O principal objetivo desse curso é mostrar as formas de declarar funções, manipular parâmetros, apresentar loops e outros argumentos, e apresentar o argumento “this”.

A estrutura da função principal é a seguinte:

*Function name(parameter){instructions};*

**Variáveis dentro da função só pode ser usado dentro dela**. quando invocamos o return dentro de uma função a **função cessa**, retornando um valor após dele se ela tiver.

Funções **anonimas** são funções que representam expressões, elas não tem nome e são chamadas por uma **variável que a contêm**:

*Const soma = function (a,b){return a + b;}*

*Soma (1,2); //3*

Funções **autoinvocaveis**, conhecida comumente pelo termo **IIFE** (immediately invoked function expression), é uma função anônima entre parenteses seguida de parenteses que a chamam *and I ask w h y ?*. ela também pode ser utilizada como parâmetros ou armazenadas em uma variável.’’

*(function() { const soma=(function(){*

*Let name = “pog” return a+b})(1,2);*

*Return name;})(parametro); console.log(soma);*

Funções de **callback** é uma função chamada como argumento para outra função, dando um maior controle da ordem de chamadas de funções.

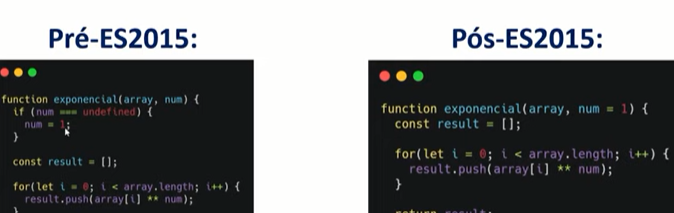
*Const calc = function(operação,num1,num2){return operação(num1,num2);}*

*Const soma = function(num1,num2){return num1+num2;}*

*Const sub = function(num1,num2){return num1-num2;}*

*Const resultsoma=calc(soma,1,2);*

*Const resultsub=calc(sub,1,2);*

**Parametro** são os comandos que dados a uma função que são utilizados durante a mesma, aquilo entre parenteses que vemos logo em cima na ultima linha, antes era um pouco complexo para definir um parâmetro, hoje em dia é um pouco mais facil de adicionar um **parâmetro padrão** para o caso da pessoa não mandar um input.

O **objeto ‘argument’** é um array com todas as parâmetros passados quando a função é invocada, assim, ele da uma lista de **todos os argumentos, com o intúito de verificar os diversos argumentos dentro do mesmo**.

O comando **spread** usado em arrays é uma forma de lidar separadamente com os elementos de uma array, tornando seus elementos independentes, utilizando o sinal de “**...**”.

*Function sum (x,y,z){return x + y + z};*

*Const numbers = [1,2,3]; console.log(sum(****…****numbers));*

O contrario acontece quando utiliza-se o comando **rest**, que combina diversos elementos independentes dentro de uma array, utiliza do mesmo sinal de “**...**” antes do argumento.

*Function conferirTamanho(...args){console.log(args.length)};*

*conferirTamanho(3,5);//2*

**object destructioning** é um método que filtra dados específicos de um objeto ao se utilizar dentro de chaves {}.

*Const user(){id:42,displayName:’joe’,fullName{firstName:’John’,lastName:’Doe’}};*

*Function userId(****{****id****}****){return id;}*

*Function getFullName({fullName:{firstName:first,lastName:last}}){return `${first} ${last}`;}*

*userId(user);getFullName(user);*

**Loops** são um dos grandes focos das linguagens de programação em geral.

**If/else** é uma estrutura de “se/ou” que serve para dividir as instruções de acordo com as possibilidades de respostas. Nem sempre é necessário o else, ao utilizar o comando **return**, a função se fecha automaticamente, então é possivel **caso haja mais apenas uma possibilidade de acontecimento e não há mais nada no código**, colocar um return com a ultima instrução após o } do if em vez de todo um else isso parece pouco, mas isso é menos uma locação na memória do computador, salvando recursos e agilizando o processo.

*If(a=b){result=true;}else{result=false};return result;*

Também é possivel fazer um novo if como parte do else (que é basicamente o que usaremos o else para fazer), note que o **javascript não possui elseif, aqui escrevemos else if sendo esse if um novo if**.

*If(ehNegativo){return “é negativo!”;}*

*else if(!ehNegativo && maiorQueDez){return “é positivo e maior que 10!;}*

*Return “esse numero é positivo!”*

**O switch/case** é uma comparação de tipo e valor (===), sempre precisa de um valor **default**, e é ideal para comparar muitos valores é considerado “código limpo” ao trabalhar com inputs de tecla do usuário.

*Function getAnimal(id){switch(id){*

*case 1: return “cão”;case 2: return “gato”;*

*case 3: return “monkey”;* ***default****: return ”peixe”}}*

*getAnimal(1);//cão*

*getAnimal(“1”);//peixe como não foi retornado um id valido (ou seja o número) ele trata como* ***default****, que nesse caso é peixe.*

**For** é um loop dentro de elementos iteráveis como arrays e strings, ele **vai repetir a instrução enquanto o argumento** (comumente nomeado “i”) **não foi validado**.

