Voltando a falar de tipos

enums

É um tipo onde podemos definir um conjunto de valores constantes pré-definidos.

```
enum Cor {
  Amarelo,
  Vermelho,
  Verde
}

let qualCor: Cor;
qualCor = Cor.Vermelho;
```

Por padrão, a cor vermelho tem o valor 1, enquanto que amarelo tem o valor 0 e verde tem o valor 2.

Mas, os valores das constantes dentro de um enum também pode ser customizados

Voltando a falar de tipos

enums

```
enum Cor {
  Amarelo = 12,
  Vermelho = 30,
  Verde = 5
}

let qualCor: Cor;
qualCor = Cor.Vermelho;
```

Assim, a cor vermelho tem o valor 30, enquanto que amarelo tem o valor 12 e verde tem o valor 5.

Voltando a falar de tipos

enums

Também é possível exibir o nome da cor…

```
enum Cor {
  Amarelo = 12,
  Vermelho = 30,
  Verde = 5
}

let qualCor: string;
qualCor = Cor[30];
```

Voltando a falar de tipos

Any

Esse tipo já existe no javascript, e como o nome já diz, ele aceita qualquer valor.

```
let ativo: any = true;
ativo = ‘Monstro’;
ativo = [100, true, 'Como isso funciona’;

console.log(ativo[2]);
```

Interpolação

Podemos definir strings utilizando backsticks (`). Este operador permite o uso de interpolação (embutir expressões no meio da string sem fechar sua delimitação). Essas expressões são chamadas de template string.

```
let nome: string = 'Jorge Santos';

console.log(`Olá, você chegou `);


console.log(`Olá, você chegou ${nome}`);
```

Usando funções

parâmetro com tipo

retorno com tipo

```
let temMaisTitulos = function(titulos : number) : boolean{
  return titulos > 35;
}
```

```
let numero = 8;
console.log(`Ter ${numero} é suficiente para passar o … ? ${temMaisTitulos(8) ? 'SIM': 'NÃO'}`);
```

expressão ternária

template string

O que acontece caso seja passado uma string?

Usando funções

Arrow function

os parâmetros ficam a esquerda da seta

a implementação fica a direita da seta

```
let escreva = (valor: string) => console.log(`O nome é ${valor}`);

escreva('João ninguém');
```

Funções

Escreva uma função que faça uso de parâmetros com valor padrão

Classes e Interfaces

Define atributos e comportamentos

Classes e Interfaces

Sintaxe

```
class Laptop {
  tela: number;

  constructor (tela: number) {
    this.tela = tela;
  }

  ligarMonitor() {
    console.log('O monitor do laptop foi ligado!');
  }
}
```

```
let computador = new Laptop(14);

computador.ligarMonitor();
```

Classes e Interfaces

Herança

```
class Lenovo extends Laptop {
  constructor () {
    super(21);
  }
  aumentarBrilho(valor: number) {
    console.log(`Brilho subiu ${valor} pontos`);
  }
}
```

```
let computador = new Lenovo();

computador.ligarMonitor();
computador.aumentarBrilho(3);
```

Classes e Interfaces

Interfaces

Uma interfaces define um contrato que toda classe que a implemente é obrigada a seguir, a cumprir.

```
interface Gamer {
  memoriaVideo: number;
}
```

```
class Lenovo extends Laptop implements Gamer {
  memoriaVideo: number = 512;

  constructor () {
    super(21);
  }

  aumentarBrilho(valor: number) {
    console.log(`Brilho subiu ${valor} pontos`);
  }
}
```