Typescript

Arrays, estruturas de decisão e de repetição,
Operadores, funções e JavaScript Objects

Ely – ely.miranda@ifpi.edu.br

1

Arrays

- Um array é estrutura de dados simples que armazena valores indexada:

```
let numeros: number[] = [1, 2, 3];
```

- Para serem declarados, seu tipo deve ser seguido de colchetes: number[], string[].

2

Arrays

- Podem ter uma ou mais dimensão (matrizes);
- Normalmente possuem um único tipo;
- Seu primeiro elemento é o elemento cujo índice é zero.

3

Arrays

- O acesso a seus elementos é feito através de uma chave entre colchetes []:

```
numeros[1] = 5;
console.log(numeros[1]); //5
console.log(numeros[-10]); //undefined
```


4

Arrays

- Podem ser declarados como “arrays tipados”:

```
let numeros2: Array<number> = [1, 2, 3];
```

- Além disso, possuem alguns métodos:

```
numeros.push(4);
console.log(numeros.reverse()); // [4,3,2,1]
```


5

Estruturas de decisão – if e else

- Igual às linguagens C, Java, JavaScript:

```
if (condição) {
    //código da condição verdadeira;
}
else {
    //código da condição falsa;
}
```


6

Estruturas de decisão - Switch

```
let expressao: number = 1;
switch (expressao) {
    case 1:
        //código a ser executado se expressao = 1
        break;
    case 2:
        //código a ser executado se expressao = 2
        break;
    default: //executado caso expressão não é nenhum dos valores acima;
}
```


7

Estruturas de repetição - while

- Executa um trecho de código enquanto uma condição for verdadeira:

```
let numero: number = 1;
while (numero <= 3) {
    console.log("O número atual é: " + numero);
    numero = numero + 1;
}
```


8

Estruturas de repetição – do while

- Executa um trecho de código enquanto uma condição for verdadeira;
- Mesmo que a condição seja falsa, o código é executado pelo menos uma vez:

```
let numero: number = 1;
do {
    console.log("O número atual é: " + numero);
    numero = numero + 1;
} while (numero <= 3);
```


9

Estruturas de repetição - for

- Executa um trecho de código por uma quantidade específica de vezes:

```
let numeros = [4, 5, 6];
for (let i = 0; i < numeros.length; i++) {
    numeros[i] = numeros[i] * 2;
    console.log(numeros[i]); // 8, 10, 12
}
```


10

Let ou var

- Let possui um escopo mais limitado, e isso é positivo:

```
for (let i : number = 0; i < 3; i++)
    console.log(i); // 1, 2, 3
console.log(i); // erro de compilação
```


11

Let ou var

- Var possui um escopo mais amplo:

```
for (var i : number = 0; i < 3; i++)
    console.log(i); // 1, 2, 3
console.log(i); // 3
```


12

Const

- Também é possível declarar variáveis com as palavras reservadas var e const;
- Usando const, a variável não pode ter seu valor alterado:

```
const i: number = 0;
i = 3; // erro de compilação
```


13

Outros tipos de for

- Além da estrutura de for e while conhecidos em Java, há outros tipos de for:

```
let numeros = [4, 5, 6];
//faz a iteração pelos elementos sem uso de []
for (let numero of numeros) {
    console.log(numero); // 4, 5, 6
}
```


14

Outros tipos de for

- Além da estrutura de for e while conhecidos em Java, há outros tipos de for:

```
//faz a iteração pelos índices
for (let numero in numeros) {
    console.log(numero); // 0, 1, 2
}
```


15

Expressões compactas

- Em TypeScript também podemos utilizar formas “compactada” das instruções:

```
let numero : number = 100;
numero++;
numero--;
numero += 2;
numero -= 1;
numero *= 4;
numero /= 2;
console.log(numero); // 202
```


16

Operadores condicionais e lógicos

- Condicionais: >, >=, <, <=, ==, !=, ===, !==;
- Lógicos: &&, ||, !
- Operadores lógicos de atribuição: &&=, ||=, ??=
  - https://pt.stackoverflow.com/questions/485878/o-que-s%C3%A3o-os-operadores-de-atribui%C3%A7%C3%A3o-l%C3%B3gica-e-em-javascript

17

O operador ===

```
let a=10;
let b="10";
//em JS puro
console.log(a==b);  //true
console.log(a === b); //false
```


18

O operador ===

```
let a=10;
let b="10";
//em TypeScript
console.log(a==b);  //erro de compilação
console.log(a === b); // erro de compilação
```

https://stackoverflow.com/questions/57125700/why-use-triple-equal-in-typescript

Funções

- São blocos de instruções reutilizáveis e permitem a modularização;
- Podem receber parâmetros e entrada e retornar valores;
- Sintaxe básica:

```
function nomeDaFuncao(parametros): tipoDeRetorno {
    //instruções da função.
}
```

Funções

- Exemplo sem parâmetro e sem retorno:

```
function saudacao() : void {
    console.log("hello world!");
}
saudacao(); // hello world
```


Funções

- Exemplo com parâmetro e sem retorno:

```
function saudacao(nome: string) : void {
    console.log(`Hello, ${nome}!`);
}

console.log(saudacao('Ely'));
```

Funções

- Exemplo com parâmetros e com retorno:

```
function add(x: number, y: number): number {
    return x + y;
}
console.log(add(2,3)); //5
```


Funções como variáveis

```
let  a = function add(x: number, y: number): number {
    return x + y;
}
console.log(a(2,4)); //6
```

