Funções

* **First-class Object (citizens)**
  + **Linguagem trata a função como um dado**
* **Higher-order function**
* **função pode ser tratada como um dado**
* **Pode passar ela como um parâmetro**
* **Consegue retornar uma função como resposta de outra**
* **Pode armazena-la dentro de uma variável**

1. **Como declarar uma função de forma literal:**

function nomeVariavel(*parametros*) {

    // instruções

}

* **Podem ou não receber parâmetros**
* **Qualquer variável criada dentro dela não pode ser utilizada fora**
* **Pois a função tem escopo**
* **Pode ou não retornar um valor - se não colocar nada retorna undefined**
* **Não pode omitir o bloco { }**

return exemplo

* **Função para de ser executada quando utilizada o return**
* **Retorna o valor que vem após dele**

**2. Armazenar em uma variável**

const amarzenandoNaVariavel = function() {

}

* **atribui uma função anônima a uma variável**

const amarzenandoNaVariavel()

**invoca a função**

**3. Armazenar uma função dentro de uma array**

const nomeArray = [function (*a*, *b*) {return *a* + *b*}, nomeDaVariavel, amarzenandoNaVariavel]

**Executar a função**

console.log(nomeArray[0](2, 3))

* **pegou a primeira função dentro da array (primeiro índice) e atribuiu 2 e 3 como parâmetro e retornou 5**

**4. armazenar uma função dentro de atributos de objeto**

const objeto = {}

objeto.falar = function () {return 'Oi'}

console.log(objeto.falar())

**5. Passar uma função como parâmetro para outra função**

function run(*fun*) {

    fun()

}

run(function () {console.log('executando...')})

**6. Uma função pode retornar/conter uma função**

function soma(*a*, *b*) {

    return function(*c*) {

        console.log(*a* + *b* + *c*)

    }

}

soma(2, 3)(4) // retornando a função ou

const cincoMais = soma(2, 3)

cincoMais(4) // retorna a mesma coisa

**Parâmetros e retornos são opcionais**

function area(*largura*, *altura*) {

    const area = *largura* \* *altura*

    if(area > 20) {

        console.log(`Valor acima do permitido ${area}m2`)

    } else {

        return area

    }

}

* **Retorna o valor apenas quando a area é menor que 20**
* **Exemplo de como a função pode ou não retornar um valor**

**Parâmetros variáveis**

* **arguments é uma array interna de uma função que tem todos os argumentos(parametros) que foram passados**
* **arguments era utilizado antes da versão EctmaScript 2015**

function soma() {

    let soma = 0;

    for (i in *arguments*) {

        soma += *arguments*[i] // pega todos os itens e soma

    }

    return soma

}

console.log(soma(1))

console.log(soma(1.1, 2.3, 3.4)) // pode passar parâmetros variáveis

console.log('teste 1', 'teste 2') // apenas concatena

**Parâmetros padrão**

**Forma antiga de fazer:**

function soma1(*a*, *b*, *c*) {

*a* = *a* || 1

*b* = *b* || 1

*c* = *c* || 1

    return *a* + *b* + *c*

}

console.log(soma1(), soma1(3), soma1(0, 0, 0))

// retorna falso e não funciona

// estratégia 2, 3 e 4

function soma2(*a*, *b*, *c*) {

*a* = *a* !== undefined ? *a* : 1

*b* = 1 in *arguments* ? *b* : 1

*c* = isNaN(*c*) ? 1 : *c // versão mais segura*

}

**Forma mais recomendada de fazer:**

// valor padrão do es2015

function soma3(*a* = 1, *b* = 2, *c* = 1) {

    return *a* + *b* + *c*

}

console.log(soma3(3), soma3(2,1), soma3(0))

* **caso seja passado parâmetros apenas para algumas, elas assumem o valor padrão**

**This**

* **Pode referenciar o objeto atual da execução**
* **This pode variar dependendo de como a função é chamada**
* **Outro motivo pra ter uma função arrow é porque o this não varia**
* **Quando a função for inscrita dentro do contexto global o this sempre vai apontar pro objeto window, se for uma função arrow**

**Bind**

* **Serve para “amarrar” um determinado objeto pra ele ser o dono da execução sempre que o método for chamado**
* **A função não é alterada com a chamada de bind**

const pessoa = {

    saudacao: 'Bom dia!',

    falar() {

        console.log(*this*.saudacao)

    }

}

pessoa.falar() // mostra o que está em saudacao

const falar = pessoa.falar // armazenou a funcao na constante

falar() // conflito entre paradigmas: Funcional e OO

const falarDePessoa = pessoa.falar.bind(pessoa)

falarDePessoa()

