Seção 4: JavaScript Funções (Avançado)

* Aula 49 maneiras de declarar funções

// Declaração de função (Function Hoisting)

falaOi();    //pode ser chamado tanto antes como depois

function falaOi(){

    console.log('Oi');

}

// First-class objects (Objetos de primeira classe)

// Function expression

const souUmDado = function() {

    console.log('Sou um dado');

};

function executaFuncao(funcao) {

    funcao();

}

executaFuncao(souUmDado)

// Arrow function

const funcaoArrow = () => {

    console.log('Sou uma arrow function');

};

funcaoArrow();

// Dentro de um objeto

const obj = {

    falar(){

        console.log('Estou falando...');

    }

}

obj.falar();

uma variável pode se tornar uma função.

* Aula 50 Parâmetro de Funções

function funcao(){

    console.log(arguments);

}

funcao('Valor', 1, 2, 3, 4, 5);

[Arguments] {

 '0': 'Valor', '1': 1, '2': 2, '3': 3, '4': 4, '5': 5 }

quando chamado uma função com function, os argumentos são passados para o atributo arguments.(com arrow function não funciona)

function funcao(){

    let total = 0;

    for(let argumento of arguments){

        total += argumento

    }

    console.log(total);

}

funcao(1, 2, 3, 4, 5, 6, 7);

esses argumento podem ser acessados

function funcao(a, b, c, d, e, f){

    console.log(a, b, c, d, e, f);

}

funcao(1, 2, 3);

quando passado menos argumentos do que paramentro e esses parâmetro forem iniciado, os que não tiverem valor serão undefined

function funcao(a, b, c, d, e, f){   // menos argumentos

 que parametro retorna undefied

    console.log(a, b, c, d, e, f);   // naqueles que não foram passado

}

funcao(1, 2, 3);

menos argumento que parâmetros, so retorna os valores passado

function funcao(a, b = 2, c = 4){ // b como padrão será

  2, caso não haja algum

    console.log(a + b + c);

}

funcao(2, undefined, 20);  // para deixar o valor padrão de b

para que o valor assuma o seu valor padrão e enviado o undefined

function funcao({ nome, sobrenome, idade }){

    console.log(nome, sobrenome, idade);

}

funcao( { nome:'Luiz', sobrenome:'Otávio', idade: 30 });

função por desestruturação

function funcao({ nome, sobrenome, idade }){

    console.log(nome, sobrenome, idade);

}

let obj = { nome: 'Luiz', sobrenome: 'Otavio', idade: 30 };

console.log(obj);

ou pode ser enviado direto com o objeto

function funcao([valor1, valor2, valor3]) {

    console.log(valor1, valor2, valor3);

}

funcao(['Luiz Otávio', 'Miranda', 30])

passando por Array

function conta(operador, acumulador, ...numeros){  // rest opera

tor - sempre deve ser o ultimo

    for(let numero of numeros){

        if(operador === '+') acumulador += numero;

        if(operador === '-') acumulador -= numero;

        if(operador === '\*') acumulador \*= numero;

        if(operador === '/') acumulador /= numero;

    }

    console.log(acumulador);

}

conta('+', 1, 20, 30, 40 ,50);

const conta = (...args) => {   // para pegar os argumentos com

arrow function

    console.log(args);

}

conta('+', 1, 20, 30, 40, 50);

para conseguir pegar os argumentos com arrow function

* Aula 51 Retorno da função

// return

// Retorna um valor

// Termina a função

function soma(a , b){

    return a + b; // retorna algo

}

let s1 = soma(2, 5);

function soma2(a, b){

    console.log(a + b); // Não retorna nada

}

soma2(5, 2);

function criaPessoa(nome, sobrenome){

    return { nome: nome, sobrenome: sobrenome };

}

const p1 = criaPessoa('Luiz', 'Otavio');

const p2 = {

    nome: 'João',

    sobrenome: 'Oliveira'

};

console.log(p1);

console.log(p2);

mesmo retorno

function falaFrase(comeco) {

    function falaResto( resto){

        return comeco + ' ' +resto;

    }

    return falaResto;

}

const olaMundo = falaFrase('Olá');

console.log(olaMundo('Mundo!'));

retorno encadeado um dentro do outro.

