

# 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');
```

O que acontece caso seja passado um número

# 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 interface 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`);  
  }  
}
```