**Função Arrow**

* **É sempre uma função anônima**
* **É vantajoso criar arrow functions para fazer coisas pequenas, são específicas, e fazem uma única coisa**

dobro = *a* => 2 + *a*

ola = () => 'Olá'

ola = *\_* => 'Olá'

let comparaComThis = function (*param*) {

    console.log(*this* === *param*)

}

comparaComThis(globalThis) // retorna true

* **O this varia numa função normal, já na arrow, o this não aponta pro objeto global**

let comparaComThisArrow = *param* => console.log(*this* === *param*)

// função arrow não aponta pro global, mas função normal sim

comparaComThis(globalThis)

comparaComThisArrow = comparaComThisArrow.bind(obj)

comparaComThisArrow(obj) // arrow function é mais forte que bind

comparaComThis(*module*.*exports*) // acessa se é o this no nodejs

comparaComThisArrow = comparaComThisArrow.bind(obj)

comparaComThisArrow(obj)

// retorna false

// arrow function é mais forte que bind

**Funções anônimas**

* **Funções que não tem nome**
* **Armazenadas em variáveis**
* **Podem ser armazenadas em atributos de um objeto**

**Função Callback**

* **“Chamar de volta”**
* **Passar uma função pra outra função, quando determinado evento acontecer ela é chamada**

const notas = [7.7, 6.5, 5.2, 8.9, 3.6, 7.1, 9.0]

notasBaixas = notas.filter()

**filter retorna verdadeiro ou falso, pode ser passada uma função dentro dele**

notasBaixas = notas.filter(function (*nota*) {

    return *nota* < 7

})

**Se houverem notas menor que 7, ele adiciona-as a notasBaixas. Mas não altera a array notas**

const notasBaixas3 = notas.filter(*nota* => *nota* < 7) //utilizando arrow

console.log(notasBaixas3)

**mais simples com arrow function**

const notasMenorQue7 = *nota* => *nota* < 7 // para poder utilizar em mais lugares

const notasBaixas3 = notas.filter(notasMenorQue7)

document.getElementsByTagName('body')[0].onclick = function (*e*) {

    console.log('O evento ocorreu!')

}

**Exemplo de uso no browser, quando o usuário clica no body dispara a função call-back**

**Funções Construtoras**

* **É uma Função molde, a partir dela podem ser criadas instâncias de objetos**
* **Iniciadas em letra maiúscula**
* **Utiliza o this pra deixar algo visível para fora da função, o let é utilizado quando queremos que o atributo seja privado**

function Ventilador(*velMax*) {

*this*.velocidadeMaxima = *velMax*; // this é utilizado para criar um objeto

*this*.ligado = false

**Como criar instâncias**

* **Para criar uma instancia de um objeto usa-se a palavra reservada “new” seguida da chamada da função**

const ventilador1 = **new** Ventilador(3) // velMax passa a ser 3

* **Armazena em uma constante**
* **Pode ou não utilizar os parênteses caso não tenha parâmetros**
* **Quando isso é feito, é criado um novo objeto com as propriedades do primeiro objeto**
* **Consegue acessar sua propriedade da seguinte forma:**

console.log(ventilador1.velocidadeMaxima)

**utilizando um ponto e após, a propriedade que quer acessar**

ventilador1.cor = "Branco" // atribuindo propriedades ao novo objeto

* **Também pode atribuir uma função dentro da propriedade do objeto**

ventilador1.ligaDesliga = function() {

    if(*this*.ligado)

*this*.ligado = false

    else

*this*.ligado = true

}

**IIFE – Immediately Inoveked Function Expression**

* **Função autoinvocável**
* **Tem a função de fugir do escopo global**
* **Tem que estar entre parênteses**
* **Qualquer elemento criado, só estará disponível dentro da função**

(function () {

    console.log('Será executado na hora!')

    console.log('Foge do escopo mais abrangente')

})()

**Call e Apply**

* **A diferença entre as duas está na forma em que são passado os parâmetros**
* **Primeiro é passado o contexto da execução e depois os parâmetros**
* **Na call é passado entre virgulas, e na apply os parâmetros são passados dentro de uma array**

**Call**

console.log(getPreco.call(carro)) // utiliza os parâmetros padrão ou

console.log(getPreco.call(carro, 0.17, '$'))

**Apply**

console.log(getPreco.apply(carro))

console.log(getPreco.apply(carro, [0.17, '$']))