Parâmetros default

- São parâmetros que, se não forem repassados, terão um valor padrão:

```
function nomeCompleto(nome: string,
                      sobrenome : string = "silva"): string {
    return nome + " " + sobrenome;
}
console.log(nomeCompleto('ely', 'miranda')); // ely miranda
console.log(nomeCompleto('ely')); // ely silva
```


Parâmetros opcionais

- Os últimos parâmetros de uma função podem ser opcionais e são identificados por uma "?";
- São necessários testes para saber se parâmetros opcionais foram preenchidos:

```
function nomeCompleto(nome: string,
                      sobrenome? : string): string {
    if (sobrenome) {
        return nome + " " + sobrenome;
    }

    return nome;
}
```

Rest parameters

- Uma forma de se passar um número indeterminado de parâmetros na forma de array:

```
function somar(...numeros : number[]) {
     let soma = 0;
     for (let numero of numeros)
             soma += numero;
     return soma;
 }
 console.log(somar()); // 0
 console.log(somar(1,2)); // 3
 console.log(somar(1,2,3)); // 6
```


Arrow functions

- São funções com sintaxe simplificada;

```
function dobra(x : number) : number {
    return x*2;
}
```

- Em formato de arrow function:

```
var dobra = (x) => x*2;
console.log(dobra(2)); //4
```

Arrow functions

- Outro exemplo:

```
function soma(x : number, y: number) {
    return x+y;
}
```

- Em formato de arrow function:

```
var soma = (x,y) => x+y;
console.log(soma(2,4)); //6
```


29

Arrow functions

- É muito comum o uso com funções “iterativas” de algumas classes:

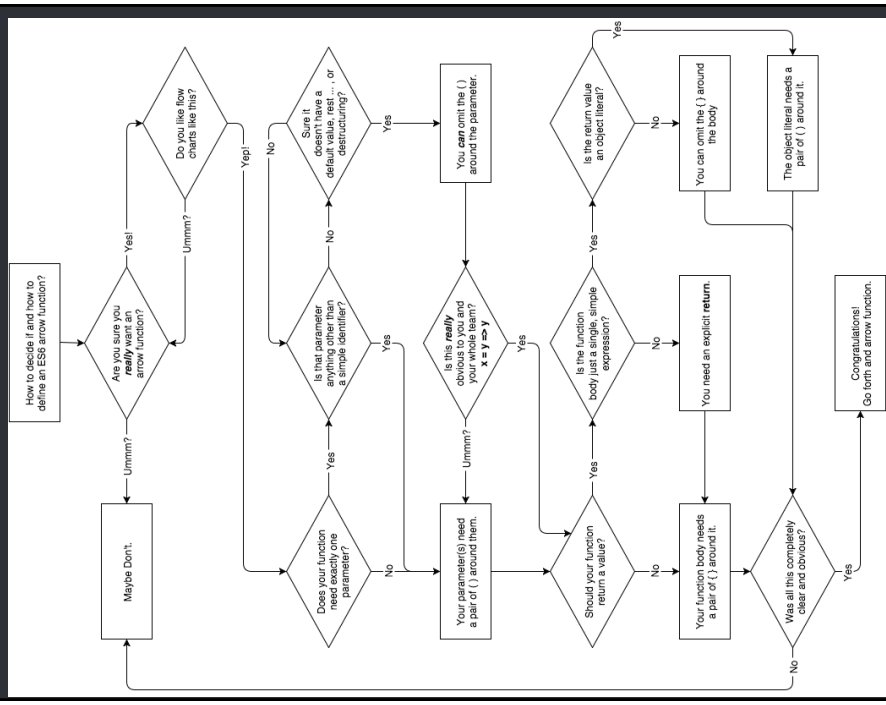
```
let numeros : number[] = [1,2,3,4];
numeros = numeros.map(x => 2*x);
console.log(numeros); //[2,4,6,8]
```


30

Quando usar uma arrow function

- https://www.freecodecamp.org/news/when-and-why-you-should-use-es6-arrow-functions-and-when-you-shouldnt-3d851d7f0b26/

31



32

32

Objetos JavaScript

- Em JS puro, são estruturas que permitem manter tipos de dados heterogêneos;
- Exemplos semelhantes são as structs de C records em Pascal;
- São definidas como estruturas “chave/valor” bem semelhantes aos elementos JSON.

33

Objetos JavaScript em TypeScript

```
// Definindo um objeto Pessoa
type Pessoa = {
    nome: string;
    idade: number;
    profissao: string;
};
```

```
// Criando um objeto Pessoa
const pessoa1: Pessoa = {
  nome: 'João',
  idade: 30,
  profissao: 'Engenheiro'
};
```


34

Objetos JavaScript em TypeScript

```
// Acessando propriedades do objeto
console.log(pessoa1.nome); // João
console.log(pessoa1.idade); // 30
console.log(pessoa1.profissao); // Engenheiro
```


35

Objetos JavaScript em TypeScript

- Nota:
  - Não confundir com o conceito de classes e objetos da P.O.O.
  - Voltaremos a tratar de classes e suas instâncias nas próximas aulas.

36

Alguns links

- https://www.typescriptlang.org/docs/handbook/2/everyday-types.html
- https://www.typescriptlang.org/docs/handbook/2/indexed-access-types.html

37

Typescript

Arrays, estruturas de decisão e de repetição,
Operadores, funções e JavaScript Objects

Ely – ely.miranda@ifpi.edu.br

38