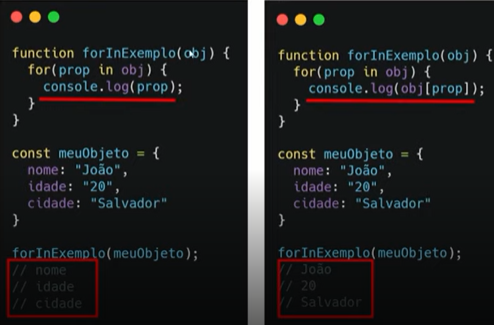
*Function multiplicarPorDois(arr){let multiplicar=[]);*

*for(let i = o; i < arr.length; i++){*

*multiplicar.push(arr[i]\*2);}}*

*return multiplicar;*

*const Numeros = [1,2,3,4,5,6,7,8]; multiplicarPorDois(Numeros);*

o **for in** é um loop entre propriedades enumeráveis de um objeto, é um loop de **enquanto dentro (do objeto x)**.

O **for of** é um loop entre estruturas iteráveis, é um loop de **enquanto algo do local**.

*Function logLetra(palavra){*

*For(letra of palavra){console.log(letra);}*

*Const palavra = “exemplo”;*

*logLetra(palavra);*

o **while** é um loop que **executa a instrução ate que ela se torne falsa**.

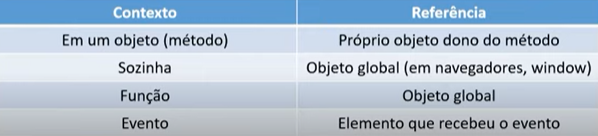
*While (num <= 5){console.log num;num++;}*

O **do while** executa a instrução até que se torne falsa **mas sempre executando-a ao menos uma vez**.

*Do{console.log(num);num++;}while(num>=5);*

O **this** é uma palavra reservada referente ao contexto, ao utilizar o this dentro de um método, então o this **vai se referir ao objeto pai desse método**. Em outras palavras, **seu valor pode mudar de acordo com o lugar do código onde foi chamado** a não ser que a gente mude o valor manualmente.

*Const pessoa ={firstName:”andre”,lastName:”soares”,id:1,fullName:function(){*

*Return this.firstName + “ “ + this.lastName;},*

*getId: function(){return this.id;}};*

*pessoa.fullName(); //andre soares*

*pessoa.getId(); //1*

é possivel manipular o valor do this, um desses métodos é o **call** que permite enviar um this de outro lugar para dentro de um outro local pense no sentido de “chamar fulano para cá”.

*Const pessoa = {nome:’miguel’};*

*Const animal={nome:‘murphy’};*

*Function getSomething(){console.log(****this****.name);}*

*getSomething.call(animal); // vai dar call no* ***this de dentro do objeto animal****.*

Também é possivel passar parametros para uma função separando-os por virgula.

*Const myObj={num1:2,num2:4,};*

*Function soma(a,b){console.log(this.num1 + this.num2 + a + b);}*

*Soma.call(myObj,1,5); //12*

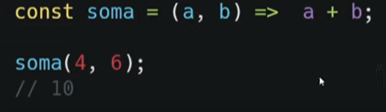
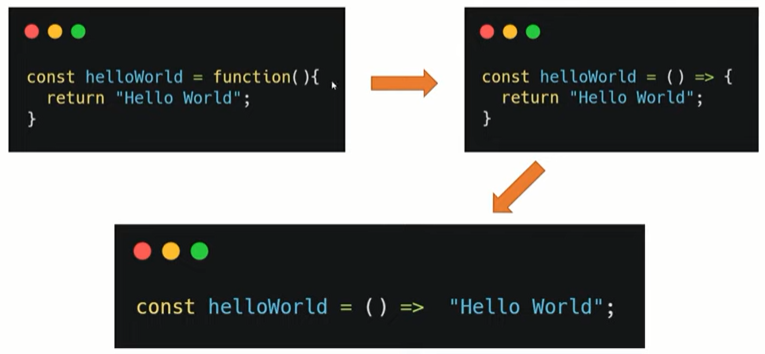
O comando **apply** tem um funcionamento parecido com o call, ate tendo o mesmo funcionamento em algumas situações, porem **quando passar argumentos, estes vão ser passados dentro de um array,** em vez de virgulas como no call agora qual a utilidade disso exatamente....

O método **bind clona a estrutura de uma função onde é chamada e aplica o valor do objeto ao parâmetro passado**.

*Const retornaNomes= function(){return this.nome};*

*Let Bruno=retornaNomes.bind({nome:’bruno’});*

*Bruno();*

******Arrow functions** é a seta =>, sendo **equivalente**, quando em uma linha, ele não precisa fazer **return** nem declarar função entre chaves, fazendo com que uma função pequena possa ser feita em uma linha curta como no exemplo. **Ela não faz hoisting!** algumas coisas não podem ser feitas usando a arrow function:

**-** This sempre sera global, métodos que modificam seu valor não funcionar.

- Não existe o objeto “arguments”.

- O constructor *(ex:new meuObjeto())* também não irá funcionar.

**Não faça arrow functions sem ser em métodos de objetos!** Pois isso pode causar problemas.

Agora para atividade prática.