Aprofundamento em Typescript

Labenu_

O que vamos ver hoje?

- Revisão dos types que vimos até agora

- + Types:
  - Union Types
  - Type Aliases
  - Intersection Types
  - Type Inference
  - Enum

Revisão - Types

Labenu_

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 básicos

- **String**
  ```
  const name: string = "Jules"
  ```
- **Boolean**
  ```
  const online: boolean = true
  ```
- **Number**
  ```
  const age: number = 21
  ```

Types básicos

- **Arrays**

```
const arr: Array<number> = [1, 2, 3]
const array: number[] = [1, 2, 3]
```

- **Objetos**

```
const person: { name: string, age: number} = {
     name: "Astrodev",
     age: 30
}
```

- **Any**

```
let aux: any
aux = "Oi"
aux = 12
```

+ Types

Labenu_

Union Type

Labenu_

Union Type

Para as variáveis poderem assumir valores de **undefined** ou **null**, podemos colocar diretamente no tipo ou usar o chamado **union type**, inserindo uma **|** entre os tipos:

```
let text: string | undefined

let message: string | null = null
```

*Lembram do operador lógico **||** em Javascript, que representa OU? No Typescript utilizamos apenas **|**

Type Aliases

Labenu_

Type Aliases

- Para evitar repetições, podemos declarar uma **variável de tipo** (Type Aliases). Aliases significa **pseudônimo** (nome falso) .

- O **type** é apenas um "esqueleto" que definirá as propriedades que aquele tipo deve ter.

```
type person = {
      name: string,
      age: number
}
```

```
const astrodev: person = {
    name: "Astrodev",
    age: 30
}
```

Type Aliases + Union Type

Labenu_

Type Aliases + Union Type

```
type BirthDate = number | string | undefined

        function user (
                    name: string,
                    birthDate: BirthDate
                    ){}

        function userProfile (
                    name: string,
                    age: number,
                    birthDate: BirthDate
                    ){}
```

Type Aliases + Union Type

```
type BirthDate = number | string | undefined
```

```
function user (
          name: string
          birthDate: BirthDate
          ){}
```

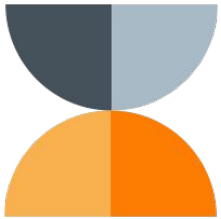
```
function userProfile (
          name: string,
          age: number,
          birthDate: BirthDate
          ){}
```

Vamos ver na prática! 🔬

Exercício 1

Crie um sistema de cadastro de usuários que contenha:

- Tipo Aliases para uma pessoa (Person) com as propriedades id, name, email, password e role;
- Tipos Aliases de conta: AdminAccount e NormalAccount; com as propriedades account e permission.



Pausa para relaxar 😴

10 min

- Utilizamos types para **definir o tipo de valor** de variáveis ou funções
- **Type Aliases** são **esqueletos** que um conjunto de dados pode assumir
- **Union Type** permite que um tipo de valor seja **um ou outro**. ex: string | undefined

Intersection Types

Labenu_

Intersection Types

- Intersection Type cria um novo tipo **combinando múltiplos types já existentes**.

- Assim como o Union Type está para **OU**, o Intersection Type está para **E**.

```
type UserInfo = User & Account
```

Type User

```
type User = {
        name: string,
        age: number
}
```

Type account

```
type Account = {
        userName: string,
        password: string
}
```

Type Intersection (união dos types User e Account)

```
type UserInfo = User & Account
```

UserInfo

```
const user: UserInfo = {
        name: "Lua",
        age: 27,
        userName:"lualua",
        password: "123abc"
}
```

O objeto `user` deve herdar todas propriedades dos types User e Account juntos

Vamos ver na prática! 🔬

Type Inference

Labenu_

Type Inference

Ou **inferência de tipo**: Para evitar tipagens redundantes, podemos declarar uma variável diretamente com o tipo de valor que irá assumir. O typescript é capaz de subentender o tipo.

```
const name: string = "Labenu"
```

```
const name = "Labenu"
```

Enum 🔑

Labenu_

Enum 🔑

Em Typescript, temos uma estrutura de dados que permite a declaração de tipos de variáveis quando **elas podem assumir valores restritos pré-definidos** (dados que não mudam):

```
enum LabenuClasses {
     LOVELACE = "Lovelace",
     MARYAM = "Maryam",
     CARVER = "Carver",
     JOY = "Joy",
     GUIMARAES = "Guimarães",
     VAUGHAN = "Vaughan"
}
```

```
const labenuTeacher = {
     name: "Janaylla",
     class: LabenuClasses.MARYAM
}
```

Enum 🔑

Em Typescript, temos uma estrutura de dados que permite a declaração de tipos de variáveis quando **elas podem assumir valores restritos pré-definidos** (dados que não mudam):

```
enum LabenuClasses {
    LOVELACE = "Lovelace",
    MARYAM = "Maryam",
    CARVER = "Carver",
    JOY = "Joy",
    GUIMARAES = "Guimarães",
    VAUGHAN = "Vaughan"
}
```

```
const labenuTeacher = {
    name: "Janaylla",
    class: LabenuClasses.MARYAM
}
```

acessamos a propriedade do enum com ponto

Vamos ver na prática! 🔬

Exercício 2

Vamos continuar nosso sistema de cadastro de usuários criando:

1. **Enum** com valores **ADMIN** e **NORMAL**;
2. Tipo **Intersection** unindo: pessoa(Person) + permissão (Role);
3. Um **array de usuários** que permite **apenas** guardar usuários do tipo Person + Role;
4. Crie duas pessoas, uma com permissão normal e a outra admin;
5. Guarde essas pessoas no array de usuários.

Labenu_

Resumo 📓

Labenu_

Resumo 📓

- Types são responsáveis por definir o **tipo de valor** a ser utilizado por cada variável ou função
- Union Type está para **OU** assim como Intersection está para **E**
- **Type Inference** permite declararmos uma variável diretamente com o tipo de valor que irá assumir
- **Enum** são estruturas que permitem que **declaremos os valores** que uma variável pode assumir, se estes forem **fixos e limitados**
- A representação de **tipo** de objetos segue uma sintaxe parecida com a atribuição de valor

Dúvidas? 🧐

Labenu_

Obrigado!