FULL STACK WEB DEVELOPER

FERNANDO LIRA

FLAG

JAVASCRIPT

APRESENTAÇÃO – FERNANDO LIRA

it.fernandolira@gmail.com

https://www.linkedin.com/in/fernandolira74/

+351 93 317 99 21

@fernandolira74

this e a função bind()

```
const pessoa = {
    saudacao: "Bom dia!!",
    falar() {
        console.log(this.saudacao);
    }
}

pessoa.falar();  //Bom dia!!

const falar = pessoa.falar;

falar(); //undefined - conflito entre paradigmas: funcional e OO

const falarDePessoa = pessoa.falar.bind(pessoa);
falarDePessoa();
```

this e a função bind() – NaN – é o setInterval que invoca a função anónima e perde o this

```
function Pessoa() {
    this.idade = 0;

    setInterval(function() {
        this.idade++;
        console.log(this.idade);
    },1000);
}

new Pessoa;
```

this e a função bind() – solução 1

```
function Pessoa() {
    this.idade = 0;

    setInterval(function() {
        this.idade++;
        console.log(this.idade);
    }.bind(this),1000);
}

new Pessoa;
```

this e a função bind() – solução 2

```
function Pessoa() {
    this.idade = 0;

    const self = this;
    setInterval(function() {
        self.idade++;
        console.log(self.idade);
    }/*.bind(this)*/,1000);
}

new Pessoa;
```

Funções Arrow – 2 objetivos

- Sintaxe mais abreviada
- Ter um this associado ao contexto

Funções Arrow – Sintaxe abreviada

```
let dobro = function(a) {
    return a * 2;
}

//Equivalente como função Arrow
//A função arrow é sempre anónima!
dobro = (a) => {
    return a * 2;
}

//Para funções de uma única linha ainda se pode simplificar
dobro = a => 2 * a; //return implicito
```

Funções Arrow – Sintaxe abreviada

```
//Função anónima sem parametros
let ola = function() {
    return 'Olá';
}

//Equivalente como Arrow com () ou _
ola = () => 'Olá';
ola = _ =>'Olá'; //Possui um paramentro mas o jscript não obriga
a passar
```

Funções Arrow – this associado ao contexto

```
function Pessoa() {
    this.idade = 0;

    setInterval(() => {
        this.idade++;
        console.log(this.idade);
    },1000);
}

new Pessoa;
```

Funções Callback

```
const fabricantes = ["Mercedes", "Audi", "BMW"];

function imprimir(nome,indice) {
    console.log(`${indice+1}. ${nome}`);
}

//Funções Callback - são chamadas quando ocorre um evento!
//Neste caso, o evento é o loop de cada elemento do array!
fabricantes.forEach(imprimir);

//Alternativa com função Arrow
fabricantes.forEach(fabricante => console.log(fabricante));
//Ou mais completo
fabricantes.forEach((fabricante,indice) =>
console.log(`${indice+1}. ${fabricante}`));
```

Funções Callback

```
const notas = [12, 15, 7, 8, 9, 14, 5, 12];

//Sem callback
const notasBaixas = [];
for (let indice in notas)
    if (notas[indice] < 10)
        notasBaixas.push(notas[indice]);

console.log(notasBaixas);
```

Funções Callback

```
//Com callback
const notasBaixas2 = notas.filter(function(nota) {
    return nota < 10;
});

console.log(notasBaixas2);


//Com callback e arrow
const notasBaixas3 = notas.filter( nota => nota < 10);

console.log(notasBaixas3);
```

Funções Callback

```
//Com callback, arrow e retutilização de função
const notasMenorQue10 = nota => nota < 10;
const notasBaixas4 = notas.filter(notasMenorQue10);

console.log(notasBaixas4);
```

IIFE - Imediate Invoked Function Expression

```
(function exemplo() {
    console.log("Qualquer coisa");
})();

const variavel = function() {
    console.log("Outra coisa");
}();
```

CALL e APPLY

```
function getPreco(imposto = 0, moeda = '€') {
    return `${moeda} ${this.preco * (1-this.desconto) * (1 +
imposto)}`
}

console.log(getPreco()); //€ NaN

const produto = {
    nome: 'Portátil',
    preco: 750,
    desconto: 0.1,
    getPreco
}

console.log(produto.getPreco()); //€ 675
```

CALL e APPLY

```
const carro = {
    preco: 5000,
    desconto: 0.2
}

console.log(getPreco.call(carro)); //€ 4000
console.log(getPreco.apply(carro)); //€ 4000

console.log(getPreco.call(carro,0.2,'$')); //$ 4800
console.log(getPreco.apply(carro,[0.2,'$'])); //$ 4800
```