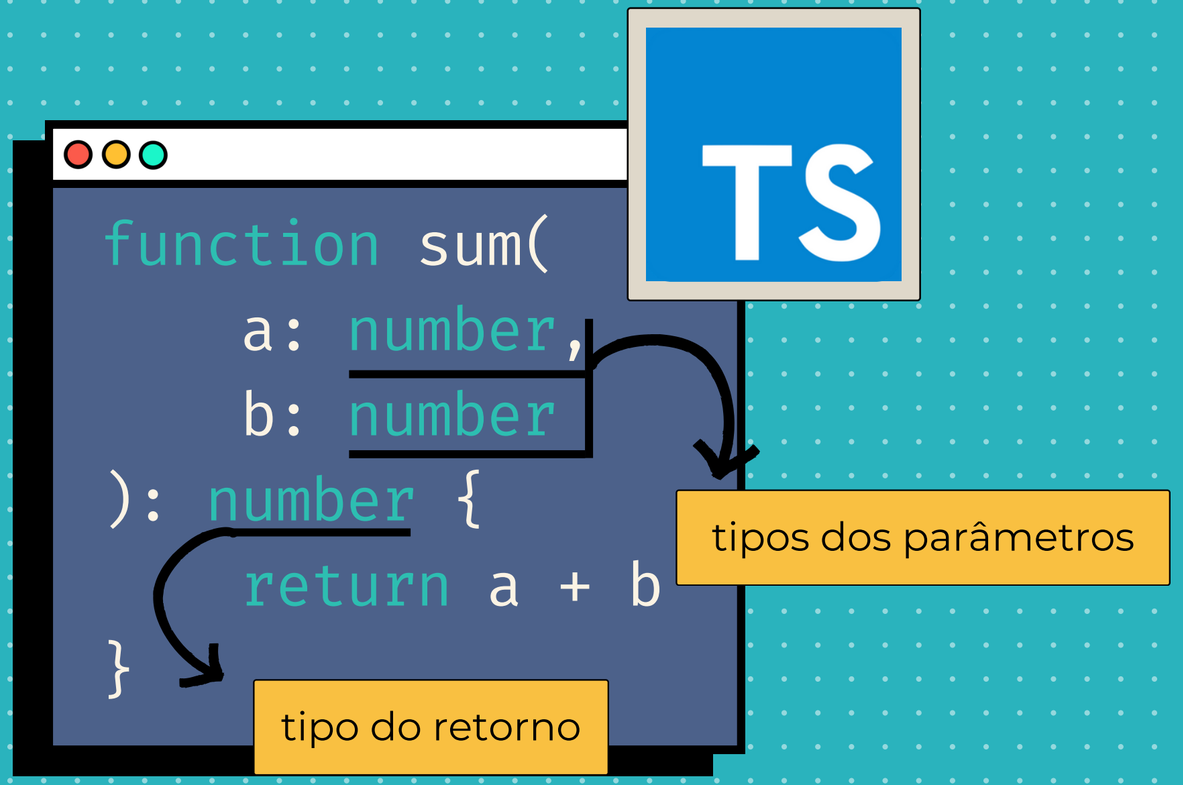
Funções Tipadas

No **Javascript**, podemos criar uma função que recebe dois números e chamá-la passando duas strings, por exemplo. Isso abre margem para bugs!

No **Typescript**, podemos tipar sua entrada (parâmetros) e saída (retorno)




JS
function sum(a, b) {
return a + b
}


TS
function sum(
a: number,
b: number
): number {
return a + b
}
tipos dos parâmetros
tipo do retorno

Parâmetros Opcionais

Nem sempre todos os dados que a função recebe são obrigatórios. Eventualmente precisaremos utilizar parâmetros opcionais, representados pelo ponto de interrogação.

```
function sayHello(name?: string): void {
 console.log("Hello,", name || "World")
}
```