function duplica(n){

    return n \* 2;

}

function triplica(n){

    return n \* 3;

}

function quadriplica(n){

    return n \* 4;

}

console.log(duplica(2));

console.log(triplica(2));;

console.log(quadriplica(2));

veja esse caso de código que poderia ser encadeados.

function criaMultiplicador(multiplicador) {

    return function(n) {

        return n \* multiplicador;

    }

}

const duplica = criaMultiplicador(2);

const triplica = criaMultiplicador(3);

const quadriplica = criaMultiplicador(4);

console.log(duplica(3));

console.log(triplica(2));

console.log(quadriplica(10));

ficaria dessa maneira

* Aula 52 escopo léxico

const  nome = 'Luiz';

function falaNome() {

    console.log(nome);

}

function usaFalaNome(){

    const nome = 'Otavio';

    falaNome();

}

usaFalaNome();

* Aula 53 closures

// Habilidade que a função tem de acessar seu escopo léxico

function retornaFuncao(nome) {

    return function () {

        return nome;

    }

}

const funcao = retornaFuncao('Luiz');

const funcao2 = retornaFuncao('João');

console.dir(funcao);

console.dir(funcao2);

console.log(funcao(), funcao2());

* Aula 54 função calback

function rand( min = 1000, max = 3000) {

    const num = Math.random() \* (max - min) + min;

    return Math.floor(num);

}

function f1() {

    setTimeout(function() {

        console.log('f1');

    }, rand())

}

function f2() {

    setTimeout(function (){

        console.log('f2');

    }, rand())

}

function f3() {

    setTimeout(function(){

        console.log('f3');

    }, rand())

}

f1();

f2();

f3();

console.log('Olá mundo');

funções que geram um tempo aleatório para serem exibidas, porem o ola mundo sempre será o primeiro a ser executado, pois não depende de nenhum tempo.

function rand( min = 1000, max = 3000) {

    const num = Math.random() \* (max - min) + min;

    return Math.floor(num);

}

function f1(callback) {

    setTimeout(function() {

        console.log('f1');

        if (callback) callback();

    }, rand())

}

function f2(callback) {

    setTimeout(function (){

        console.log('f2');

        if (callback) callback();

    }, rand())

}

function f3(callback) {

    setTimeout(function(){

        console.log('f3');

        if (callback) callback();

    }, rand())

}

f1(function() {

    f2(function() {

        f3(function(){

            console.log('Olá mundo');

        });

    });

});

Temos aqui uma função de callback, que so sera exibida quando a anterios for acabar de ser executada.

Ou pode ser assim

function rand( min = 1000, max = 3000) {

    const num = Math.random() \* (max - min) + min;

    return Math.floor(num);

}

function f1(callback) {

    setTimeout(function() {

        console.log('f1');

        if (callback) callback();

    }, rand())

}

function f2(callback) {

    setTimeout(function (){

        console.log('f2');

        if (callback) callback();

    }, rand())

}

function f3(callback) {

    setTimeout(function(){

        console.log('f3');

        if (callback) callback();

    }, rand())

}

f1(f1Callback);

function f1Callback() {

    f2(f2Callback);

}

function f2Callback() {

    f3(f3Callback);

}

function f3Callback() {

    console.log('Olá mundo');

}

* Aula 55 funções imediatas (IIFE) ou funções autoinvocadas

//IIFE -> Immediately invoked function expression

function qualquerCoisa(){

    console.log(123456);

}

qualquerCoisa()

aqui essa função toca o escopo global quando ela é chamada.

//IIFE -> Immediately invoked function expression

(function() {

    console.log(123456);

})()

Uma função que não toca o escopo global, ficando protegido. Pois fica sendo um função anônima.

//IIFE -> Immediately invoked function expression

(function(idade, peso, altura) {

    const sobrenome = 'Miranda';

    function criaNome(nome){

        return nome + ' ' + sobrenome;

    }

    function falaNome(){

        console.log(criaNome('Luiz'));

    }

    falaNome();

    console.log(idade, peso, altura);

})(30, 80, 1.80);

Como ficaraia essa função protegida que não toca o escopo global

* Aula 56 funções fabrica (factory function)