parâmetro opcional

tipo informado quando não há retorno

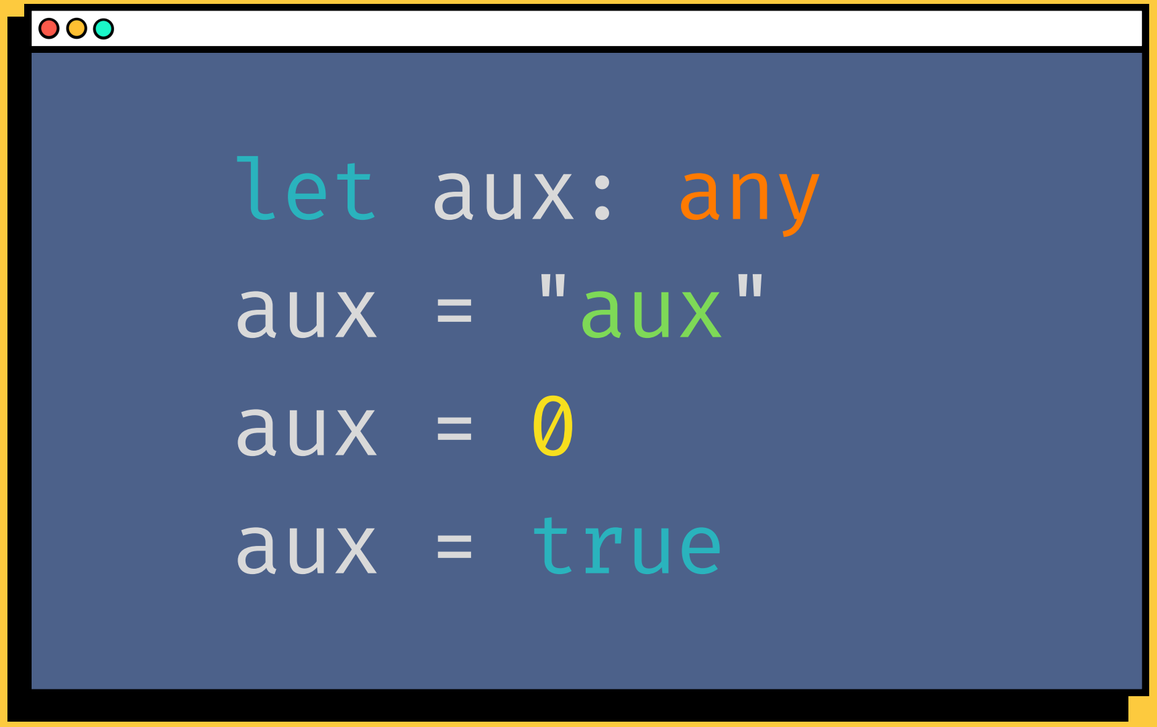
```
function sayHello(name?: string): void {
 console.log("Hello,", name || "World")
}
```

```
function applyDiscount(
    price: number,
    discount: number = 0.05
): number {
    return price * (1 - discount)
}
```

quando não enviado o valor de "discount" a função usará "0.05"

Tipo Any

Há um tipo especial, que deve ser evitado, mas as vezes é a única opção: **any**. Indica que a variável pode assumir qualquer valor



tsconfig.json

Em um projeto com vários arquivos, o processo de transpilação que vimos até agora é muito pouco prático.

Por isso, vamos criar um arquivo de configuração chamado **tsconfig.json** dentro da pasta do nosso projeto, usando o comando `npx tsc --init`

tsconfig.json

Configurando esse arquivo só precisamos usar o comando **npx tsc** no script, e, depois, rodar os javascripts que forem criados.

```
"start": "npx tsc && node ./build/index.js"
```

tsconfig.json

```
{
    "compilerOptions": {
        "target": "es6",                  /* Specify ECMAScript target version */
        "module": "commonjs",             /* Specify module code generation */
        "sourceMap": true,                /* Generates corresponding '.map' file. */
        "outDir": "./build",              /* Redirect output structure to the directory. */
        "rootDir": "./src",               /* Specify the root directory of input files. */
        "removeComments": true,           /* Do not emit comments to output. */
        "noImplicitAny": true,            /* Raise error on declarations with an implied 'any' type. */
    }
}
```

#DicaDoProf

Se você tem mais de um exercício na mesma pasta pode criar diferentes starts dentro do package.json. Por exemplo, considere que temos uma lista com 3 questões:

```
"start1": "npx tsc && node ./build/index1.js",
"start2": "npx tsc && node ./build/index2.js",
"start3": "npx tsc && node ./build/index3.js"
```

Obrigado!

Dúvidas? Escreva para vhlopes@senacrs.com.br